

# SEKE 2014

**Proceedings of the Twenty-Sixth  
International Conference on  
Software Engineering &  
Knowledge Engineering**

Vancouver  
July 1-3, 2014

**PROCEEDINGS**  
**SEKE 2014**

**The 26<sup>th</sup> International Conference on  
Software Engineering & Knowledge  
Engineering**

**Sponsored by**

Knowledge Systems Institute Graduate School, USA

**Technical Program**

**July 1 – 3, 2014**

**Hyatt Regency, Vancouver, Canada**

**Organized by**

Knowledge Systems Institute Graduate School, USA

Copyright © 2014 by 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: 1-891706-35-7

ISSN: 2325-9000 (print)

2325-9086 (online)

Additional copies can be order from:

Knowledge Systems Institute Graduate School

3420 Main Street

Skokie, IL 60076 USA

Tel: +1-847-679-3135

Fax: +1-847-679-3166

Email: [seke@ksiresearch.org](mailto:seke@ksiresearch.org)

Web: <http://www.ksi.edu>

Proceedings preparation, editing and printing are sponsored by Knowledge Systems Institute Graduate School, USA. Some computers and projectors for conference presentation are provided by Department of Electrical and Computer Engineering, University of Calgary, Canada and University of British Columbia, Canada.

Printed by Knowledge Systems Institute Graduate School

# FOREWORD

It is my genuine pleasure to welcome everyone to the 26th International Conference on Software Engineering and Knowledge Engineering (SEKE) in the beautiful city of Vancouver, British Columbia, Canada. As an international conference we just passed a quarter of century, and look forward for next quarters to come. 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 Co-Chairs and the entire Program Committee, I would like to extend to you the warmest welcome to SEKE 2014.

We received 229 submissions from 33 countries this year. Through a rigorous review process where a majority (85 percent) of the submitted papers received three reviews, and the rest with two reviews, we were able to select 67 full papers for the general conference (29 percent), and 69 short papers (30 percent). Out of that 11 papers have been accepted for two special tracks. All papers are scheduled for presentation in forty-five sessions during the conference. In addition, the technical program includes excellent keynote speech, poster and demo presentations, as well as two special tracks: Software Assurance, and Knowledge Management in Software Engineering.

The high quality of the SEKE 2014 technical program would not have been possible without the tireless effort and hard work of many individuals. First of all, I would like to express my sincere appreciation to all the authors whose technical contributions have made the final technical program possible. I am very grateful to all the Program Committee members whose expertise and dedication made my responsibility that much easier. My gratitude also goes to the keynote speaker and panelists 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, I owe a debt of gratitude to a number of people whose help and support with the technical program and the conference organization are unflinching and indispensable. I am 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 2014. My heartfelt appreciation goes to Dr. Swapna Gokhale of University of Connecticut, USA, the Conference Chair, for her help and experience, and to the Program Committee Co-Chairs, Dr. Haiping Xu of University of Massachusetts Dartmouth, USA, Dr. Claudia Werner of Federal University of Rio de Janeiro, Brazil, and Kehan Gao of Eastern Connecticut State University, USA, for their outstanding team work.

I am truly grateful to the special track organizers, Dr. Dianxiang Xu of Boise State University, USA, Dr. Haiping Xu of University of Massachusetts Dartmouth, USA, Dr. Meira Levy of Shenkar College of Engineering and Design, Israel, Dr. Irit Hadar of University of Haifa, Israel, and Dr. Sivan Rapaport of Marketing Division Columbia Business School, USA, for their excellent job in organizing the special sessions.

I would like to express my great appreciation to all the Publicity Chair, Dr. Xiaoying Bai of Tsinghua University, China, for her important contributions, to the Asia, Europe, India, and South America liaisons, Dr. Hironori Washizaki of Waseda University, Japan, Dr. Raul Garcia Castro of Universidad Politecnica de Madrid, Spain, Swapan Bhattacharya, NITK, Surathakl, India, and Dr. Jose Carlos Maldonado of University of Sao Paulo, Brazil, for their great efforts in helping expand the SEKE community, and to the Poster/Demo session Co-Chairs, Dr. Farshad Samimi of Trilliant, USA, and Dr. Dragutin Petkovic of San Francisco State University, USA, for their work.

Last but certainly not the least, I must acknowledge the important contributions the following KSI staff members have made. Their timely and dependable support and assistance throughout the entire process have been truly remarkable. It has been a great pleasure to work with all of them. Finally, I hope you will enjoy both the scientific part of SEKE 2014 – exciting exchange of ideas related to software and knowledge engineering topics – and the beauty of Vancouver, British Columbia, Canada, one of the most livable cities in the world.

Marek Reformat, SEKE 2014 Program Chair

# SEKE 2014

## The 26<sup>th</sup> International Conference on Software Engineering & Knowledge Engineering

July 1 – 3, 2014

Hyatt Regency Hotel, Vancouver, Canada

### Conference Organization

#### CONFERENCE CHAIR

Swapna Gokhale, Univ. of Connecticut, USA

#### PROGRAM COMMITTEE CHAIR

Marek Reformat, University of Alberta, Canada

#### PROGRAM COMMITTEE CO-CHAIRS

Haiping Xu, University of Massachusetts Dartmouth, USA  
Claudia Werner, Federal University of Rio de Janeiro, Brazil  
Kehan Gao, Eastern Connecticut State University, USA

#### 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  
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, Memorial University of Newfoundland, Canada  
Taiseera Albalushi, Sultan Qaboos University, Oman  
Omar El Ariss, Penn State Univ at Harrisburg, USA  
Doo-hwan Bae, Korea Advanced Institute of Science and Technology, Korea  
Ebrahim Bagheri, National Research Council Canada, Canada  
Hamid Bagheri, University of Virginia, USA  
Xiaoying Bai, Tsinghua University, China  
Purushotham Bangalore, University of Alabama at Birmingham, USA  
Fevzi Belli, University of Paderborn, Germany  
Ateet Bhalla, Oriental Institute of Science & Technology, Bhopal, India  
Swapan Bhattacharya, NITK, Surathakl, India  
Alessandro Bianchi, University of Bari, Italy  
Ivo Bukovsky, Czech Technical University in Prague, Czech Republic  
Jaelson Castro, Universidade Federal de Pernambuco - UFPE, Brazil  
Raul Garcia Castro, Universidad Politecnica de Madrid, Spain  
Keith Chan, Hong Kong Polytechnic University, Hong Kong  
Kuang-nan Chang, Eastern Kentucky University, USA  
Ned Chapin, InfoSci Inc., USA  
Meiru Che, University of Texas at Austin, USA  
Shu-Ching Chen, Florida International University, USA  
Wen-Hui Chen, National Taipei University of Technology, Taiwan  
Zhenyu Chen, Nanjing University, China  
Stelvio Cimato, University of Milan, Italy Peter Clarke,  
Florida International University, USA Esteban Clua,  
Universidade Federal Fluminense, Brazil  
Nelly Condori-fernandez, University of Twente, The Netherlands  
Fabio M. Costa, Instituto de Informatica, Brazil  
Maria Francesca Costabile, University of Bari, Italy  
Jose Luis Cuadrado, University of Alcala, Spain  
Aldo Dagnino, ABB, USA  
Jose Luis De La Vara, Simula Research Laboratory, Norway  
Massimiliano Di Penta, University of Sannio, Italy  
Scott Dick, University of Alberta, Canada Junhua  
Ding, East Carolina University, USA  
Jing Dong, University of Texas at Dallas, USA  
Weichang Du, University of New Brunswick, Canada  
Philippe Dugerdil, HEG - Univ. of Applied Sciences, Switzerland  
Christof Ebert, Vector Consulting Services, Germany  
Ali Ebnehasir, Michigan Technological University, USA  
Raimund Ege, NIU, USA  
Magdalini Eirinaki, San Jose State University, USA  
Davide Falessi, University of Rome, TorVergata, Italy  
Behrouz Far, University of Calgary, Canada  
Liana Fong, IBM, USA  
Ellen Francine Barbosa, University of Sao Paulo, Brazil  
Fulvio Frati, University of Milan, Italy  
Jerry Gao, San Jose State University, USA  
Felix Garcia, University of Castilla-La Mancha, Spain  
Ignacio Garcia Rodriguez De Guzman, University of Castilla-La Mancha, Spain  
Vahid Garousi, University of Calgary, Canada  
Swapna Gokhale, Univ. of Connecticut, USA  
Wolfgang Golubski, Zwickau University of Applied Sciences, Germany  
Desmond Greer, Queen's University Belfast, United Kingdom  
Christiane Gresse Von Wangenheim, UFSC - Federal University of Santa Catarina, Brazil

Katarina Grolinger, University of Western Ontario, Canada  
 Irit Hadar, University of Haifa, Israel  
 Hassan Haghighi, Shahid Beheshti University, Iran  
 Hao Han, National Institute of Informatics, Japan  
 Xudong He, Florida International University, USA  
 Miguel Herranz, University of Alcala, Spain  
 Rubing Huang, Huazhong University of Science and Technology, China  
 Bassey Isong, University of Venda, South Africa  
 Clinton Jeffery, University of Idaho, USA Jason  
 Jung, Yeungnam University, South Korea  
 Taghi Khoshgoftaar, Florida Atlantic University, USA  
 Claudiu V. Kifor, Lucian Blaga University of Sibiu, Romania  
 Jun Kong, North Dakota State University, USA Aneesh  
 Krishna, Curtin University of Technology, Australia Vinay  
 Kulkarni, Tata Consultancy Services, India  
 Jeff Lei, University of Texas at Arlington, USA  
 Meira Levy, Shenkar College of Engineering and Design, Israel  
 Bixin Li, Southeast University, China  
 Ming Li, Nanjing University, China  
 Zhi Li, Guangxi Normal University, China  
 Shih-hsi Liu, California State University, Fresno, USA  
 Xiaodong Liu, Edinburgh Napier University, United Kingdom  
 Yi Liu, GCSU, USA  
 Luanna Lopes Lobato, Federal University of Goi, Brazil  
 Hakim Lounis, UQAM, Canada  
 Baojun Ma, Beijing University of Posts and Telecommunications, China  
 Marcelo de Almeida Maia, Federal University of Uberlândia, Brazil Vijay  
 Mann, IBM Research, India  
 Michele, Marchesi, University of Cagliari, Italy  
 Riccardo Martoglia, University of Modena and Reggio Emilia, Italy  
 Santiago Matalonga, Universidad ORT Uruguay, Uruguay Hong  
 Mei, Peking University, China  
 Hsing Mei, Fu Jen Catholic University, Taiwan  
 Andre Menolli, Universidade Estadual do Norte do Parana (UENP), Brazil  
 Ali Mili, NJIT, USA  
 Alok Mishra, Atilim University, Turkey  
 Manuel Mora, Autonomous University of Aguascalientes, Mexico  
 Kia Ng, ICSRiM - University of Leeds, United Kingdom  
 Allen Nikora, Jet Propulsion Laboratory, USA Amjad  
 Nusayr, University of Houston-Victoria, USA  
 Edson A. Oliveira Junior, State University of Maringa, Brazil  
 Xin Peng, Fudan University, China Oscar  
 Pereira, University of Aveiro, Portugal Antonio  
 Piccinno, University of Bari, Italy Alfonso  
 Pierantonio, University of L'Aquila, Italy Daniel  
 Plante, Stetson University, USA  
 Huseyin Polat, Anadolu University, Turkey  
 Rick Rabiser, Johannes Kepler University, Austria  
 Filip Radulovic, Universidad Politécnic de Madrid, Spain Damith  
 C. Rajapakse, National University of Singapore, Singapore Rajeev  
 Raje, IUPUI, USA  
 Jose Angel Ramos, Universidad Politecnica de Madrid, Spain  
 Kattur Soundarapandian Ravichandran, SASTRA University, India  
 Henrique Rebelo, Universidade Federal de Pernambuco, Brazil  
 Marek Reformat, University of Alberta, Canada  
 Robert Reynolds, Wayne State University, USA  
 Daniel Rodriguez, Universidad de Alcala, Spain

Ivan Rodero, The State University of New Jersey, USA  
 Samira Sadaoui, University of Regina, Canada  
 Masoud Sadjadi, Florida International University, USA  
 Claudio Sant'Anna, Universidade Federal da Bahia, Brazil  
 Andreas Schoenberger, Siemens AG, Germany Ashish  
 Sharma, GLA University, Mathura, India Michael Shin,  
 Texas Tech University, USA  
 Qinbao Song, Xi'an Jiaotong University, China George  
 Spanoudakis, City University, United Kingdom Jing  
 Sun, University of Auckland, New Zealand Yanchun  
 Sun, Peking University, China  
 Gerson Sunye, Institut de Recherche en Informatique et Systemes Aleatoires, France  
 Jeff Tian, Southern Methodist University, USA  
 Genny Tortora, University of Salerno, Italy  
 Mark Trakhtenbrot, Holon Institute of Technology, Israel  
 Peter Troeger, Universitat zu Potsdam, Germany T.  
 H. Tse, The University of Hong Kong, Hong Kong  
 Burak Turham, Oulu University, Finland  
 Giorgio Valle, University of Milan, Italy  
 Sylvain Vauttier, Ecole des mines d'Ales, France Silvia  
 Vergilio, Federal University of Parana (UFPR), Brazil  
 Sergiy Vilkomir, East Carolina University, USA  
 Aaron Visaggio, University of Sannio, Italy  
 Arndt Von Staa, PUC-Rio, Brazil  
 Gurisimran Walia, North Dakota State University, USA  
 Huanjing Wang, Western Kentucky University, USA  
 Jiacun Wang, Monmouth University, USA  
 Linzhang Wang, Nanjing University, China Xiaoyin  
 Wang, University of Texas at San Antonio, USA Ye  
 Wang, Zhejiang Gongshang University, China Hironori  
 Washizaki, Waseda University, Japan  
 Victor Winter, University of Nebraska at Omaha, USA  
 Guido Wirtz, Bamberg University, Germany  
 Eric Wong, University of Texas, USA  
 Franz Wotawa, TU Graz, Austria  
 Dianxiang Xu, Boise State University, USA  
 Frank Xu, Gannon University, USA  
 Haiping Xu, University of Massachusetts Dartmouth, USA  
 Chi-lu Yang, Taiwan Semiconductor Manufacturing Company, Taiwan  
 Hongji Yang, Bath Spa University, United Kingdom  
 Huiqun Yu, East China University of Science and Technology, China  
 Jiang Yue, Fujian Normal University, China  
 Du Zhang, California State University, USA  
 Hongyu Zhang, Tsinghua University, China  
 Yong Zhang, Tsinghua University, China  
 Zhenyu Zhang, University of Hong Kong, Hong Kong  
 Lei Zhao, Wuhan University, China  
 Jiang Zheng, ABB US Corporate Research Center, USA Hong  
 Zhu, Oxford Brookes University, United Kingdom Jianlin Zhu,  
 South-Central University for Nationalities, China Xingquan  
 Zhu, Florida Atlantic University, USA  
 Eugenio Zimeo, University of Sannio, Italy

## **DEMO&POSTER SESSIONS CO-CHAIRS**

Farshad Samimi, USA  
Dragutin Petkovic, San Francisco State University, USA

## **PUBLICITY CHAIR**

Xiaoying Bai, Tsinghua University, China

## **ASIA LIAISON**

Hironori Washizaki, Waseda University, Japan

## **EUROPE LIAISON**

Raul Garcia Castro, Universidad Politecnica de Madrid, Spain

## **INDIA LIAISON**

Swapan Bhattacharya, NITK, Surathakl, India

## **SOUTH AMERICA LIAISON**

Jose Carlos Maldonado, ICMC-USP, Brazil

# Keynote

## Toward the **G System**: The Dynamic Context-Aware *Internet of Things* for Advanced Society

**Son Vuong** Computer Science  
Department University of British  
Columbia Vancouver, BC V6T  
1Z4 Canada Email:  
[vuong@cs.ubc.ca](mailto:vuong@cs.ubc.ca)

*The Internet of Things*, the ultimate in ubiquitous networking, will greatly impact everyone's life in every facet, including entertainment, socialization, business, healthcare and education. Yet, the real structure of the future networks and the adaptable ubiquitous computing model are still in clouds. To approach this ultimate goal of ubiquitous networking, we undertake to develop the so-called **G System**, that combines aspects of context-aware ubiquitous networking, ambient intelligence and smart environments based on dynamic wireless sensor networks (WSNs). To alleviate the explosion problem of collecting, processing and dissemination of huge amount of data and contents in the modern communication and interactions among humans, mobile devices and *things*, we must take into consideration of underlying dynamic context information and the small world model.

The **G System** is defined to be the ultimate application of the Internet of Things to enhance the quality of life and to harmonize the society. In this talk, we will discuss our ongoing development of the overall G System and some various key technologies, subsystems and components that are essential for the G System, including **WISEMAN**, a mobile agent system for intelligent management of wireless sensor networks (WSN); **RFID-G2**, the next-generation RFID for object identification to convey intelligence in the code contained in the RFID-G2 tag; **MOPAR**, a P2P Interest Management System for data collection of relevant local information, **LePlaza**, a location-based social networking system; and **LIVES**, a voice-based system for mobile learning and social networking. Some demo of these subsystems with a particular focus on LIVES system for mobile learning will also be presented, if time permits.

We will discuss the basic architecture and infrastructure of the G System as well as the underlying basic principle of harmonization. The research on the G System is still at the infancy stage, further work and collaborative efforts will be needed to realize the true potential of applying advanced technology to improve society and the wellbeing of mankind.

## Biographical Sketch

**Dr. Son T. Vuong** Received his B.S. in Electrical Engineering from California State University, Sacramento, USA; M.Eng. in Systems Engineering at Carleton University in Ottawa, Canada; and Ph.D. in Computer Science from the University of Waterloo, Canada where he was a lecturer and assistant professor in 1981-1982. Since 1983, he has been a professor of Computer Science at the **University of British Columbia** in Vancouver, Canada, where he was a founder of the Distributed System Research Group and has been the Director of the Laboratory for Networks and Internet Computing (NICLab). He is an international renowned researcher on protocol engineering, distributed multimedia systems, and collaborative computing. His areas of research interest in recent years include the Internet of Things, ambient networks, grid and p2p computing, P2P video streaming, network security, mobile computing and mobile learning, and especially the G (Green) System.

Dr. Vuong has (co) authored a US patent, 200 papers and co-edited three books, including the book on "Recent Advances in Distributed Multimedia Systems" published in 1999. He has supervised thesis research of 80 graduate (PhD and MSc) students. He was a co-leader of the major \$30M grant proposal to establish a new Network of Centres of Excellence in 1999, called *Global Information Systems and Software Technology – GISST* that made to the final short list. Dr. Vuong served on many conference program committees and has been general or technical (co)chair and (co)organizer of thirteen international conferences (AMT'14, iThings'13, ICASA'13, NCAS'11, ACM Multimedia'08, DMS'08, IEEE NOMS'06, DMS'97, ICDCS'95, PSTV'94, FORTE'89, IWPTS'88). He served on the Canadian National Science and Engineering Research Council (NSERC) Grant Selection Committee in 1999-2003, and also serves on the Board of Directors of two high-tech companies.

Since 2013 he also serves as the Co-chair of the Green World System to realize the visions of a clean green world and an advanced society as set out by the respectable founding chair, Mr. Thai Quang Trung.

## Table of Contents

<b>Foreword</b> .....	iii
<b>Conference Organization</b> .....	iv
<b>KeyNote: Toward the G System: The Dynamic Context-Aware Internet of Things for Advanced Society</b> <b>Professor Son T. Vuong</b> .....	ix
<hr/> <b>Metrics Analysis and Utilization</b> <hr/>	
Service Identification Based on Quality Metrics - Object-Oriented Legacy System Migration Towards SOA (S)..... <i>Seza Adjoyan, Abdelhak-Djamel Seriai and Anas Shatnawi</i>	1
Automated Metrics Collection for IEC61131-3 Languages from Engineering Tools (S)..... <i>Mani Shankar and Anil Nair</i>	7
GUIEvaluator: A Metric-tool for Evaluating the Complexity of Graphical User Interfaces (S) .....	13
<i>Khalid Alemerien and Kenneth Magel</i>	
<hr/> <b>Performance Analysis and Evaluation</b> <hr/>	
Improving Static Analysis Performance Using Rule-Filtering Technique (S)..... <i>Deng Chen, Rubing Huang, Binbin Qu and Sheng Jiang</i>	19
Performance and Usability Evaluation of a Pattern-Oriented Parallel Programming Interface for Multi-Core Architectures (S) .....	25
<i>Dalvan Griebler, Daniel Adornes and Luiz Fernandes</i>	
Performance Benchmarking of BPEL Engines: A Comparison Framework, Status Quo Evaluation and Challenges (S) .....	31
<i>Cedric Röck, Simon Harrer and Guido Wirtz</i>	
<hr/> <b>Software Development</b> <hr/>	
A Tool for Trade-off Resolution on Architecture-Centered Software Development (S)..... <i>Italo Silva, Patrick Brito, Balduino Neto, Evandro Costa and Hemilis Rocha</i>	35
GreenRM: Reference Model for Sustainable Software Development (S)..... <i>Marcello Thiry, Liliane Frez and Alessandra Zoucas</i>	39
A Multicriteria Approach to Project Portfolio Selection (S)..... <i>Everton Gomedé and Rodolfo Barros</i>	43
<hr/> <b>Mobile Applications</b> <hr/>	
Design and Development of a Mobile Classroom Response Software for Interactive Problem Solving (S)..... <i>Mohammad Muztaba Fuad and Debzani Deb</i>	49

Towards Automatic Consistency Checking between Web Application and its Mobile Application (S) .....	53
<i>Xiangping Chen and Zhensheng Xu</i>	
Mobile Applications: The Paradox of Software Estimation (S) .....	59
<i>Laudson Silva de Souza and Gibeon Soares de Aquino Jr.</i>	
<hr/> <b>Software Product Line</b> <hr/>	
An Evolutionary Methodology for Optimized Feature Selection in Software Product Lines (S) .....	63
<i>Xiaoli Lian and Li Zhang</i>	
Flexible Modeling and Product Derivation in Software Product Lines (S) .....	67
<i>Jorge Barreiros and Ana Moreira</i>	
A Critical Embedded System Product Line Model-based Approach (S) .....	71
<i>Paulo Queiroz and Rosana Braga</i>	
<hr/> <b>Ontology and Knowledge Discovery</b> <hr/>	
Creating Proprietary Terms Using Lightweight Ontology: A Case Study on Acquisition Phase in a Cyber Forensic Process (S) .....	76
<i>Tamer Gayed, Hakim Lounis and Moncef Bari</i>	
DKDs: An Ontology-based System for Distributed Teams (S) .....	82
<i>Rodrigo Rocha, Ryan Azevedo, Marcos P. Duarte, Dimas Cassimiro, Ana Raquel M. Alves and Silvio Meira</i>	
Industrial Analytics to Discover Knowledge from Instrumented Networked Machines (S) ..	86
<i>Aldo Dagnino and David Cox</i>	
<hr/> <b>Testing: Network Protocols, Models and Development</b> <hr/>	
Testing Network Protocols: formally, at runtime and online (S) .....	90
<i>Xiaoping Che, Stephane Maag, Jorge Lopez and Ana Cavalli</i>	
Testing Model Transformation Programs using Metamorphic Testing (S) .....	94
<i>Mingyue Jiang, T.Y. Chen, Fei-Ching Kuo, Zhiquan Zhou and Zuohua Ding</i>	
Reactive Variability Realization with Test Driven Development and Refactoring (S) .....	100
<i>Glauco Neves and Patrícia Vilain</i>	
<hr/> <b>Testing: Generation of Tests and Unit Tests</b> <hr/>	
A Controlled Experiment to Explore Potentially Undetectable Defects for Testing Techniques (S) .....	106
<i>Martín Solari and Santiago Matalonga</i>	
Test Data Generation for Web Applications: A Constraint and Knowledge-based Approach (S) .....	110
<i>Hibiki Saito, Shingo Takada, Haruto Tanno and Morihide Oinuma</i>	

Towards a Unified Metrics Suite for JUnit Test Cases (S).....	115
<i>Fadel Toure, Mourad Badri and Luc Lamontagne</i>	

---

### **Prioritization of Tests and SVN Analysis**

---

How to Do Tie-breaking in Prioritization of Interaction Test Suites? (S).....	121
<i>Rubing Huang, Jinfu Chen, Rongcun Wang and Deng Chen</i>	
An Application of Adaptive Random Sequence in Test Case Prioritization (S).....	126
<i>Xiaofang Zhang, Tsong Yueh Chen and Huai Liu</i>	
An Empirical Study on Inter-Commit Times in SVN (S).....	132
<i>Qiuju Hou, Yutao Ma, Jianxun Chen and Youwei Xu</i>	

---

### **Object-Oriented Programming**

---

Documenting the Mined Feature Implementations from the Object-oriented Source Code of a Collection of Software Product Variants.....	138
<i>Ra'fat Al-Msie'Deen, Abdelhak Seriai, Marianne Huchard, Christelle Urtado and Sylvain Vauttier</i>	
An empirical study on the adoption of C++ templates: Library templates versus user defined templates.....	144
<i>Di Wu, Lin Chen, Yuming Zhou and Baowen Xu</i>	
TyS A Framework to Facilitate the Implementation of Object-Oriented Type Checkers...	150
<i>Francisco Ortin, Daniel Zapico, Jose Quiroga and Miguel García</i>	

---

### **Architecture: Analysis and Self-Adaptation**

---

Software Architecture Rationale Capture through Intelligent Argumentation.....	156
<i>Xiaoqing Liu, Nagaprashanth Chanda and Eric Christopher Barnes</i>	
An Approach for Capturing and Documenting Architectural Decisions of Reference Architectures.....	162
<i>Milena Guessi, Flavio Oquendo and Elisa Yumi Nakagawa</i>	
Towards a Tactic-Based Evaluation of Self-Adaptive Software Architecture Availability...	168
<i>Alireza Parvizi-Mosaed, Shahrouz Moaven, Jafar Habibi and Abbas Heydarnoori</i>	

---

### **Architecture: Synthesis and Implementation**

---

Automated Software Architectural Synthesis using Patterns: A Cooperative Coevolution Approach.....	174
<i>Yongrui Xu and Peng Liang</i>	
Towards Reusing Architectural Knowledge as Design Guides.....	181
<i>Mohsen Anvaari and Olaf Zimmermann</i>	
Aspect-Oriented Secure Connectors for Implementation of Secure Software Architecture ..	187
<i>Chase Baker and Michael Shin</i>	

---

### **Testing: Randomness, Implementation and Event-Based**

---

Applying Random Testing to Constrained Interaction Testing .....	193
<i>Yasuhiro Hirasaki, Hideharu Kojima and Tatsuhiro Tsuchiya</i>	
An Extensible Framework to Implement Test Oracle for Non-Testable Programs .....	199
<i>Rafael A.P. Oliveira, Atif Memon, Victor N. Gil, Fatima L.S. Nunes and Marcio Delamaro</i>	
Empirical Comparison of Intermediate Representations for Android Applications .....	205
<i>Yauhen Arnatovich, Hee Beng Kuan Tan, Ding Sun, Kaping Liu and Lwin Khin Shar</i>	
<hr/> <b>Software Assurance I</b> <hr/>	
Effectiveness of Automated Function Testing with Petri Nets: A Series of Controlled Experiments .....	211
<i>Dianxiang Xu and Ning Shen</i>	
Automatic XACML requests generation for testing access control policies.....	217
<i>Yongchao Li, You Li, Linzhang Wang and Guanling Chen</i>	
Reducing Test Cases with Causality Partitions .....	223
<i>Haijun Wang, Xiaohong Guan, Qinghua Zheng, Ting Liu, Xiangyang Li, Lechen Yu and Zijiang Yang</i>	
<hr/> <b>KMISE I</b> <hr/>	
How to Enhance the Creativity of Software Developers: A Systematic Literature Review..	229
<i>Reshma Hegde and Gursimran Walia</i>	
Knowledge Transfer between Senior and Novice Software Engineers: A Qualitative Analysis .....	235
<i>Davi Viana, Tayana Conte and Cleidson De Souza</i>	
A Knowledge & Competencies Checklist for Software Project Management Success .....	241
<i>Lawrence Peters</i>	
<hr/> <b>Control Experiments in Use</b> <hr/>	
Identifying strategies on god class detection in two controlled experiments .....	244
<i>José A. M. Santos and Manoel Mendonça</i>	
An Empirical Study to Evaluate a Domain Specific Language for Formalizing Software Engineering Experiments.....	250
<i>Marília Freire, Uirá Kulesza, Eduardo Aranha, Andreas Jedlitschka, Edmilson Campos, Silvia T. Acuña and Marta Gómez</i>	
Identifying Threats to Validity and Control Actions in the Planning Stages of Controlled Experiments .....	256
<i>Amadeu Anderlin Neto and Tayana Conte</i>	
<hr/> <b>Collaborative Software Development</b> <hr/>	
Case-based Reasoning for Experience-based Collaborative Risk Management.....	262
<i>Nielsen Luiz Rechia Machado, Luís Alvaro de Lima Silva, Lisandra Manzoni Fontoura and John A. Campbell</i>	

Collaborative Merge in Distributed Software Development: Who Should Participate? . . . . .	268
<i>Catarina Costa, Jose J. C. Figueiredo and Leonardo Murta</i>	
An Argument-based Collaborative Negotiation Approach to Support Software Design Collaboration . . . . .	274
<i>Nan Jing, Stephen Lu and Hung-Fu Chang</i>	
<hr/> <b>Data Analysis and Quality in Software Development</b> <hr/>	
Comparing Two Approaches for Adding Feature Ranking to Sampled Ensemble Learning for Software Quality Estimation . . . . .	280
<i>Kehan Gao, Taghi Khoshgoftar and Amri Napolitano</i>	
CoMoVi: a Framework for Data Transformation in Credit Behavioral Scoring Applications Using Model Driven Architecture . . . . .	286
<i>Rosalvo Oliveira Neto, Paulo Adeodato, Ana Carolina Salgado, Dailton Rodrigues de Carvalho Filho and Genival Rocha Machado</i>	
Tracing Domain Data Concepts in Layered Applications . . . . .	292
<i>Mohammed Daubal, Nathan Duncan, Delmar Davis and Hazeline Asuncion</i>	
<hr/> <b>Agile Development: Quality, Effort and Documenting</b> <hr/>	
The Agile Quality Culture - A survey on agile culture and software quality (S) . . . . .	298
<i>Bruno Oliveira and Simone Souza</i>	
Story Point Approach based Agile Software Effort Estimation using Various SVR Kernel Methods (S) . . . . .	304
<i>Shashank Mouli Satapathy, Aditi Panda and Santanu Kumar Rath</i>	
Identifying and Recording Software Architectural Assumptions in Agile Development . . . . .	308
<i>Chen Yang and Peng Liang</i>	
<hr/> <b>UML: Transformation, Semantics and Model Generation</b> <hr/>	
Extended DEVSML as a Model Transformation Intermediary to Make UML Diagrams Executable (S) . . . . .	314
<i>Jianpeng Hu, Linpeng Huang, Bei Cao and Xuling Chang</i>	
Detecting Semantic Equivalence in UML Class Diagrams (S) . . . . .	318
<i>Valeria Costa, Rodrigo Monteiro and Leonardo Murta</i>	
Behavioral Model Generation from Use Cases Based on Ontology Mapping and GRASP Patterns . . . . .	324
<i>Nurfauza Jali, Des Greer and Philip Hanna</i>	
<hr/> <b>Agents: Componentets and Application</b> <hr/>	
Semantic-based Repository of Agent Components (S) . . . . .	330
<i>Merlin Parra Jiménez, Andrew Diniz Da Costa and Carlos José Pereira de Lucena</i>	
A Multi-Agent-Based Approach for Autonomic Data Exchange Processes (S) . . . . .	334
<i>Hicham Assoudi and Hakim Lounis</i>	

Analysis, Design and Implementation of an Agent Based System for Simulating Connected Vehicles .....	338
<i>Elahe Paikari and Behrouz Far</i>	

---

## **KMISE II**

---

Towards a Taxonomy of Services for Developing Service-Oriented Robotic Systems (S)....	344
<i>Lucas Bueno Ruas Oliveira, Fernando Santos Osório, Flavio Oquendo and Elisa Yumi Nakagawa</i>	

Knowledge from Document Annotations as By-Product in Distributed Software Engineering (S) .....	350
<i>Anna Averbakh, Kai Niklas and Kurt Schneider</i>	

Dedicated Support for Experience Sharing in Distributed Software Projects.....	355
<i>Anna Averbakh, Eric Knauss, Stephan Kiesling and Kurt Schneider</i>	

---

## **Web: Service Composition, Mining and Recommendation**

---

QoS-Based Web Service Composition by GA Using Consumer Decision-Making Function (S) .....	361
<i>Gang Wang</i>	

Using Web Mining to Support Low Cost Historical Vehicle Traffic Analytics (S) .....	365
<i>Charanjeet Kaur, Diwakar Krishnamurthy and Behrouz Far</i>	

Cold-Start Web Service Recommendation Using Implicit Feedback.....	371
<i>Gang Tian, Jian Wang, Keqing He, Weidong Zhao and Pan-Pan Gao</i>	

---

## **Business Process: Models, Rules and Management**

---

An Improved Structure-based Approach to Measure Similarity of Business Process Models (S) .....	377
<i>Jimin Ling, Li Zhang and Qi Feng</i>	

A Method for Verifying the Consistency of Business Rules Using Alloy (S).....	381
<i>Denilson Guimaraes, Eber Schmitz, Antonio Juarez Alencar, Priscila Lima and Alexandre Correa</i>	

Making the link between strategy and process model collections: a multi-layered approach	387
<i>Felipe Dallilo, Joao Porto De Albuquerque and Marcelo Fantinato</i>	

---

## **Requirements: Verification and Utilization**

---

Formal Verification of Coordination Systems' Requirements - A Case Study on the European Train Control System (S) .....	393
<i>Huu Nghia Nguyen and Ana Cavalli</i>	

Evaluating the Use of Model-Based Requirement Verification Method: An Empirical Study (S) .....	397
<i>Munmun Gupta, Daniel Aceituna, Gursimran Walia and Do Hyunsook</i>	

On the Requirements and Design Decisions of an In-House Component-Based SPL Automated Environment .....	402
<i>Elder Macedo Rodrigues, Leonardo Passos, Leopoldo Teixeira, Avelino F. Zorzo, Flavio Oliveira and Rodrigo Saad</i>	
<hr/> <b>Software Development Aspects</b> <hr/>	
A COSMIC Measurement Procedure for BPMN Diagrams (S) .....	408
<i>Beatriz Mariñ and José Quinteros</i>	
Proposing a Software Process Model for Follow the Sun Development (S) .....	412
<i>Josiane Kroll, Ita Richardson and Jorge L. N. Audy</i>	
Efficient Points-To Analysis for Partial Call Graph Construction .....	416
<i>Zhiyuan Wan, Bo Zhou, Ye Wang and Yuanhong Shen</i>	
<hr/> <b>Clustering and Feature Analysis</b> <hr/>	
Text-Based Clustering and Analysis of Intelligent Argumentation Data (S) .....	422
<i>Eric Barnes and Xiaoqing Liu</i>	
Feature Location in a Collection of Product Variants: Combining Information Retrieval and Hierarchical Clustering (S) .....	426
<i>Hamzeh Eyal-Salman, Abdelhak Seriai and Christophe Dony</i>	
Feature model recovery from product variants based on a cloning technique .....	431
<i>Jihen Maâzoun, Nadia Bouassida and Hanêne Ben-Abdallah</i>	
<hr/> <b>Release Evaluation and Change Analysis</b> <hr/>	
RELREA - An Analytical Approach for Evaluating Release Readiness (S) .....	437
<i>S. M. Shahnewaz and Guenther Ruhe</i>	
Change and Role as First-Class Abstractions for Realising Dynamic Evolution (S) .....	443
<i>Yin Chen and Xinjun Mao</i>	
Feature-Level Change Impact Analysis Using Formal Concept Analysis .....	447
<i>Hamzeh Eyal-Salman, Abdelhak Seriai and Christophe Dony</i>	
<hr/> <b>Ontology and Database Access Control</b> <hr/>	
Are The Integrations Between Ontologies and Databases Really Opening the Closed World in Ubiquitous Computing? (S) .....	453
<i>Vinícius Maran, Iara Augustin and José Palazzo M. de Oliveira</i>	
ONTO-ResAsset Development: An Ontology for Reusable Assets Specification and Management (S) .....	459
<i>Luciano Da Silva, Débora Paiva, Ellen Barbosa, Rosana Braga and Maria Istela Cagnin</i>	
Extending RBAC Model to Control Sequences of CRUD Expressions .....	463
<i>Oscar Pereira, Diogo Regateiro and Rui Aguiar</i>	
<hr/> <b>Software Project, Informaiton and Skills</b> <hr/>	

Assisting Software Projects with Assignment Recommen-der Creation (S) .....	470
<i>John Anvik, Marshall Brooks, Henry Burton and Justin Canada</i>	
Recovering Valuable Information Behaviour from OSS Contributors: An Exploratory Study (S) .....	474
<i>Mário André De Freitas Farias, Paulo Ortins, Ranato Novais, Methanias Colaço and Manoel Mendonça</i>	
Measurement of the Non-Technical Skills of Software Professionals: An Empirical Investigation .....	478
<i>Lisa Bender, Walía Gursimran, Fabian Fagerholm, Max Pagels and Kendall Nygard</i>	
<hr/> <b>Repositories: Utilization and Analysis</b> <hr/>	
Understanding the popularity of reporters and assignees in the Github (S) .....	484
<i>Joicy Xavier, Autran Macedo and Marcelo Maia</i>	
APT: Approximate Period Detection in Time Series (S) .....	490
<i>Rasaq Otunba and Jessica Lin</i>	
Exploratory Data Analysis of Software Repositories via GPU Processing .....	495
<i>Jose Ricardo Da Silva Júnior, Esteban Clua, Leonardo Murta and Anita Sarma</i>	
<hr/> <b>Applications</b> <hr/>	
Artificial neural networks for infectious diarrhea prediction using meteorological factors in Shanghai (S) .....	501
<i>Yongming Wang, Junzhong Gu and Zili Zhou</i>	
Topic Evolutions in Scientific Conferences (S) .....	507
<i>Lin Zhang, Yao Guo, Xiangqun Chen, Weizhong Shao and Lei Chen</i>	
Applications of Slow Intelligence Frameworks for Energy-Saving Control .....	511
<i>Wen-Hui Chen, Shi-Kuo Chang and Wen-Ping Hung</i>	
<hr/> <b>Management and Software Engineering Education</b> <hr/>	
PSP support component integrated into a web project management environment (S) .....	516
<i>Antonio Marcos Neves Esteca, Rogéria C. G. Souza, Adriana Barbosa Santos, Carlos Roberto Valêncio and Vanessa Dos Anjos Borges</i>	
A Semantic Analyzer for Simple Games Source Codes to Programming Learning (S) .....	522
<i>Elanne Cristina Oliveira Dos Santos, Gleison Brito Batista, Victor Hugo Vieira de Sousa and Esteban Clua</i>	
Supporting Online Synchronous Education for Software Engineering via Web-based Operation Record and Replay .....	528
<i>Dejian Chen and Yanchun Sun</i>	
<hr/> <b>Machine Learning and Prediction</b> <hr/>	
A DIMENSIONALITY REDUCTION PROCESS TO FORECAST EVENTS THROUGH STOCHASTIC MODELS .....	534
<i>Paulo Fernandes, Joaquim Assunção, Lucelene Lopes and Silvio Gomez</i>	

Choosing the Best Classification Performance Metric for Wrapper-based Software Metric Selection for Defect Prediction .....	540
<i>Huanjing Wang, Taghi Khoshgoftaar and Amri Napolitano</i>	
Synthetic Minority Over-sampling TEchnique (SMOTE) for Predicting Software Build Outcomes (S) .....	546
<i>Jacqui Finlay, Russel Pears and Andy Connor</i>	
<hr/> <b>Software Knowledge and Inf. Representation</b> <hr/>	
Building Empirical Software Engineering Bodies of Knowledge with Systematic Knowledge Engineering .....	552
<i>Stefan Biffl, Marcos Kalinowski, Fajar J. Ekaputra, Estefania Serral and Dietmar Winkler</i>	
A Body of Knowledge for Executing Performance Analysis of Software Processes .....	560
<i>Natália Chaves Lessa Schots, Ana Regina Rocha and Gleison Santos</i>	
Towards a flexible approach to manage varying and altering information representations (S) .....	566
<i>Tobias Haubold, Georg Beier and Wolfram Hardt</i>	
<hr/> <b>Risk Management</b> <hr/>	
Flood Citizen Observatory: a crowdsourcing-based approach for flood risk management in Brazil .....	570
<i>Lívia Castro Degrossi, João Porto de Albuquerque, Maria Clara Fava and Eduardo Mario Menciondo</i>	
Lightweight Risk Management in Agile Projects .....	576
<i>Edzreena Edza Odzaly, Des Greer and Darryl Stewart</i>	
Snowball Effects on Risk Mitigation Scheduling: Process and Tool (S) .....	582
<i>Hareton K. N. Leung, Kim Man Lui and Peng Zhou</i>	
<hr/> <b>Software Runtime Aspects</b> <hr/>	
Reasoning at Runtime using time-distorted Contexts: A Models@run.time based Approach .....	586
<i>Thomas Hartmann, Francois Fouquet, Grégory Nain, Brice Morin, Jacques Klein and Yves Le Traon</i>	
Generating Real-Time Profiles of Runtime Energy Consumption for Java Applications ....	592
<i>Muhammad Nassar, Julian Jarrett, Iman Saleh and M. Brian Blake</i>	
Towards Sustainability-Oriented Development of Dynamic Reconfigurable Software Systems (S) .....	598
<i>Shan Tang, Liping Li, Wenjing Yang and Jianxin Xue</i>	
<hr/> <b>Security: from Requirements to Data</b> <hr/>	
Persona Security: A Technique for Supporting the Elicitation of Security Requirements ...	603
<i>Marco Aurélio Dos Santos, Jacilane Rabelo, Raimundo Barreto and Tayana Conte</i>	

Runtime Code Reuse Attacks: A Dynamic Framework Bypassing Fine-Grained Address Space Layout Randomization.....	609
<i>Yi Zhuang, Tao Zheng and Zhitian Lin</i>	
Industry-wise Analysis of Security Breaches in Data Loss Incidents (S) .....	615
<i>Rehab El-Kharboutly, Swapna Gokhale and Lance Fiondella</i>	
<hr/> <b>Software Assurance II</b> <hr/>	
Bug Inducing Analysis to Prevent Fault Prone Bug Fixes.....	620
<i>Haoyu Yang, Chen Wang, Qingkai Shi, Yang Feng and Zhenyu Chen</i>	
Development of A Sliding Window Protocol for Data Synchronization in a Flow Cytometer	626
<i>Junhua Ding, Yuxiang Shao and Dongmei Zhang</i>	
An Empirical Study on the Test Adequacy Criterion Based on Coincidental Correctness Probability (S).....	632
<i>Xiaoli Zhou, Linzhang Wang, Xuandong Li and Jianhua Zhao</i>	
<hr/> <b>Social Networks and Ontology-based Search</b> <hr/>	
Forwarding Links without Browsing Links in Online Social Networks .....	636
<i>Jing Jiang, Xiao Wang, Li Zhang and Yafei Dai</i>	
Accurate Local Estimation of Geo-Coordinates for Social Media Posts .....	642
<i>Derek Doran, Swapna Gokhale and Aldo Dagnino</i>	
Two-Level Smart Search Engine Using Ontology-Based Semantic Reasoning (S) .....	648
<i>Haiping Xu and Arturo Li</i>	
<hr/> <b>Social Networks, Representation and Influence Measurement</b> <hr/>	
User Profile Visualization to facilitate MSLIM-model-based Social Influence Analysis based on Slow Intelligence Approach .....	653
<i>Yingze Wang and Shi-Kuo Chang</i>	
Method for Measuring Twitter Content Influence .....	659
<i>Euijong Lee, Jeong-Dong Kim and Doo-Kwon Baik</i>	
An Exploratory Search for Presentation Contents based on Slide Semantic Structure (S) ..	665
<i>Yuanyuan Wang, Yukiko Kawai and Kazutoshi Sumiya</i>	
<hr/> <b>Product Lines and Business Process</b> <hr/>	
From Intentions to Decisions: Understanding Stakeholders Objectives in Software Product Line Configuration .....	671
<i>Mahdi Noorian, Ebrahim Bagheri and Weichang Du</i>	
Towards the Establishment of a Software Product Line for Mobile Learning Applications .	678
<i>Venilton Falvo Junior, Nemésio Freitas Duarte Filho, Edson Oliveira Junior and Ellen Francine Barbosa</i>	

Automated transformation of business rules specification to business process model (S) . . .	684
<i>Olfa Chourabi Tan Tan and Jacky Akoka</i>	

---

### **Computational Intelligence and Machine Learning**

---

Practical Human Resource Allocation in Software Projects Using Genetic Algorithm . . . . .	688
<i>Jihun Park, Dongwon Seo, Gwangui Hong, Donghwan Shin, Jimin Hwa and Doo-Hwan Bae</i>	
Mental models analysis based on fuzzy rules for collaborative decision-making . . . . .	695
<i>Pedro I. Garcia-Nunes, Ana E. Silva, Antonio C. Zambon and Gisele Baioco</i>	
Software Requirement Prioritization using Machine Learning (S) . . . . .	701
<i>Deepali Singh and Ashish Sharma</i>	

---

### **Database Usage and Refactoring**

---

Detecting Anomaly in the Usage of Database Attribute . . . . .	705
<i>Kaiping Liu, Hee Beng Kuan Tan and Yauhen Arnatovich</i>	
RefactoringScript: A Script and Its Processor for Composite Refactoring . . . . .	711
<i>Linchao Yang, Tomoyuki Kamiya, Kazunori Sakamoto, Hironori Washizaki and Yoshiaki Fukazawa</i>	
Investigation for Software Power Consumption of Code Refactoring Techniques (S) . . . . .	717
<i>Jae-Jin Park, Jang-Eui Hong and Sang-Ho Lee</i>	

---

### **Testing and Economical Aspects**

---

Testing as an Investment . . . . .	723
<i>Xiaoran Xu, Chunrong Fang, Qing Wu, Jia Liu and Zhenyu Chen</i>	
A Proposal for the Improvement of Projects Cost Predictability using Earned Value Management and Historical Data of Cost An Empirical Study . . . . .	729
<i>Adler Diniz De Souza, Ana Regina Rocha and Djenane Cristina Silveira dos Santos</i>	
Improving the Cost Effectiveness of Software Inspection Teams: An Empirical Investigation (S) . . . . .	735
<i>Anurag Goswami and Gursimran Walia</i>	

---

### **Poster and Demo**

---

DBPD: A Dynamic Birthmark-based Software Plagiarism Detection Tool (P) . . . . .	740
<i>Zhenzhou Tian, Qinghua Zheng, Ming Fan, Eryue Zhuang, Haijun Wang and Ting Liu</i>	
A MAPE Loop Control Pattern for Heterogeneous Client/Server Online Games (P) . . . . .	742
<i>Satoru Yamagata, Hiroyuki Nakagawa, Yuichi Sei, Yasuyuki Tahara and Akihiko Ohsuga</i>	
DiCoMEF: A Distributed Collaborative Model Editing Framework (P) . . . . .	744
<i>Amanuel Koshima and Vincent Englebert</i>	
Model-based time-distorted Contexts for efficient temporal Reasoning (P) . . . . .	746
<i>Thomas Hartmann, Francois Fouquet, Grégory Nain, Brice Morin, Jacques Klein and Yves Le Traon</i>	

Agent-based Stochastic Simulation of Schema Matching (P) .....	748
<i>Hicham Assoudi and Hakim Lounis</i>	
<b>Author's Index</b> .....	A-1
<b>Program Committee Reviewers' Index</b> .....	A-11
<b>External Reviewers' Index</b> .....	A-15

**Note:**

**(S)** indicates a short paper.

**(P)** indicates a poster or demo, which is not a refereed paper.

# Service Identification Based on Quality Metrics

## Object-Oriented Legacy System Migration Towards SOA

Seza Adjoyan<sup>\*</sup>, Abdelhak- Djamel Seriai<sup>\*</sup>, Anas Shatnawi<sup>\*</sup>

<sup>\*</sup>LIRMM, CNRS and University of Montpellier 2

161 rue Ada, Montpellier, France

{adjoyan, seriai, shatnawi}@lirmm.fr

**Abstract**—Migrating towards Service Oriented Architecture SOA has become a major topic of interest during the recent years. Since emerging service technologies have forced non-service based software systems to become legacy, many efforts and researches have been carried out to enable these legacy systems to survive. In this context, several service identification solutions have been proposed. These approaches are either manual, thus considered expensive, or rely on ad-hoc criteria that fail to identify relevant services. In this paper, within the big picture of migrating object-oriented legacy systems towards SOA, we address automatic service identification from source code based on service quality characteristics. To achieve this, we propose a quality measurement model where characteristics of services are refined to measurable metrics.

**Keywords**- *SOA; reengineering; migration; reverse engineering; Object-Oriented; service identification; quality; software reuse; legacy system.*

### I. INTRODUCTION

Service Oriented Architecture SOA, whose main bricks are services [7], has become a trend [3, 5] of computing paradigm to describe business functionalities and application logics. In SOA, a system is structured into a set of loosely coupled [6, 13, 20] and interoperable business services that can be easily composed [7], reused [7] and shared [8] regardless of their physical location. Services could either be all implemented on a single machine, residing on several machines of company's internal network, or even distributed on several systems over internet [2]. Moreover, having solid service oriented architecture in place will provide the infrastructure needed to successfully deploy services in cloud environment.

The evolution of service technologies in recent years has led non-service based software systems to become legacy software [3, 5]. Any software which has been developed using outdated technology [1], but still brings great value to the organization that uses it, is considered as a legacy software [18, 1]. In order to follow new technological advances and yet to conserve existing business value of current systems, a migration, which is considered as a variation of wrapping methodology [18], of legacy system should be carried out. Several approaches for legacy system migration towards SOA have been reported in literature [1, 2, 3, 4, 7, 9, 10, 14, 18, 19, 21]. SOA migration is achieved through two major phases: (1) legacy analysis, where available software artifacts are analyzed to identify provided services and (2) service implementation, that leverages extracted legacy code as usable services, wraps them by interfaces and orchestrates their operations. The first

phase (i.e. service identification) is crucial in this process, especially with the unavailability of certain resources (e.g. developers, architects) and poor documentation [4, 7]. Even more, it is a challenging task, since legacy systems are not necessarily built with the vision of service. Therefore many approaches have been proposed to identify services by analyzing legacy software artifacts. The majority of them are carried out manually [1, 7, 9]. These solutions are considered as expensive in terms of expertise. Thus, some automatic or quasi-automatic approaches were proposed [3, 5, 6, 19, 21]. Most of these approaches assume the existence of large range of information about legacy systems such as their documentation, architecture and design documents [7, 21]. Therefore they are specific to systems where such information is available. They cannot be applied to a large number of systems where only the source code is available [13]. In addition, these approaches rely on ad-hoc criteria for evaluating candidate services, hence a gap between identified services and expected ones.

Our contribution in this paper is to automatically identify services from object-oriented source code. Unlike existing approaches, our service identification process is based on a quality function that measures the semantic correctness of identified services. We introduce a semantic correctness model in order to refine well-known service characteristics to measurable metrics.

The rest of the paper is structured as follows: In section 2, we outline the related works for service identification within migration towards SOA approaches. In section 3, we present our approach of service identification from object-oriented source code by defining quality metrics to evaluate services. In section 4, we evaluate our approach on two case studies. Finally, section 5 concludes the paper and provides some future directions.

### II. RELATED WORK

Most of the approaches proposing migration of legacy systems to service-based ones offer only guidelines to identify services [4, 9, 10, 14]. Few of them propose technical steps. In [7], authors present a migration approach called Service-Oriented Migration and Reuse Technique (SMART). It defines five steps to achieve the migration of legacy system towards SOA. However, the proposed approach requires several sources of information (e.g. documentation) to support the analysis of the legacy system. Besides, the approach largely relies on human interaction (e.g. system analyst, maintenance programmer, etc.) that gathers information through

interviewing stakeholders in order to fill the gap between existing legacy system and target architecture. In [5], an architecture-based and requirement-driven service-oriented reengineering method is discussed. Services are identified by domain analysis and business function identification. The approach is based on both requirements abstraction and source code levels. This approach needs architectural and requirement information to be available. [1] proposes an automatic approach to evaluate candidate services. Candidate services are considered as groups of object-oriented classes evaluated in terms of development, maintenance and estimated replacement costs. Other approaches propose to evaluate services either by code pattern matching and graph transformation [19], feature location [3] or formal concept analysis [6]. A detailed survey of all service identification methods is discussed in [11].

Services and software components have several characteristics in common, in particular, those related to their quality, nature, structure and behavior. For that obvious reason, component identification techniques from object oriented legacy system could be considered as related to this paper. One of the previous works in our team identifies components from object-oriented source code based on quality-centric metrics [22].

As to SOA quality metrics, diverse studies have been proposed in literature for measuring qualitative properties of SOA systems. Most of these works either assess systems that are already service based or evaluate systems only after their implementation. Unfortunately, such researches are not adapted to the context of reengineering an object-oriented system towards service oriented system. [23] proposes a framework to measure the degree of service orientation in SOA systems. It focuses on the internal SOA attribute, decomposes selected attribute to a set of factors and maps each factor to a set of measurable criteria. Each criterion is typically evaluated by a set of software metrics, though no dedicated metrics are defined for each criterion in the paper.

### III. PROPOSED APPROACH

We propose a migration technique that identifies services as groups of classes in the legacy software source code. We base our legacy system analysis on the source code, since it is the only resource that is always available, while other resources such as documentation or architecture could often be missing. Unlike other approaches that identify candidate services in source code manually, we propose an automatic identification method of candidate services. Our approach is based on the definition of a fitness function that measures semantic correctness of each group of source code elements to be considered as a service.

#### A. Object-to-service mapping model

In order to be capable to identify services from object-oriented source code, we define a mapping between object oriented and SOA concepts (see Figure 1). We consider a service as a group of classes defined in object-oriented source code. Among these classes, some define the operations provided by the service, whereas others are inner classes. Inner classes are those which only have internal connections to other classes of the same service. Classes that define the operations

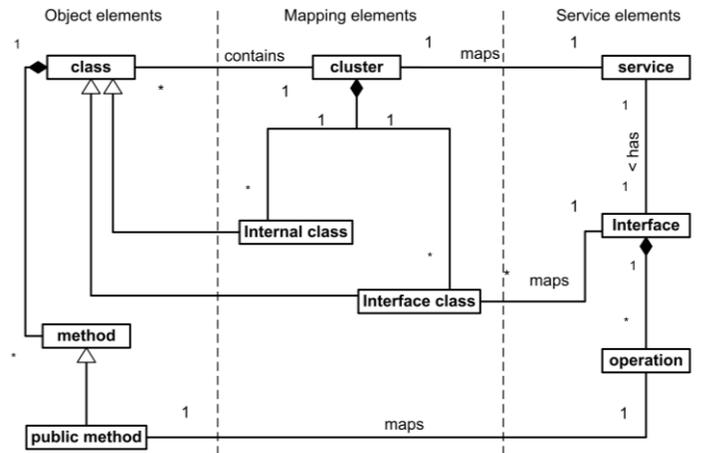


Figure 1. Object-service mapping model.

provided by the service are the classes that define its interface. Inner classes do not define operations provided by the service. Operations provided by the service are class's public methods.

#### B. Quality Measurement Model of Services

As we have mentioned earlier, a service is identified from a group of object-oriented classes. Initially, each group of classes is considered as a candidate service. A qualified service is selected from candidate ones based on a function that measures its quality. Similar to the standard for the evaluation of software quality ISO/IEC 25010:2011 [12], we define this quality function of services based on a set of characteristics that are mapped to a set of properties. Each property is later measured using a set of metrics.

##### 1) Characteristics of Services

We deduce the quality characteristics of services based on the analysis of the most commonly used definitions of services in literature.

In literature, there are several definitions of services [2, 5, 13]. According to [5], a service is an abstract resource that performs a coherent and functional task. [13] considers a service as a process that has an open interface, self-containedness and coarse granularity. It can be easily composed and decomposed to implement various business workflows. [2] defines the service in terms of its characteristics: A service is a coarse-grained and discoverable software entity that interacts with applications and other services through a loosely coupled, often asynchronous, message-based communication model. Coarse-grained means that services implement more than one functionality and operate on larger data sets. Discoverable means that services can be found at both design time and run time, not only by unique identity but also by interface identity and by service kind. Self-contained refers to the self-sufficiency a service has, where context or state information is not required from other services. For loosely coupled, services are connected to other services and clients using standard, dependency-reducing, decoupled message-based methods such as XML document exchanges.

TABLE I. CHARACTERISTICS OF SERVICES

Characteristic	Type	
	Structural and Behavioral	SOA platform
coarse-grained = functionality	✓	
discoverable		✓
self-contained = loosely-coupled	✓	
dynamic-binding		✓
composable	✓	
message-based		✓
asynchronous		✓

Table I lists the characteristics of services as mentioned in the definitions above. We have categorized them into two categories: those related to the structure and behavior of services and others related to the SOA platform. In order to measure the semantic correctness of candidate services, we select from the aforementioned characteristics the ones that define service structure and behavior: self-containment, composability and coarse-grained (functionality).

### 2) Refinement of Service Characteristics

The former selected characteristics are refined to measurable quality properties.

- A service can be completely self-contained if it does not require any interface, i.e. it can be deployed as a single unit without depending on other services [13]. Thus, the property number of interfaces the service requires gives us a good indication on the self-containment of the service.

The higher the number of required interfaces is, the less the service is self-contained.

- A service is subject to composition with other services. This composition is realized without internal modifications but through service interface. A decomposition of the legacy system will be effective with the principle of composing those services with high cohesion and loose coupling, i.e. two services are composed with each other if their interfaces are cohesive. Thus, the average of services' cohesion within an interface gives us a good indication on the composability of the service.
- A service is more likely to be coarse-grained and hence represent complex, rich and high-level business functionality. However, it may sometimes be fine-grained and hence represent low-level primitive functionality [14]. Choosing the right level of granularity is the key for a successful service reuse. The bigger the service grains are, the less the service becomes reusable. It is relatively difficult to determine from source code the exact number of functionalities that the service provides. However, several factors can help measuring the functionality of a service. (1) A service that provides several interfaces may

provide numerous functionalities, thus the higher the number of interfaces is, the more the service provides functionalities. (2) An interface whose services are highly cohesive probably provide single functionality. (3) A group of interfaces with high cohesion are most favorable to provide single or limited number of functionalities. (4) When the extracted code of candidate service is highly coupled, this means that the service probably provides very few or single functionality. (5) When the extracted code of candidate service is highly cohesive, this means that the service probably provides very few or single functionality. Thus, we suggest binding the functionality characteristic to properties as indicated in Table II.

TABLE II. BINDING FUNCTIONALITY CHARACTERISTIC TO PROPERTIES

Functionality Characteristic	Property
A service that provides several interfaces may provide numerous functionalities, thus the higher the number of interfaces is, the more the service provides functionality.	Number of provided interfaces
An interface whose services are highly cohesive probably provide single functionality.	Average of service's interface cohesion within the interface
A group of interfaces with high cohesion are most favorable to provide single or limited number of functionality.	Cohesion between interfaces
When the extracted code of candidate service is highly coupled, this means that the service probably provides very few or single functionality.	Coupling inside a service
When the extracted code of candidate service is highly cohesive, this means that the service probably provides very few or single functionality.	Cohesion inside a service

### 3) The Quality Metrics

In our approach, according to the characteristics and properties of services we have chosen above, we build our quality metrics to evaluate the quality of candidate services. This quality will be the factor in distinguishing the extracted candidate services. The property functionality requires coupling and cohesion measurements, while composability only requires a cohesion measurement (see Figure 2). As to [15], cohesion of a service measures how strong the elements within this service are related to each other. A service is considered as highly cohesive, if it performs a set of closely related functions and cannot be split into finer elements. The metric LCC Loose Class Cohesion proposed by [16] measures the overall connectedness of the class. It is calculated by:

$$LCC = \frac{\text{number of direct and indirect connections}}{\text{maximum number of possible connections}}$$

Coupling means the degree of direct and indirect dependence of a class on other classes in the system. Here, two measures are counted: method calls and parameter use, i.e. two classes are considered coupled to each other if the methods of one class use the methods or attributes of the other class. In our approach,  $Coupl(E)$  measures the internal coupling of the

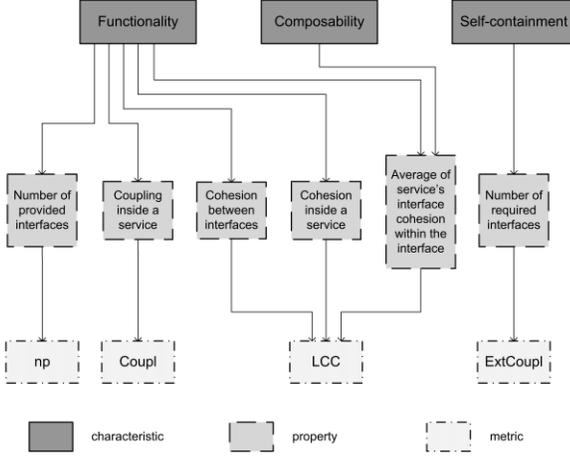


Figure 2. Refinement model of service characteristics.

candidate service  $E$  and is calculated by the ratio between number of classes inside the service that are internally called to the total number of classes within the candidate service  $E$ .  $ExtCoupl(E)$  measures the coupling of the candidate service  $E$  with other services. It is calculated as  $ExtCoupl(E) = 1 - Coupl(E)$ .

#### 4) Fitness Function Definition

We define a fitness function  $FF(E)$  for an identified candidate service  $E$  as a linear combination between the 3 characteristics of services previously defined,  $F(E)$  for functionality,  $C(E)$  for composability and  $S(E)$  for self-containment as follows:

$$FF(E) = \frac{\alpha F(E) + \beta C(E) + \gamma S(E)}{n}$$

Where  $\alpha, \beta, \gamma$  are coefficient weights for each characteristic that are determined by software architect and  $n = \sum(\alpha, \beta, \gamma)$ .

The characteristics functionality  $F(E)$ , composability  $C(E)$  and self-containment  $S(E)$  are measured according to their definition as follows:

$$F(E) = \frac{1}{5} (np(E) + \frac{1}{I} \sum_{i \in I} LCC(i) + LCC(I) + Coupl(E) + LCC(E))$$

Where  $np(E)$  refers to number of provided interfaces,  $LCC(i)$  refers to the average of service's interface cohesion within the interface,  $LCC(I)$  refers to the cohesion between interfaces,  $Coupl(E)$  refers to the coupling inside a service, and  $LCC(E)$  refers to the cohesion inside a service.

$$C(E) = \frac{1}{I} \sum_{i \in I} LCC(i); \text{ where } i \text{ refers to interface}$$

$$S(E) = ExtCoupl(E)$$

---

**Input:** OO source code classes;  
**Output:** A hierarchy of clusters (dendrogram);

---

- 1: *let each class be a cluster;*
- 2: *compute fitness function of pair classes;*
- 3: **repeat**
- 4: *merge two "closest" clusters based fitness function value;*
- 5: *update list of clusters;*
- 6: **until** *only one cluster remains*
- 7: **return** *dendrogram*

---

### C. Clustering Process

In order to recover services from OO legacy code, we group classes based on their dependencies. For that purpose, we propose a hierarchical agglomerative clustering algorithm. This algorithm groups together the classes with the maximized value of the fitness function. At the outset, every class is considered as a single cluster. Next, we measure the fitness function between all pairs of clusters. The algorithm merges the pair of clusters with the highest fitness function value into a new cluster. Then, we measure the fitness function between the new formed cluster and all other clusters and successively merge the pair with the highest fitness function value. These steps are repeated as long as the number of clusters is bigger than one, as illustrated in Pseudo code 1. As a result, the legacy system is expressed in hierarchical view presented in a dendrogram, as illustrated in Figure 3.

To obtain a partition of disjoint clusters, the resulting hierarchy needs to be cut at some point. To determine the best cutting point we employ the standard depth first search (DFS) algorithm. Initially on the root node, we compare the similarity of the current node to the similarity of its child nodes. If the current node's similarity value exceeds the average of similarity value of its children, then the current node is a cutting point, otherwise, the algorithm continues recursively through its children.

By applying the aforementioned clustering algorithm, we evaluate the legacy system and represent its classes in coarse-grained and loose-coupled disjoint set of services. An example of partitioning legacy system's classes to services is illustrated in Figure 3.

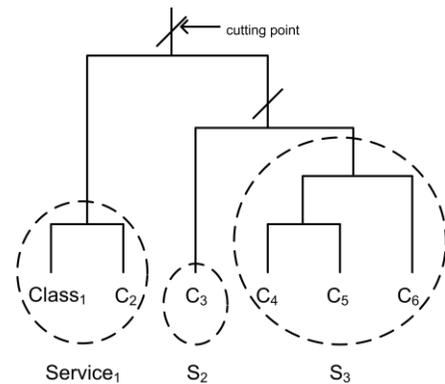


Figure 3. Dendrogram with set of services.

## IV. EVALUATION

The proposed approach has been evaluated on two realistic case studies: Java Calculator Suite<sup>1</sup> which is a small system with 17 classes and MobileMedia<sup>2</sup> which is a medium sized system with 51 classes. Table III gives the number of classes and LOC (Line of Code) of these two case studies.

TABLE III. CASE STUDIES INFORMATION

Case study	Number of classes	LOC
Java Calculator Suite	17	2360
MobileMedia	51	3016

Java Calculator Suite is an open-source calculator implemented in Java. It performs basic mathematical operations, has a graphic interface and supports Booleans, large numbers, machine numbers, and about 25 different operations. MobileMedia is an open-source Java application used for managing media (photo, music, and video) on mobile devices.

### A. Service Identification

#### 1) Results

In this phase, we partition the source code of each case study into a set of clusters. Each cluster is composed of one or more classes. Each resulting cluster corresponds to one service. Table IV shows the results in terms of number of obtained services for each case study and the corresponding average service quality value for each of the three characteristics: functionality, composability and self-containment. The distribution rate of classes to services is  $17/7 = 2.4$  classes per service for Java Calculator Suite and 3.9 for MobileMedia. Even more, we notice that almost all classes of same service are grouped to offer single functionality. For example, “Entries”, “GuiCommandLine” and “ResultList” handle I/O issues.

TABLE IV. SERVICE IDENTIFICATION RESULTS

Case study	Number of services	Functionality	Composability	Self-containment
Java Calculator Suite	7	0.73	0.88	0.41
MobileMedia	13	0.60	0.79	0.59

#### 2) Validation and Discussion

We validate the consistency of our proposed approach either by comparing resulting services with the known architectural design or by analyzing the relevance of the identified services.

In Java Calculator Suite, we notice that lexically relevant classes were grouped in one cluster. We document the resulting components by assigning a name based on the most frequent tokens in their classes’ names. In Table V, we display service identification results in terms of clusters’ names and their composing classes.

TABLE V. SERVICE IDENTIFICATION RESULTS

Cluster Number	Cluster Name	Composing Classes
1	Calc Machine Number	CalcMachineNumber
2	Operator Center Control	OperatorControlCenter
3	Calculator Gui Results Command	Calculator CalculatorException, CalculatorTester Jcalc jcalc_applet
4	E jcalc_math jcalc_trig	E jcalc_math jcalc_trig variable_interface
5	Variable operator Checker Table	VariableTable, operatorChecker
6	PI	PI
7	Gui Line Results Command List	Entries GuiCommandLine ResultsList

MobileMedia has a known architecture model. In [17], the authors presented aspect oriented architecture for MobileMedia. We manually compare our extracted services with the modules of this design, after excluding aspect modules. We have found out that some services were directly mapped to one corresponding module in the architecture, such as the service that includes two classes “MediaListScreen” and “MediaData” was mapped to the module named “MediaListScreen”. In total, 5 services were successfully mapped to 5 modules. Some other extracted services could be mapped to more than one module. This category can be divided to two types. The first type is one module with closely related functionalities such as the service named “Video Media Util Screen Play Capture Music” was mapped to three modules “PlayMediaScreen”, “VideoAccessor” and “VideoAlbumData”. These three modules are in fact functionally related and the resulted service was more coarse-grained than the architecture design. The second type is modules that are weakly related. For this case, we have found two services that each of them was mapped to respectively 3 and 4 modules of the architecture. Some services that are functionally closely related (in our case study, 4 services related to the functionality of transferring media via SMS) were mapped to many modules of the architecture (in our case study, 2 modules related to media transfer functionality). These extracted services were finer-grained than their corresponding modules. Finally, one service that groups exception classes is missing from the architectural design since in the architecture, non-functional modules are not represented.

The results show that 77% (10/13) of extracted services were successfully mapped in the architectural design.

### B. Example of Service Deployment

We deployed identified services as Web services using Apache Axis2 on Apache Tomcat Web server and then wrapped these Web services by generating their WSDL interfaces. For example, in MobileMedia case study, the configuration file services.xml (Figure. 4) describes the “mapping” between Web service “Video Media Util Screen

<sup>1</sup> <http://sourceforge.net/projects/bfegler/>

<sup>2</sup> <http://homepages.dcc.ufmg.br/~figueiredo/spl/icse08/>

Play Capture Music” and the Java classes composing this service.

```

<service name="VideoMediaUtilScreenPlayCaptureMusic"
scope="application">
  <description>
    Video Media Util Screen Play Capture Music
  </description>
  <messageReceivers>
    <messageReceiver
      mep="http://www.w3.org/2004/08/wsdl/in-only"
      class="org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver"/>
    <messageReceiver
      mep="http://www.w3.org/2004/08/wsdl/in-out"
      class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
  </messageReceivers>
  <parameter name="ServiceClass">
    sample.pojo.service.VideoMediaUtilScreenPlayCapture
    Music
  </parameter>
</service>

```

Figure 4. Services.xml file.

## V. CONCLUSION

The main contribution of the work presented in this paper is the extraction of services from legacy source code based on service quality characteristics. For this purpose, we first set a mapping model between object and service concepts. Then, unlike most ad-hoc identification approaches, we introduced a fitness function that measures the quality of identified services. The measurement metrics of fitness function are based on a refinement model of service’s semantic characteristics. It is worthy to note that this approach is especially applicable to modernize legacy systems for which no software assets but the source code is available. Finally, to demonstrate the applicability of our proposed approach, we have applied it on two Java OO applications and obtained satisfying results. As a part of future work, we plan to apply our proposed on more complex case studies.

## REFERENCES

- [1] Sneed, H.M., "Integrating legacy software into a service oriented architecture," *Proceedings of the 10th European Conference on Software Maintenance and Reengineering, 2006. CSMR 2006*. pp.11 pp.,14, 22-24 March 2006.
- [2] Brown, A; Johnston, S.; & Kelly, K. "Using Service-Oriented Architecture and Component-Based Development to Build Web Service Applications". Santa Clara, CA: Rational Software Corporation, 2002.
- [3] Feng Chen; Shaoyun Li; Yang, H.; Ching-Huey Wang; Chu, W.C.-C., "Feature analysis for service-oriented reengineering," *12th Asia-Pacific Software Engineering Conference, 2005. APSEC '05*, pp. 201-208, 15-17 Dec. 2005.
- [4] Khadka, R., Saeidi, A., Jansen, S., Hage, J, "A structured legacy to SOA migration process and its evaluation in practice", *Proceedings of the 7th International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA 2013)*.
- [5] Zhang, Z., Liu, R., Yang, H., "Service identification and packaging in service oriented reengineering". *Proceedings of the Seventeenth International Conference on Software Engineering and Knowledge Engineering SEKE'2005*, Taipei, Taiwan, Republic of China, July 14-16, 2005.
- [6] Feng Chen; Zhuopeng Zhang; Jianzhi Li; Jian Kang; Yang, H., "Service Identification via Ontology Mapping," *33rd Annual IEEE International*

*Computer Software and Applications Conference COMPSAC '09*, vol.1, pp.486,491, 20-24 July 2009.

- [7] G. Lewis, E. Morris, L. O’ Brien, D. Smith, L. Wrage, "Smart: The service-oriented migration and reuse technique," CMU/SEI, Tech. Rep. CMU/SEI-2005-TN-029, Sept 2005, Available from: <http://www.sei.cmu.edu/reports/05tn029.pdf>
- [8] Oracle Corporation, 2008. Business process management, service-oriented architecture, and web 2.0: Business transformation or train wreck? Oracle Corporation, White Paper, available online from <http://viewer.media.bitpipe.com/934318651-120/1252521184-411/SOA-US-EN-WP-BPM2008.pdf>.
- [9] Khadka, R.; Reijnders, G.; Saeidi, A.; Jansen, S.; Hage, J., "A method engineering based legacy to SOA migration method," 2011 27th IEEE International Conference on Software Maintenance (ICSM), pp.163,172, 25-30 Sept. 2011 doi: 10.1109/ICSM.2011.6080783
- [10] Cetin, S.; Ilker Altintas, N.; Oguztuzun, H.; Dogru, A.H.; Tufekci, O.; Suloglu, S., "Legacy Migration to Service-Oriented Computing with Mashups," *International Conference on Software Engineering Advances ICSEA 2007*, pp.21, 25-31 Aug. 2007.
- [11] R. Khadka, A. Saeidi, A. Idu, J. Hage, S. Jansen, "Legacy to SOA evolution- a systematic literature review," in *Migrating Legacy Applications: Challenges in Service Oriented Architecture and Cloud Computing Environments*, A. D. Ionita, M. Litoiu, and G. Lewis, Eds. IGI Global, 2012, pp. 40–71.
- [12] ISO/IEC 25010:2011, Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuARE) - System and software quality models, 2011.
- [13] Nakamura, M.; Igaki, H.; Kimura, T.; Matsumoto, K.-i., "Extracting service candidates from procedural programs based on process dependency analysis," *IEEE Asia-Pacific Services Computing Conference, APSCC 2009*, pp.484,491, 7-11 Dec. 2009.
- [14] Channabasavaiah, K., Holley, K., Edward M. Tuggle, Jr., "Migrating to a service-oriented architecture", White Paper, IBM Corporation - Software Group, April 2004.
- [15] Patidar, M. K., Gupta, R., & Chandel, G. S., "Coupling and Cohesion Measures in Object Oriented Programming". *International Journal of Advanced Research in Computer Science and Software Engineering*, Volume 3, Issue 3, March 2013, ISSN: 2277 128X.
- [16] Bieman, James M.; Kang, Byung-Kyoo: Cohesion and Reuse in an Object-Oriented System. In *Proc. Int'l Symp. Software Reusability*, 1995, S. 259-262.
- [17] E. Figueiredo et al., "Evolving software product lines with aspects: an empirical study on design stability", *Proc. of ICSE*, pp. 261-270, 2008.
- [18] Stehle, E., Piles, B., Max-Sohmer, J., & Lynch, K. "Migration of Legacy Software to Service Oriented Architecture" *Department of Computer Science Drexel University Philadelphia, PA 19104* (2008): 2-5.
- [19] Matos, Carlos M. P. ; Heckel, Reiko, "Migrating Legacy Systems to Service-Oriented Architectures" In *ECEASST'16* .2008.
- [20] Mike P. Papazoglou ; Willem-Jan Heuvel, "Service oriented architectures: approaches, technologies and research issues". *The VLDB Journal*, vol.16, n.3, July 2007, pp. 389-415.
- [21] O'Brien, L.; Smith, D.; Lewis, G., "Supporting Migration to Services using Software Architecture Reconstruction," *13th IEEE International Workshop on Software Technology and Engineering Practice*, 2005, pp.81,91.
- [22] Kebir, S.; Seriai, A.-D.; Chardigny, S.; Chaoui, A., "Quality-Centric Approach for Software Component Identification from Object-Oriented Code," *Joint Working IEEE/IFIP Conference on Software Architecture (WICSA) and European Conference on Software Architecture (ECSA)*, pp.181,190, 20-24 Aug. 2012
- [23] Aldris, A.; Nugroho, A.; Lago, P.; Visser, J., "Measuring the Degree of Service Orientation in Proprietary SOA Systems," *IEEE 7th International Symposium on Service Oriented System Engineering (SOSE)*, pp.233,244, 25-28 March 2013

# Automated Metrics Collection for IEC61131-3 Languages from Engineering Tools

Mani Shankar  
ABB GISL  
Bangalore, India - 560048  
Email: mani.shankar@in.abb.com

Anil Nair  
ABB Corporate Research  
Bangalore, India - 560048  
Email: anil.nair@in.abb.com

**Abstract**—Quantitative metrics is useful for improving software development process. Automatic capturing of metrics from development projects enhances the usage of software metrics. This paper presents an automatic metrics capturing tool and its implementation to capture different software product metrics during product development using IEC 61131-3 languages. We are discussing about the method to capture the metrics for function block and structured text languages of IEC 61131-3. We also discuss about the normalization of different languages in IEC61131-3.

**Keywords**—Software metrics, size metrics, control application, automation application, Tool development

## I. INTRODUCTION

Complexity of Software logic in control / automation applications is increasing rapidly due to improved capabilities of field devices, usage of more powerful programmable controllers, etc. IEC61131-3 [1] languages are used to develop majority of these software applications. Control / automation applications are being predominantly developed by the functional / domain experts, who often have limited understanding of the software engineering practices used in the industry, due to the lack of software engineering background. To manage the software development in this complex scenario, we need the support of good software engineering practices. According to IEEE Standard 610.12, software engineering is defined as, “the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software” [2]. Software metrics are used by the software engineering community for quantitative management of the development, operation and maintenance of software [3]. It helps the industry to get quantitative information about developer productivity, quality, complexity and maintainability of software. The selection of metrics depends on the goal we want to achieve. They give us the status of the project in quantitative manner that is used to track the goal in objective manner and can help to compare the process and product with respect to the goals to achieve. This information can also be used to take corrective and preventive actions as well as to plan definite improvement and strategy plans on software development [4].

In control and automation field, one can find very few works related to software metrics. Gharieb [5] defined 4 metrics specific to ladder logic programming. Dubey [6] reviewed the need for software engineering in context of control and automation. She pointed the need of normalization between

different IEC61131-3 languages. Based on these needs, Nair [4] defined a set of product metrics, which are applicable to all IEC61131-3 languages. To know about the status of a project with respect to goals, we might need the number of lines of code or POU's, its complexity, the functionality it covers, the number of potential defects it may have. However, these metrics can only be used when tools are available for the developer to extract the required data without spending much effort.

The unavailability of software metrics tool for IEC61131-3 languages is a major drawback in estimating and tracking productivity of software project development using IEC 61131-3 languages. In this paper, product metrics such as size, reuse and coupling are implemented for estimating and tracking productivity of software project development using IEC 61131-3 languages. This tool is helpful to managers and developers in optimizing resources during development and maintenance of project developed using IEC61131-3 languages.

The rest of the paper is organized as follows. We introduce the working principle of the tool along with the basic concepts of IEC 61131-3 in section II. Next we explain the implementation of the tool for the IDEs in section III. We describe the conclusion and future work in section IV.

## II. METHODOLOGY

Most of the development environments (IDEs) like CoDeSys [7], [8], Control Builder [9] etc., save the code of control application project in native format and then convert it to an intermediate format to export the code. In our approach, we parse this intermediate format to collect the information required to calculate the metrics. Figure 1 depicts the block diagram of the methodology followed in our tool.

### A. Intermediate format

Some of the IDEs like control builders etc., use proprietary intermediate formats in XML, and other IDEs use open source intermediate formats like PLCopen XML, which is a XML representation. The W3C consortium calls XML “a common syntax for expressing structure in data” [10]. Structured data refers to data that is tagged for its content, meaning, or use. The XML data format PLCopen XML specified by the PLCopen association has the purpose to store and exchange all relevant programming information for IEC 61131-3 PLC programming projects. Thereby, it covers the exchange of the five IEC 61131-3 programming languages, graphical information for

the graphic based programming languages and the structure of programming projects as well as supplier specific information [11]. In this paper, we are talking about the metrics capturing tool which uses propriety XML schema in place of PLCopen XML. However, this tool can easily modified for PLCopen.

### B. Introduction of IEC61131-3

The standard includes both the common concepts already in use in PLC programming and additional new programming methods. IEC 61131-3 defines a guideline for PLC programming. The standard was established by working group SC65B WG7 (originally: SC65A WG6) of the international standardization organization IEC (International Electrotechnical Commission) which consists of representatives of different PLC manufacturers, software houses and users [12]. The IEC 61131-3 is a worldwide standard for PLC programming in recent years.

IEC 61131-3 provides three textual languages and three graphical languages for writing application programs.

The textual languages are [12]:

- Instruction list (IL)
- Structured Text (ST)
- Sequential Function Chart (SFC-textual version).

The graphical languages are:

- Ladder Diagram (LD)
- Function Block Diagram (FBD)
- Sequential Function Chart (SFC - graphical version)

### C. POU

POUs are the smallest independent software units of a user program. IEC 61131-3 calls the blocks from which programs and projects are built as Program Organization Units (POUs).

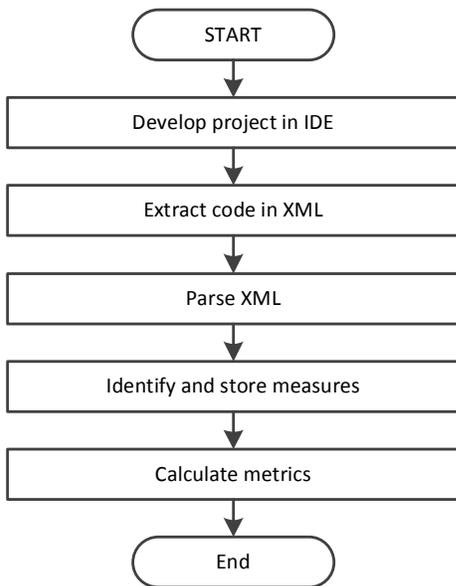


Fig. 1: Flow chart of obtain metrics from IDEs

POUs correspond to the program blocks, organization blocks, sequence blocks and function blocks of the conventional PLC programming world. There are three types of POUs: Function (FUN), Function block (FB) and Program (PROG). Functions always produce the same result (function value) when called with the same input parameters, i.e. they have no “memory” but function blocks have their own data record and hence “remember” status information (instantiation).

These POU types differ from each other in certain features:

*Function (FUN)* : A Function whenever invoked with same input, will generate same output [12].

*Function blocks (FB)* : A Function Block (for example a counter or timer block), when invoked with the same input parameters, will output values depending on the state of its internal (VAR) and external (VAR\_EXTERNAL) variables; these variables are retained from one execution of the function block to the next [12].

*Program (PROG)* : This POU represents the “main program”. The variables used in the program which need to be assigned physical addresses (for example PLC inputs and outputs) must be declared in this POU or above it (Resource, Configuration). Except this point, it behaves like an FB [12].

### D. Metrics Implemented

The tool discussed in the paper implements three product metrics - size, reuse and coupling which were defined in [4].

*1) Size Metrics:* By representing the project using a quantitative size, we can calculate the productivity (i.e., effort taken by the developers to develop a typical POU of unit size). Following parameters were considered for defining the size metrics.

*Base Functional complexity ( $C_{POU}$ )* : This parameter is arrived from the time required to configure a standard POU by an average resource. The value for each POU is defined by the experts in a scale of 1 to 10.

*No of additional operands ( $N_{AOP}$ )* : This identifies additional parameters that will provide the incremental changes to the Base functional complexity.

*Complexity of module ( $C_M$ )* : While calculating the size, we need to consider the additional efforts spent on arranging POU inside the module. Depending upon the complexity of the logic, the relationships between the POUs inside a control module/type circuit/program may be easy or difficult to configure. This parameter will provide the details regarding the effort required for combining different POUs to arrive at the module.

*Number of physical I/O ( $N_{IO}$ )* : Number of physical I/Os connected to the project.

Using the above four parameters, we can arrive at the project size. The steps for determining size are as follows.

*Module Size,*

$$S_M = C_M \times \sum_1^{N_{POU}} [C_{POU} \times (1 + \alpha \times N_{AOP})] \quad (1)$$

*Project Size,*

$$S_P = \left( \sum_1^{N_M} S_M \right) + (\beta \times N_{IO}) \quad (2)$$

Here  $\alpha$  is the size constant for operands,  $\beta$  is the size constant of physical I/Os and  $N_M$  is the number of modules.

2) *Reuse Metrics*: Two Metrics are defined for reuse - reuse count and reuse size. The metric reuse ratio assesses the percentage of POUs reused in the project. Reuse ratio of the project can be arrived by the formula,

$$[RE_P]_{Count} = \frac{[N_{POU}]_{Reuse}}{N_{POU}} \quad (3)$$

$$[RE_P]_{Size} = \frac{[S_{POU}]_{Reuse}}{S_P} \quad (4)$$

Where the base measures are,

*Number of re-used POU* ( $[N_{POU}]_{Reuse}$ ): Count of POUs reused in the project. (POUs used more than once)

*Total number of POUs* ( $N_{POU}$ ): Total number of POUs including the reused POU

*Size of re-used POU* ( $[S_{POU}]_{Reuse}$ ): Sum of size of Program Organization Units (POU) reused in the project. (POUs used more than once) (Calculate using Eq. 1)

*Total size of project* ( $S_P$ ): Size of the total project (Calculate using Eq. 2)

3) *Coupling Metrics*: Coupling metrics [13] is the measure of dependencies of a given module onto the other modules that the project consists of. Using this metrics, we can measure stability, maintainability, and testability of the whole project by considering it as a collection of modules.

*Afferent coupling (fan-in)* ( $C_a$ ): describes the number of POUs outside a module that depend on POUs inside the module. Greater this value, larger is the module's responsibility in maintaining the overall project stability.

$$C_a = \text{Count of POUs having direct inputs from the POUs in the module} \quad (5)$$

*Efferent coupling (fan-out)* ( $C_e$ ): is the number of POUs inside a module that depend on POUs outside the module; an indicator of the package's independence. It builds a unique set for these dependencies by considering all modules that the entity depends on.

$$C_e = \text{Count of POUs having direct inputs to the POUs in the module} \quad (6)$$

*Instability* ( $I$ ): is an indicator of the function's resilience to change. The range for this metric is 0 to 1, where, 0 indicates a completely stable package and 1 indicates a completely unstable package.

$$\text{Instability, } I = \frac{C_e}{(C_a + C_e)} \quad (7)$$

### III. IMPLEMENTATION OF TOOL

#### A. Steps for calculating metrics

According to IEC 61131-3 the basic building blocks are function, function block and program. The tool first reads XML of project, after reading XML the tool will parse XML and collect all measures we require to calculate the metrics. The measures at all three level are collected from the project written using IEC61131-3 languages. Working principal of the tool is explained below.

Step 1. Read the XML of project from development IEDs.

Step 2. Parse XML and collect following data at different levels:

*Data collection on function block level*: The data collected by the tool at function block level are name of the project, name of the application, name of the program, name of the function block, function block type, no of inputs, no of outputs, input extension possibility, name of the variable attached, and variable type with each function block.

*Data collection on Program level*: The data collected by the tool at program level are the name of project, name of the application, name of the program, name of the function block, function block type, no of inputs, no of outputs on program level.

*Data collection on Application level*: The data collected by the tool at application level are Name of the project, name of the application, name of the variable.

Step 3. Collect base complexity of each Function block and function and base complexity of each Program from the users. This will be a onetime activity for an organization.

Step 4. Calculate metrics as mathematical formula defined in previous equation.

Step 5. Show the metrics value (output) to user.

Step 6. Implementation of metrics

#### B. Metrics implementation for Function Block Diagram (FBD)

According to IEC 61131-3 the basic building blocks are function, function block and program. Our tool collects data from project written using Function Block Diagram (FBD) language at all three levels. In figure 2, a sample project written in FBD language is depicted. Sample project has one application, *Application\_1*. *Application\_1* has two programs, *Program\_1* and *Program\_2*. *Program\_1* has 2 AND blocks (one with 2 inputs and another with 3 inputs) and 1 OR (3 input) block. *Program\_2* has 1 AND (2 input) block and 1 Ton timer. The variable defined in *Program\_1* and *Program\_2* are local to respective programs and variables defined at application level are available for both *Program\_1* and *Program\_2*. Figure 2c shows relationship between *Program\_1* and *Program\_2*.

Table II and I show the tools output measures (i.e. data collected at different level) and metrics values and metrics calculated by the tool. For recognizing function Block in program, a unique ID is given to each function and function Block (as shown in table II, *Program\_1* has 2 AND Logic Blocks, 1<sup>st</sup> has ID1 and 2<sup>nd</sup> has ID2). The tool also considers the physical (input/output from controller) and extra input/output attached with function blocks, because each extra and physical input/output affect the function blocks complexity because developer will spend extra time to configure them.

**Example:** If we consider an AND block with 3 inputs to calculate metrics,

$$\begin{aligned} \text{Base complexity of AND Block, } C_{POU} &= 1 \\ \text{Size constant for operands, } \alpha &= 0.1 \\ \text{Complexity of module, } C_M &= 1 \\ \text{Size constant of physical I/Os, } \beta &= 0.2 \\ \text{No of additional operands, } N_{AOP} &= 1 \\ \text{Using these values, size of AND block with 3 inputs} \\ &= C_{POU} + \alpha \times N_{AOP} = 1.1 \end{aligned}$$

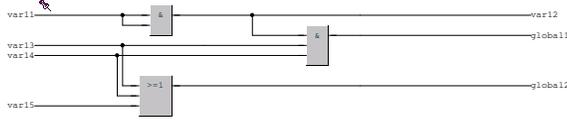
Similarly the tool also calculates size of each FB, which is shown in table II. Using these values in equation 1,

$$\begin{aligned} (S_M)_{Prog1} &= 3.2 \\ (S_M)_{Prog2} &= 4.0 \end{aligned}$$

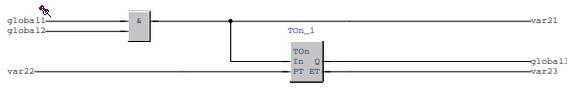
and, using equation 2, Size of the project,

TABLE I: Project Metrics calculated for sample FBD project

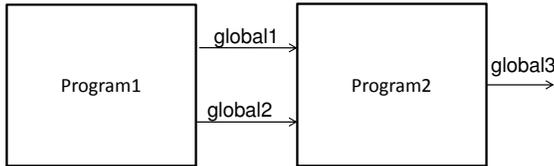
Project	Test project	
Application	Application_1	
Program	Program 1	Program 2
Program Size	3.2	4
Afferent coupling	0	2
Efferent coupling	2	1
Instability	1	0.33
REUSE(COUNT)	0.67	0
REUSE(SIZE)	0.65	0



(a) Program 1



(b) Program 2



(c) Relation between programs

Fig. 2: Sample project written in FBD

$$S_P = 7.2$$

The tool also calculates the coupling metrics at program level:

For program 2,

$$(C_a)_{Prog2} = 2$$

$$(C_e)_{Prog2} = 1$$

Using equation 7,

$$(I)_{Prog2} = \left(\frac{1}{1+2}\right) = 0.33$$

Similarly, for Program 1,

$$(I)_{Prog1} = 1.0$$

All the metrics calculated by the tool are mentioned in table I.

### C. Metrics implementation for Structure Text (ST)

The tool collects data from project written using structured text (ST) language at all three levels (function, function block and program) as defined in IEC 61131-3. For explaining basic concepts, a sample project written in structure text, is shown below. Sample project has one application, *Application\_1*. *Application\_1* contains one program, *Program\_1*. *Program\_1* has 1 TON timer and “if-else” conditional statement. *Application\_1* has 2 global variables and rest are local variables. Table III and IV provides the results obtained by running the tool on this sample code to collect the metrics.

```
if Start_TempControl then
```

TABLE IV: Project Metrics calculated for sample ST project

Project	Sample_ST1
Application	Application_1
Program	Program1
Program Size	12.3
Afferent coupling	2
Efferent coupling	2
Instability	0.5
REUSE(COUNT)	0.5
REUSE(SIZE)	0.423

```
if Start_TempControl then
    CoolingSetpoint := Sp_Temp + TempDeadband;
    HeatingSetpoint := Sp_Temp - TempDeadband;
if gTemp < HeatingSetpoint - Temp_Hysteresis then
    gHeater_Ord := true;
end_if;
if gTemp > 130 then
    Temp_High := True;
    gHeater_Ord := False;
    Start_TempControl := False;
else
    Temp_High := False;
end_if;
end_if;
```

```
TON1( In := Temp_High,
      PT := time1,
      Q => out,
      ET => time2 );
```

For calculation of metrics for function and function blocks from the code written in structured text, our tool uses the same rules and steps as used for function block diagram. But structure text have extra features like loops (for, while) and conditional statement (if, else). Also simple arithmetic and logical expressions are different from other IEC 61131-3 languages. Hence for size metrics calculation, we need a different approach compared to FBD. Size metrics calculation scheme for ST is explained below with the help of an example.

The tool traverses through the ST code and identifies statements that contain loops and conditional statements. If these statements contain operators, then it will count the number of operators in those statements and will give equal base complexity for each operators (say, Loop1). The base complexities are defined with the help of experts in the domain and is a onetime activity.

For example take an *If* statement from above ST code:

```
if gTemp < HeatingSetpoint - Temp_Hysteresis then
    gHeater_Ord := true;
end_if;
```

Base complexity of Loop1,  $C_{POU} = 0.8$

Number of Loop1 in above “*if*” statement

$$= 2$$

$$\text{Size of “if” statement} := 2 \times *C_{POU} \text{ Loop1} \\ = 2 \times 0.8 = 1.6$$

In case, the statement does not have any operators, then the tool will assign a different base complexity to that statement (say, Loop2). I.e., “If-else” Conditional statement contains only one statement which is called as “Loop2” and “Loop2” will have less base complexity than “Loop1”.

For example take an *If* statement from above ST code

```
If Start_Temcontrol Then
```

Base complexity of Loop2,  $C_{POU} = 0.5$

Number of Loop2 in above “*if*” statement

TABLE II: Measures for sample FBD project

Project	Application	Program	POU	POU Type	POU ID	No Of OutPut	No of Input	No Extra Input	FB Size	II	IO	OI	OO	Physical I/Os
Test_case1	Application_1	Program1	AND	AND	ID 2	1	3	1	1.1	3	0	0	1	0
Test_case1	Application_1	Program1	OR	OR	ID 3	1	3	1	1.1	3	0	0	1	0
Test_case1	Application_1	Program1	AND	AND	ID 1	1	2	0	1	2	1	0	0	0
Test_case1	Application_1	Program2	AND	AND	ID 1	1	2	0	1	0	1	2	0	0
Test_case1	Application_1	Program2	TON_1	TON	ID 2	2	2	0	3	2	1	0	1	0

TABLE III: Measures for sample ST project

Project	Application	Program	POU	POU ID	No Of OutPut	No of Input	POU Size	II	IO	OI	OO	Physical I/Os
sample_ST1	Application_1	Program1	+	ID 6	1	2	0.8	2	1	0	0	0
sample_ST1	Application_1	Program1	:=	ID 1	1	1	0.8	1	0	0	1	0
sample_ST1	Application_1	Program1	:=	ID 2	1	1	0.8	1	0	0	1	0
sample_ST1	Application_1	Program1	Loop1	ID 3	0	2	1	1	0	1	0	0
sample_ST1	Application_1	Program1	Loop1	ID 5	0	2	1	1	0	1	0	0
sample_ST1	Application_1	Program1	-	ID 7	1	2	0.8	2	1	0	0	0
sample_ST1	Application_1	Program1	Loop2	ID 8	0	1	0.5	1	0	0	0	0
sample_ST1	Application_1	Program1	:=	ID 9	1	1	0.8	1	1	0	0	0
sample_ST1	Application_1	Program1	:=	ID 10	1	1	0.8	1	1	0	0	0
sample_ST1	Application_1	Program1	:=	ID 11	1	1	0.8	1	1	0	0	0
sample_ST1	Application_1	Program1	Loop1	ID 4	0	2	1	2	0	0	0	0
sample_ST1	Application_1	Program1	TON1		2	2	3.2	2	2	0	0	0

$$\begin{aligned} &= 0 \\ \text{Size of "if" statement:} &= C_{POU_{Loop2}} \\ &= 0.5 \end{aligned}$$

Similarly size is calculated for loops such as (For, While etc.). For simple arithmetic and logical expressions, total number of operators is counted and identified with unique ID and size metrics is calculated by providing base complexity to each type of operators and summing up the size of each operators.

Considering the difference in development approach, the base complexity provided to the same function block may differ in ST and FBD. In most cases base complexity of a function block in ST is more when compared to the same provided to FBD.

This way, we can calculate the measures and metrics for the entire ST program. The results arrived are given in Tables III and IV.

#### D. Tool Piloting

To explore the usefulness of the tool, we have used the tool to an industrial project. In this project FBD and Structure test is used for developing the logic. It consists of one application, 4 programs, 8 code panes and 50 POU's. With the help of domain experts, we have defined base function complexity measure and operand constant for each POU. Similarly, physical I/O constant and program complexity were also calculated.

The result of tool is correlated with the help of experts. The results are found accurate.

The metrics of programs are almost able to predict the program effort. From the above result we concluded that reuse size metrics give us more accurate figure about re-usability compared to reuse count. Zero reuse metrics mean that all the function blocks in that particular program are unique.

#### E. Normalization between two different IEC61131-3 languages

In practice, most industrial projects use more than one language to develop a project. This is mainly to utilize the best properties of each language (for example, use FBD when standard function blocks and libraries are available; use ST when a new library or function block is to be developed etc.). Each language has got their own benefits. Ladder diagram is preferred for sequential logic implementation, structured text is preferred for mathematical implementation, sequential function chart is used for structuring programs and controlling their execution [12]. To express the complete project in terms of single entity (for example, size of the project is x POU's), we need some sort of normalization of metrics among IEC61131-3 languages. The main objective of normalization is to calculate the productivity of developer more precisely [6].

In this paper a project is represented in terms of basic function blocks, but the base complexity of function block depends on the developers' choice of programming language. Effort required to configure the particular function block varies with programming language. Consider ST and FBD, usually developer may take more time to configure the same function block in ST compared to FBD. So base complexity of the

same function block may be more in structured text compared to function block diagram.

For example, for implementing 3 input “AND” block in FBD, developer have option to increase number of inputs in “AND” block. However, in ST we have to use *AND* block twice to get the same result. Hence developers may have to put more effort for implementing same logic in most of the cases.

In this paper an example is used to explain how different programming language effect metrics values. The sample project is implemented using both FBD and ST and the tool captures the metrics for both. FBD and ST metrics captured by the tool are given in Table I and V respectively.

Structure text code for project (Fig 2) is shown below:

**Program 1**

```
var12:=var11 and var12;
global1:=var12 and var13 and var14;
global2 := var13 and var14 and var15;
```

**Program 2**

```
var21:=global1 And global2;
Ton_1( In := var21,
      PT := var22,
      Q => global3,
      ET => var23 );
```

TABLE V: Project Metrics for sample FBD project in ST

Project	Sample_st2	
Application	Application_1	
Program	Program1	Program2
Program Size	5	4.2
Afferent coupling	0	2
Efferent coupling	2	1
Instability	1	0.33
REUSE(COUNT)	0.8	0
REUSE(SIZE)	0.8	0

It is clear from Tables I and V that change in language affects program size and reuse metrics and does not affect coupling metrics. Size of program 1 and 2 in FBD are 3.2 and 4 respectively, but in case of structured Text size is 5 and 4.2. Hence the developer will take more time to write the same logic in structured text.

This information can be useful while estimating the projects. During estimation, the normalization factor can be used to arrive at the approximate effort required if we know the nature of the program and usage of different languages.

*F. Benefits of usage of the metrics in SDLC*

*Size metrics:* In control and automation, common practice is to express project size in terms of number of input/output loops. For estimation, we will divide a project in three categories according to the complexity of loop (simple, medium and complex). But there is no clear cut definition of simple, medium and complex, so with the help of size metrics we can define these categories in terms of quantitative value and try to explore whether we need to divide a project in more than three categories for better estimation and implementation.

*Coupling Metrics:* This measures dependencies of one module to other module of the project. As dependency of module increases it’s hard to maintain. So we need to decide threshold value of a particular project. If metrics value is going to be beyond the threshold then we can suggest the various possibilities of bringing the metrics value below threshold

for e.g. by trying to split a particular module and reducing dependency etc.

*Reuse Metrics:* Before starting a project development, a team in the organization will build libraries, specific to the project to maximize the reuse and save developers time. Most of organizations will fix a specific goal of re-usability. This metrics will help the manager to keep track of Re-usability. Are we able to achieve re-usability goal or not? If not, then take necessary action to reach that goal.

IV. CONCLUSION AND FUTURE WORK

We have implemented three product metrics size, reuse and coupling for two most used IEC61131-3 languages, Function Block Diagram and Structure Text. Results of the tool are satisfactory. These metrics will be helpful to track productivity of developer’s (effort), keeping track of re-usability in development phase and tracking instability of the module (Program).

We can define more metrics like complexity of the particular module and loops. For calculating size metrics more accurately we have to take into consideration the variables that are repeating within more than one function block. We have to count the total number of variables repeating in each program and subtract the effect from the size, because developer will save effort not to declare those variables again. For deciding threshold for coupling metrics we need to further analyze and find the factor’s on which threshold will depend. In future we can add metrics of reliability which will be helpful in tracking the correctness and robustness of the control algorithm and incremental code churn to track changes between two versions of the software.

REFERENCES

- [1] IEC61131-3. *IEC International Standard. Programmable Controllers. Part 3: Programming Languages*. International Electro technical Commission, Geneva, 2003.
- [2] IEEE Computer Society. Guide to the software engineering body of knowledge (SWEBOK). <http://www.computer.org/portal/web/swebok/html/ch1>, 2013. [Online; accessed 12-march-2013].
- [3] N. E. Fenton and S. L. Pfleeger. *Software metrics: a rigorous and practical approach*. PWS Publishing Co, 1998.
- [4] A. Nair. Product metrics for iec 61131-3 languages. *IEEE Conference on Emerging Technologies and Factory Automation (ETFA 2012)*, sept 2012.
- [5] W. Ghariieb. Software quality in ladder programming. *International Conference on Computer Engineering and Systems*, pages 150–154, 2006.
- [6] A. Dubey. Evaluating software engineering methods in the context of automation applications. *9th IEEE International Conference on Industrial Informatics (INDIN)*, pages 585–590, July 2011.
- [7] CoDeSys static analysis 3.5.2.0, codesys professional developer edition, 2013. <http://www.codesys.com/products/codesys-engineering/professional-developer-edition.html>.
- [8] R. C. Harwell and K. L. Sparks. IEC 61131-3, CoDeSys standardize control logic programming. March 2011.
- [9] ABB Control Builder M, 2013. <http://www.abb.co.in/product/seitp334/a862651a2e6d1704c125716400428d4a.aspx>.
- [10] Introduction to XML. [http://www.w3schools.com/xml/xml\\_whatis.asp](http://www.w3schools.com/xml/xml_whatis.asp), 2013. [Online; accessed 12-march-2013].
- [11] PLCopen. [http://www.plcopen.org/pages/tc6\\_xml/xml\\_intro/](http://www.plcopen.org/pages/tc6_xml/xml_intro/), 2013. [Online; accessed 12-march-2013].
- [12] J. Karl-Heinz and M. Tiegelkamp. *IEC 61131-3: Programming Industrial Automation Systems*. Springer, 2010.
- [13] R. C. Martin. *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2003.

# GUIEvaluator: A Metric-tool for Evaluating the Complexity of Graphical User Interfaces

Khalid Alemerien and Kenneth Magel

*Computer Science Department*

*North Dakota State University*

*Fargo, North Dakota, USA*

*khalid.alemerien@my.ndsu.edu, kenneth.magel@ndsu.edu*

**Abstract**—User interface design metrics assist developers to evaluate interface designs in early phase before delivering the software to end users. This paper presents a metric-based tool called GUIEvaluator for evaluating the complexity of the user interface based on its structure. We defined a metrics model that includes five modified structural measures of interface complexity: Alignment, grouping, size, density, and balance. The results of our tool are discussed in comparison with the subjective evaluations of interface layouts. This paper demonstrates the potential benefits of incorporating automated complexity metrics into the user interface design process. Our findings show that the means of the interface complexity values for both the subjective evaluation and the GUIEvaluator are equal at a significance level 0.01.

**Keywords**—User Interface Design, GUI, Complexity Metrics, Screen Aesthetics, Usability, Automated Tools.

## I. INTRODUCTION

Despite the abundance of usability and interface design principles and guidelines, interface complexity continues to be a pressing Human Computer Interaction (HCI) issue. With the tremendous advances in technology, the user interfaces become complex. Therefore, interface design is a difficult process that needs effective tools and techniques to evaluate interface layouts. One of these techniques is the software metrics. With the rapid advance of software, software metrics have become one of the foundations of software management and essential to the success of software development. Nowadays, software metrics are integrated into programming languages that provide developers statistical data about their code. One of the metrics suites is structural metrics. Of the structural metrics, particularly important are complexity metrics, which are used widely with code. Thus, complexity metrics have played a significant role in software development. Therefore, the growth of interface complexity calls for the growth of the complexity metrics in the interface design process. Although people realize the importance of interface metrics, the complexity metrics field still needs to grow quickly to meet the requirement of interface design. This encourages us to

investigate the importance of applying complexity metrics in interface design.

We need to study and merge the complexity metrics into the interface design environment for two reasons: First, there are several applications of the complexity measures in the software development process such as project size estimation, cost and time estimation, effort estimation, and software maintenance. In order to improve the user interface quality and the project controllability, it is necessary to control the interface complexity by measuring the related aspects. Second, user interface is a key component of any software application. Moreover, good interfaces make the interaction between the human and the software seamless and as effortless as possible. Previous empirical studies have proven that the layout complexity is important to aesthetic perception of user interfaces [11, 12].

Quantitative tools can help developers identify better solutions to user interface design problems and make better decisions when faced with alternative designs [14]. In order to enhance the user satisfaction through well-designed interfaces in early stages, we present five design factors with their metrics based on the structural aspects of the interface layout. Those design factors are: Alignment, grouping, size, density, and balance, which are considered significant influences on interface usability [8]. This metrics suite is supported by a complementary tool called GUIEvaluator, which provides automatic metrics calculation. This tool is used to measure the complexity of an interface layout in order to judge the quality of the interface design.

The ultimate goal of this research is to develop an effective metric-based tool to evaluate the complexity of interface layouts. This quantitative tool may mitigate the cost and time of the evaluation of user interfaces in early stages. The rest of this paper is organized as follows. Section II describes the GUIEvaluator tool and the metrics model in further detail. Section III presents the related work. Section IV shows the collected data and procedure. Section V shows the results of our user study. Section VI discusses the results. Section VII concludes the paper.

## II. RELATED WORK

Several research studies run in the interface design field focus on developing and calculating metrics models manually or semi-automatically. Stickel et al. [2] introduced a method to calculate the visual complexity, which is based on the number of possible interactions or functional elements, number of higher level of structures or organizational elements, and summed entropy of RGB values. Sears [3] developed a layout metric called Layout Appropriateness (LA). Each task needs a sequence of actions and LA metric attempts to calculate the costs of each sequence of actions. Kang and Seong [5] proposed three measures to determine the complexity of UI design: Operation, transition, and screen complexity. In addition, Xing [7] developed metrics which utilize three factors associated with complexity: numeric size, variety of elements, and relation between elements.

Ngo et al. [11, 12] introduced a new aesthetic model which consists of 14 aesthetic measures for screen layouts. Fongling et al. [9] developed a metrics model to evaluate the complexity of web pages, including four design factors: density, alignment, grouping, and size. In addition, Parush et al. [10] developed a numerical model to evaluate the GUI. This model consists of four screen factors: element size, grouping, alignment, and local density. All of the above metrics were calculated manually.

A number of researchers developed semi-automated tools that help them to calculate some aspects of user interface designs. Comber and Maltby [1] developed an application called Launcher. Zheng et al. [4] developed a MatLab tool to calculate image vectors. Sears [6] introduced a metric-based tool called AIDE, which partially automates the designing and evaluating user interface layouts. Miyoshi and Murata [8] developed a numerical tool for evaluating the usability of a screen design based on its complexity measures. González et al. [13] developed a tool called BGLayout which helps to automate the metrics calculations using image processing techniques.

GUIEvaluator extends the ideas presented in previous work in two directions: First, GUIEvaluator is a full automated tool that calculates and analyzes the five structural interface measures. This helps designers to determine which design factor has a high complexity rating and allows designers to evaluate, redesign, and reevaluate their interface designs from within a single interface design environment. The previous tools are semi-automated tools. Furthermore, these tools calculate four design factors. Second, GUIEvaluator incorporates the weight for each design factor using the user preferences. This helps us to consider the human factors as a part of our metrics.

The previous studies do not take into account the importance of design factors.

## III. GUIEvaluator TOOL AND METRICS MODEL

### A. The GUIEvaluator Tool

GUIEvaluator is a tool for supporting the effectiveness evaluation of our metrics model. GUIEvaluator calculates the complexity of the interface layout. By using this tool we can evaluate user interface layouts whether the interface is under development or a part of running applications. GUIEvaluator was developed in Visual Basic 2012. This tool extracts the interface layout information using reflection techniques that are supported by Visual Basic 2012. The extraction and analysis window of GUIEvaluator is shown in Fig. 1.

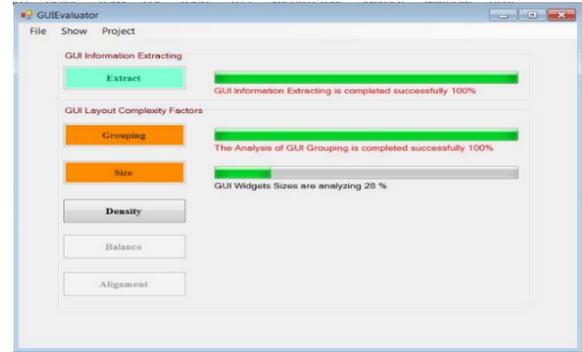


Figure. 1. The extraction and analysis window of the GUIEvaluator

### B. Metrics Model

The purpose of the metrics-model is to reduce the time and costs of subjective evaluation of user interfaces early in the software development. This model consists of five metrics for measuring five design factors: Alignment, grouping, density, balance, and size. The values of these metrics will be used to calculate the interface design complexity. These metrics are the following:

#### 1. Alignment

Alignment metric measures the vertical and horizontal alignment of objects in two levels: A group level (Local Alignment) and screen level (Global Alignment). These two alignment levels are combined to calculate the Total Screen Alignment Complexity.

##### 1.1 Local Alignment

Equation 1 calculates the alignment for each group. Where  $V_p$  is the number of vertical alignment points and  $H_p$  is the number of horizontal alignment points.  $K$  is the number of grouped objects on the

screen. The range of values of GA is [0, 1]. Equation 2 calculates the Alignment Complexity (AC) for all groups on the screen. Where the Weight is the number of objects in a group (i) divided by the total number of grouped objects. And m is the number of groups on the screen.

$$\text{Group Alignment (GA}_i) = \frac{\sum_{i=1}^K (Vp+Hp)}{2K} \quad (1)$$

$$AC = \sum_{i=1}^m \text{GA}_i * \text{Weight}(i) \quad (2)$$

### 1.2 Global Alignment

The global alignment is calculated as shown in Eq. 3, where Vp is the number of the vertical alignment points and Hp is the number of the horizontal alignment points. N is the number of ungrouped objects on the screen. The range of values of the SA is [0, 1].

$$\text{Screen Alignment (SA)} = \frac{\sum_{i=1}^N (Vp+Hp)}{2N} \quad (3)$$

### 1.3 Total Screen Alignment Complexity

Equation 4 shows the Total Alignment Complexity (TAC). Where weight1 is the ratio of the number of grouped-objects to the total number of objects on the screen. Whilst the weight2 is the ratio of the number of ungrouped-objects to the total number of objects on the screen.

$$TAC = AC * \text{weight1} + SA * \text{weight2} \quad (4)$$

## 2. Balance

Balance metric uses the number and size of objects from Eq. 5 and Eq. 6, respectively, for each quarter of screen to calculate the Total Balance Complexity (TBC). The BQni and BQnj represent the number of objects in *ith* and *jth* quarters. The range of ratio of BQni and BQnj is [0, 1] and the range of values of BQn overall is [0, 1], where 0 means unbalanced and 1 means fully balanced in terms of number of objects. The BQsi and BQsj represent the sum of sizes of objects in *ith* and *jth* quarters. The range of ratio of BQsi and BQsj is [0, 1] and the range of values of BQs overall is [0, 1], where 0 means unbalanced, 1 means fully balanced in terms of object size. Equation 7 is the Total Balance Complexity (TBC).

$$BQn = \frac{\sum_{k=1}^6 \frac{BQni}{BQnj}}{6} \quad (5)$$

$$BQs = \frac{\sum_{k=1}^6 \frac{BQsi}{BQsj}}{6} \quad (6)$$

$$TBC = 1 - (0.5 * BQn + 0.5 * BQs) \quad (7)$$

## 3. Density

Density metric measures the screen occupation by objects, where Eq. 8 calculates the Local Density (LDj) for the group *jth* and Eq. 9 calculates the Global Density (GD). In Eq. 10, the Density-Complexity (DC) is calculated taking into account the W1 that is the ratio of the area of groups to the screen area, and W2 is the ratio of ungrouped area to the screen area.

$$LDj = \frac{\sum_{i=1}^{\text{groupedobjects}} \text{Size of object } i \text{ in a group } j}{\text{Area of group } j} \quad (8)$$

$$GD = \frac{\sum_{k=1}^{\text{ungroupedobjects}} \text{Size of object } k}{\text{Area of ungrouped}} \quad (9)$$

$$DC = \left( \frac{\sum_{j=1}^n LDj}{n} \right) * W1 + GD * W2 \quad (10)$$

## 4. Size

Size metric measures the object sizes complexity of two levels: The object size complexity (SC<sub>k</sub>) as in Eq. 11 and the size complexity overall (SC) as in Eq. 12, where N is the number of objects in type *kth* and S<sub>j</sub> is the number of different sizes, where S<sub>j</sub> is 1 if the object size is not counted before and 0 if the object size is counted. W<sub>i</sub> is the number of objects types and Weight (j) is the number of objects in category *jth* divided by the total number of objects on the screen.

$$\text{Object Size Complexity (SC}_k) = \frac{\sum_{j=1}^N S_j}{N} \quad (11)$$

$$SC = \frac{\sum_{k=1}^{W_i} S_{Ck} * \text{Weight}(k)}{W_i} \quad (12)$$

## 5. Grouping

Grouping metric measures the number of objects that have a clear boundary by line, background, color, space, or size. Equation 13 calculates the percentage of ungrouped objects (UG), where the GW is the object that is grouped. The value of GW equals 1, if the object exists in a group, otherwise GW equals 0. N is the total number of objects on the screen. Equation 14 calculates the ratio of the number of different object types (G) to the total number of objects (M) in all groups, where the Weight is the ratio of total number of grouped objects to total number of objects on the screen. To calculate the grouping complexity (GT), we use Eq. 15.

$$UG = 1 - \frac{\sum_{i=1}^N GW}{N} \quad (13)$$

$$GC = \frac{G}{M} * \text{Weight} \quad (14)$$

$$GT = UG + GC \quad (15)$$

#### 6. Overall Screen Layout\_Complexity (LC)

Equation 16 calculates the Overall Screen Layout\_Complexity (LC), which includes the values of TAC, TBC, DC, SC, and GT metrics. Each metric has a weight (w1, w2, w3, w4, and w5, respectively), which are calculated based on the participant rating. The values of those weights are 0.84, 0.76, 0.80, 0.72, and 0.88, respectively.

$$LC = (TAC * w1 + TBC * w2 + DC * w3 + SC * w4 + GT * w5) / 5 * 100\% \quad (16)$$

## IV. METHODOLOGY

### A. Participants

The study was conducted at North Dakota State University. Participants included 50 student volunteers (17 females, 33 males), where 80% of participants were graduate students and 20% were undergraduate students. The participants who are computer science or related fields were 50% and 50% were from other majors. The data that were collected about the participants show 46% of participants have no experience in software development. But 24%, 16%, and 14% of participants have one or two years, three to six years, and six years and above, respectively.

### B. Data Collection

This section presents a brief description of data collected during the study. We developed an application to collect our data. The study took approximately 35 minutes. First, we showed the participants examples that explain the design factors and how to rate them. Second, we asked participants to provide background information. Third, the participants were asked to rate the five design factors (alignment, grouping, density, balance, and size) and the user interface design overall using a 7-point Likert scale. Finally, to obtain the weight for each of the five complexity measures in Eq.16, the participants rated the importance of each measure.

## V. RESULTS AND ANALYSIS

The major objective of this study is to investigate the usefulness and effectiveness of using the GUIEvaluator and its metrics in evaluating the user interface complexity. Therefore, our investigation addresses the following two research questions:

**RQ1:** Is the GUIEvaluator effective to measure the complexity of user interface?

**RQ2:** How do the structural measures of an interface affect the interface complexity rating?

Our hypotheses associated with RQ1 are:

**H1:** *given a specific user interface, the means of interface complexity for the user rating and the GUIEvaluator are equal.*

**H2:** *given a specific user interface, there is a strong positive correlation between the user rating and the GUIEvaluator in terms of interface complexity value.*

To test these two hypotheses, we performed the t-test on (H1) and the Pearson correlation on test (H2) for both the user rating and the GUIEvaluator, at a significance level of 0.01, on 18 interface layouts. Table 1 shows that the means of the user rating and the GUIEvaluator are 0.561 and 0.575, respectively. Furthermore, to test the hypothesis (H1), we performed the t-test for (H1) as shown in Table 1,  $df=32.39$ , for a significance level of 0.01. It shows there is no difference between the means of interface complexity of the user rating and the GUIEvaluator. In addition, to test the hypothesis (H2), we performed the Pearson correlation test. In Table 1, the R value (0.804) shows a strong positive correlation between the user rating and the GUIEvaluator at a significance level of 0.01.

Figure 2 presents the complexity values of 18 interface layouts, which are rated by both the participants and the GUIEvaluator. We have observed strong similarities between the complexity values for both the user rating and the GUIEvaluator. Therefore, our tool can be used to accurately evaluate the complexity of user interfaces.

Our hypothesis associated with RQ2 is:

**H3:** *given a specific user interface, the values of the metrics which exist in our metrics-model are strongly correlated with interface complexity values given by both the users and the GUIEvaluator.*

To test this hypothesis, we performed the Pearson correlation test and the t-test for both the user rating and the GUIEvaluator with the five complexity measures, at a significance level of 0.01, on 18 screen layouts. Table 2 shows the results of the Pearson correlation test and t-test for the values of the five complexity measures and the values of interface complexity given by both the user rating and the GUIEvaluator. On the one hand, Table 2 shows that there is a strong correlation among the user rating and the GUIEvaluator and the following design factors at a significance level of 0.01: Size, alignment, density, and balance. On the other hand, the grouping factor has a weak correlation with the user rating and the GUIEvaluator with R values 0.462 and 0.095, respectively. Therefore, we can accept the hypothesis

Table 1. Pearson correlation test and t-test results between the user rating and GUIEvaluator rating

	User Rating	GUIEvaluator Rating
Mean	0.561	0.575
R		0.804
p-val.		<0.001
t-val.		0.366
df		32.39
p-val.		0.7166

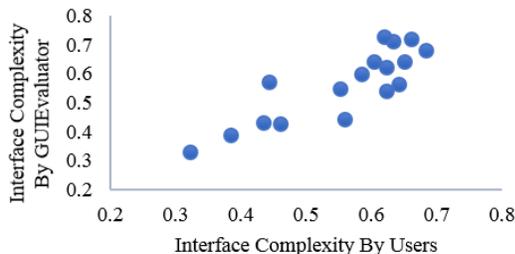


Figure 2. Comparison of the interface complexity values of the user rating and the GUIEvaluator

(H3) for the alignment, size, balance, and density metrics, but we fail to accept it for grouping metric.

Figure 3 compares the values of five complexity measures (size, alignment, density, grouping, and balance) with the interface complexity values of the user rating. From Fig. 3 we have observed the following: First, the interface complexity is strongly affected by the size measure. Our findings show that the size measure has a strong correlation with the interface complexity that was rated by the participants, but the participants rated the size measure less important compared with the rest of measures. Second, another surprising result was that the grouping measure had the highest importance according to the user rating, but our findings show that the grouping measure has the lowest correlation with the values of interface complexity that are rated by the participants. Third, the value of interface complexity appears to increase as the values of balance, density, and alignment measures increase. To sum up, the values of structural measures are consistent with overall interface complexity.

Figure 4 compares the values of five complexity measures (size, alignment, density, grouping, and balance) given by the participants with the interface complexity values given by the GUIEvaluator. By analyzing Fig. 4, we have observed a strong resemblance with the results from Fig. 3. Both results are consistently correlated. Thus, we can claim that our metrics model is effective to evaluate the complexity of user interfaces.

## VI. DISCUSSION

As just outlined, the interface complexity values, which are rated by both the participants and the GUIEvaluator, are consistent. Furthermore, our findings show that our complexity metrics of interface layouts are useful, but these metrics do not have equal magnitude of importance. We investigated that the grouping measure has the less correlation with complexity values for both the user rating and the GUIEvaluator. Perhaps the reasons behind this are: (1) Misunderstanding the objects grouping on the screens, (2) 46% of participants have no experience in software development, or (3) our grouping metric may be insufficient to measure the objects grouping.

In Table 1, our findings show that the R value is 0.804 between the user rating and the GUIEvaluator for 18 interface layouts. However, the interface layouts 3, 8 and 14, we observed that there is a non-trivial difference between the interface complexity values for both the user rating and the GUIEvaluator. The complexity measure that causes this difference is the objects grouping given by user rating. The values of grouping are less than the average of the values of other complexity measures for the interface layouts 3, 8, and 14. This encourages us to focus on grouping measure to investigate the causes behind the variance of the values between the user rating and the GUIEvaluator. In summary, whether we use GUIEvaluator or user rating to evaluate the interface complexity, we reach the same conclusion. Therefore, our findings confirm the effectiveness of our tool and its metrics-model to be used during the early stage software development.

## VII. CONCLUSION

In this paper, we identify a set of structural measures of user interfaces. This metrics-model consists of five metrics: Alignment, grouping, size, density, and balance. This model is supported by a complementary tool called GUIEvaluator, which automatically calculates the values for the metrics. Our findings confirm the effectiveness of our metrics model and GUIEvaluator to measure the complexity of interface layouts. Some issues still need investigating. Our future work will include conducting an experiment to compare our metrics with other existing metrics and extend our model with new structural metrics. Therefore, this paper is the start point of developing our framework that consists of tools and metrics to automate the evaluation of user interfaces in early stages. To sum up, this work addresses the lack of effective tools and techniques to evaluate the complexity of user interfaces during software development. So, automated tools can play a significant role in evaluating of the complexity of user interfaces.

Table 2. Pearson correlation test and t-test results for each user interface design factor with the user rating and GUEvaluator rating

Interface Design Factor	User Rating				GUEvaluator Rating			
	Pearson Correlation Test		T-test		Pearson Correlation Test		T-test	
	R	p-val.	t-val.	p-val.	R	p-val.	t-val.	p-val.
<b>Alignment</b>	0.836	<0.00001	-0.780	0.442	0.662	0.0028	-1.005	0.323
<b>Balance</b>	0.825	<0.00001	-0.560	0.579	0.705	0.0011	-0.857	0.398
<b>Density</b>	0.683	0.0018	-0.772	0.445	0.585	0.0098	-1.050	0.302
<b>Size</b>	0.943	<0.00001	-0.731	0.470	0.767	<0.001	-1.013	0.319
<b>Grouping</b>	0.462	0.0537	-2.608	0.014	0.095	0.708	-2.657	0.012

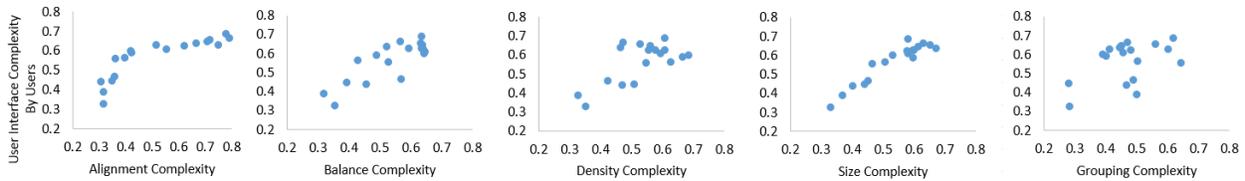


Figure 3. Comparison of the values of five complexity measures with the interface complexity values of the user rating

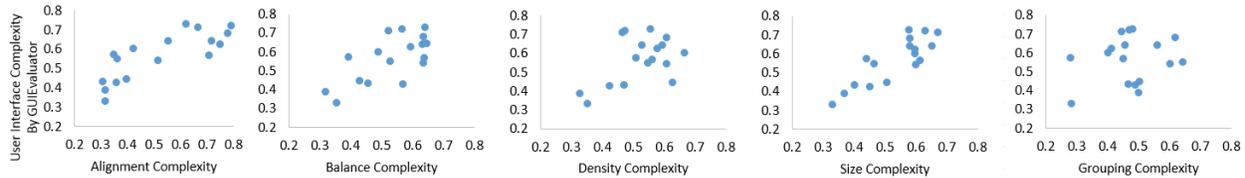


Figure 4. Comparison of the values of five complexity measures with the interface complexity values of the GUEvaluator rating

## REFERENCES

- [1] T. Comber and J.R. Maltby, 1997, "Layout complexity: does it measure usability?," *Proceedings of Interact'97 Conference on Human-computer Interaction (1997)*, pp. 623-626.
- [2] C. Stickel, M. Ebnar, and A. Holzinger, "The XAOS metric: understanding visual complexity as measure of usability," *Proceedings of the 6th international conference on HCI in Work and Learning, Life and Leisure*, Springer, pp. 278-290.
- [3] A. Sears, "Layout appropriateness: A metric for evaluating user interface widget layout," *IEEE Transactions on Software Engineering*, vol.19, no.7, 1993 pp. 707-719.
- [4] X.S. Zheng, I. Chakraborty, J. Lin, and R. Rauschenberger. "Developing quantitative metrics to predict users' perceptions of interface design." In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 52, no. 25, pp. 2023-2027. SAGE Publications, 2008.
- [5] H. G. Kang and P. H. Seong, "An information theory-based approach for quantitative evaluation of user interface complexity," *IEEE TRANSACTIONS ON NUCLEAR SCIENCE*, vol. 45, no. 6, (1998), 3165 – 3174.
- [6] A. Sears, "AIDE: a tool to assist in the design and evaluation of user interfaces." *Interactive Systems Research Center*, Baltimore, 1993.
- [7] J. Xing, (2004). *Measures of information complexity and the implications for automation design* (No. DOT/FAA/AM-04/17). FEDERAL AVIATION ADMINISTRATION OKLAHOMA CITY OK CIVIL AEROMEDICAL INST.
- [8] T. Miyoshi and A. Murata, "A method to evaluate properness of GUI design based on complexity indexes of size, local density, alignment, and grouping," *Systems, Man, and Cybernetics*. In: *IEEE International Conference on*, vol.1, pp.221-226, (2001)
- [9] F. Fongling, C. Shao-Yuan, and H. S. Chiu, "Measuring the screen complexity of web pages." *Human Interface and the Management of Information. Interacting in Information Environments*. Springer Berlin Heidelberg, 2007. 720-729.
- [10] A. Parush, R. Nadir, and A. Shtub, "Evaluating the layout of graphical user interface screens: validation of a numerical computerized mode," *International Journal of Human-Computer Interaction*, vol. 10, no.4, pp. 343-360, (1998).
- [11] D.C.L Ngo, L.S. Teo, and J.G. Byrne, "Formalizing guidelines for the design of screen layouts," *Display* vol. 21, no. 1, pp. 3–15, (2000)
- [12] D.C.L Ngo, L.S. Teo, and J.G. Byrne, "Modeling interface aesthetics," *Information Science* vol. 152, no. 1, 25–46, (2003)
- [13] S. González, F. Montero, and P. González, "BaLOReS: a suite of principles and metrics for graphical user interface evaluation," In *Proceedings of INTERACCION'12*, 13th International Conference on Interacción Persona-Ordenador, 2012.
- [14] L.L. Constantine, (1996) "Usage-centered software engineering: new models, methods, and metrics," In *Software Engineering: Education and Practice*, 1996. *Proceedings. International Conference*, pp. 2-9. IEEE, 1996.

# Improving Static Analysis Performance Using Rule-Filtering Technique

Deng Chen<sup>1</sup>, Rubing Huang<sup>2</sup>, Binbin Qu<sup>1\*</sup>, Sheng Jiang<sup>1</sup>

<sup>1</sup>School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, P.R. China  
{chendeng8899, bbqu, jwt}@hust.edu.cn

<sup>2</sup>School of Computer Science and Telecommunication Engineering, Jiangsu University, Zhenjiang, P.R. China  
rbhuang@mail.ujs.edu.cn

**Abstract**—Static analysis is an efficient approach for software assurance. It is indicated that the most effective usage of it is to perform analysis in an interactive way through software development process, which has a high performance requirement. This paper concentrates on rule-based static analysis tools and proposes an optimized rule-checking algorithm to improve their performance. Our technique filters rules according to their characteristic objects before checking the rules against a specific source file. It is based on an observation that a source file always contains vulnerabilities of a small part of rules rather than all. To investigate our technique’s feasibility and effectiveness, we implemented it in an open source static analysis tool called PMD and used it to perform an evaluation. The evaluation results show that our approach can get an average performance promotion of 28.7% compared with original PMD. Additionally, our technique incurs trivial runtime overhead.

**Keywords**- rule-based static analysis; software quality; software validation; performance improvement

## I. INTRODUCTION

Static analysis is one of the most important techniques for software assurance. It can find potential vulnerabilities as early as in software implementation phase by scanning application programs’ source code, byte code, binary code or other artifacts rather than running the executable files. In recent years, many tools have been developed for automatically finding bugs in program source code, using techniques such as syntactic pattern matching [1], data flow analysis, type systems, model checking, and theorem proving [2]. Just as David Hovemeyer [3] pointed out that integrating bug-finding tools into the development process is an important direction for future research, these tools are not only powerful but also possess the feature to be used in an interactive way through software development process. For instance, Coverity and Klocwork [4] are two powerful software assurance tools, both of which have plug-in versions for Eclipse IDE. However, the plug-in versions are always more lightweight tools, which just carry out shallow analysis due to high performance requirements.

To resolve the performance issue, many techniques have been proposed, such as incremental analysis technique and distributed technology [3]. Incremental analysis automatically infers what parts of source code have to be reanalyzed after the code has been modified. This approach typically reduces the

analysis time substantially, but may of course imply a complete reanalysis in the worst case [4]. Distributed technology is a general approach to cope with efficiency issues, but building a robust distributed system is a complicated and complex task. Aside from above techniques, some other techniques exist, but many of them have side effects on static analysis precision which is a key index of the tools.

In this paper, we aim to improve the performance of rule-based static analysis tools. A rule-based static analysis tool detects vulnerabilities by matching vulnerability rules lexically or syntactically against program source code or other artifacts. We focus on this particular type of tools for several reasons: (1) this kind of tools is used extensively and many recently developed tools are of the kind, such as FindBugs [3, 5, 6, 7], PMD [5, 8, 9], CheckStyle [10], etc.; (2) this kind of tools is able to detect many kinds of vulnerabilities only if the corresponding rules are added; (3) these tools have some unique characteristics which make it possible to improve their performance by novel approaches rather than the above traditional ones.

As Figure 1 illustrates that rule-based static analysis tools usually consist of four parts: compiler front end, vulnerability rules library, rules checking engine and vulnerability reporter. Compiler front end is the main component which is responsible for translating source code into intermediate forms such as *Abstract Syntax Trees (AST)*. Vulnerability rules library contains a set of vulnerability rules. A *vulnerability rule* is a code idiom representing a weakness, the number of which is an important indicator of the tools’ detection capability. Typically, vulnerability rules are described using various kinds of rule description languages, e.g. PQL [11, 12], Datalog [13-16],

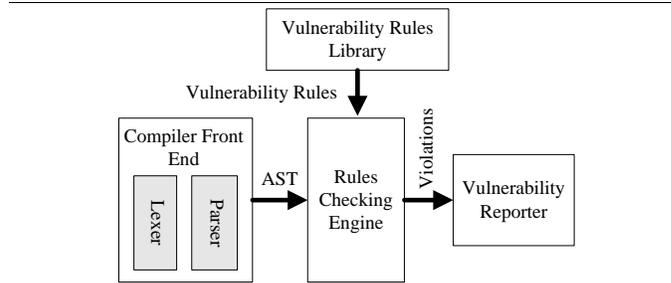


Figure 1. Composition of rule-based static analysis tools

```

<![CDATA[
//TypeDeclaration/ClassOrInterfaceDeclaration
[
  (
    (./ImplementsList/ClassOrInterfaceType
[ends-with(@Image,'SessionBean')]) or
    (./ImplementsList/ClassOrInterfaceType
[ends-with(@Image,'MessageDrivenBean')])
  )
  and not (ends-with(@Image,'Bean'))
]]>

```

Figure 2. Rule `MDBAndSessionBeanNamingConvention` excerpted from PMD. It is described using XPath expression.

XPath expression [17, 18], etc. The functionality of vulnerability reporter is displaying vulnerabilities in different formats. Another key component is the rules checking engine, which finds bugs by checking rules against AST nodes. The *rule-checking algorithm* that it employs is a decisive factor of the tools’ performance especially for a large vulnerability rules library.

To date, most of the rule-checking algorithms are based on *visitor pattern*, that is, each rule checks against the whole AST using depth-first or breadth-first tree searching approach. The problem is that, a large number of unnecessary AST nodes (which are impossible to contain violations of rules) may be visited, which increases time overhead enormously. Based on the observation, we propose a novel rule-checking algorithm. Given a source file  $f$  required to be validated, prior to checking vulnerability rules against the file, our technique first filters out unnecessary rules according to *characteristic objects* (which is defined in later sections). Theoretically, our approach is effective in improving performance, because a source file always contains violations of a small part of rules in the library rather than all. Additionally, our technique incurs little runtime overhead, because we can recognize characteristic objects efficiently by inspecting several lines of code at the beginning of a source file.

To investigate our technique’s feasibility and effectiveness, we implemented our technique in a prototype tool EPMD which developed from an open source static analysis tool called PMD and used it to perform an empirical evaluation. The results of our evaluation show that our approach can improve performance of static analysis effectively with an average promotion of 28.7%. Moreover, we believe that our technique is promising to be more effective when testing on some special subjects or using tools with more vulnerability rules.

The contributions of this paper are:

- A novel rule-filtering technique that can improve the performance of rule-based static analysis tools.
- A formal language which is used to describe characteristic objects of rules.
- A prototype tool EPMD that implements our technique.
- An empirical evaluation that shows the feasibility and effectiveness of our technique.

The rest of this paper is organized as follows: Section II describes our technique in detail. Section III demonstrates our empirical evaluation. Section IV and Section V discuss related work and present our conclusions.

## II. OUR TECHNIQUE

In this section, we present our technique of checking vulnerability rules against source files. Our technique can handle application programs written in most of mainstream programming languages (e.g. Java, C/C++, C#, etc). In this section, we take Java as an example to demonstrate its working principle. We first provide an intuitive description of our technique and then discuss its main characteristics in detail.

### A. General Approach

Our technique is based on the observation that a vulnerability rule is dedicated to some particular objects. Take the rule `MDBAndSessionBeanNamingConvention` illustrated in Figure 2 for example, it is designed to check the EJB specification that any class implementing `MessageDrivenBean` or `SessionBean` interface should be suffixed by “Bean”. As we can see, the rule aims to detect violations concerning class `SessionBean` and `MessageDrivenBean`. We can conclude that, a source file must be free of vulnerabilities of the rule, if neither of the classes has been used in the file. In other words, the classes form a prerequisite for a source file to contain vulnerabilities of rule `MDBAndSessionBeanNamingConvention`. On the other hand, in Java programs, to use a class, we always import it first at the beginning of the file using import statements. Consequently, we can judge whether a vulnerability rule should be employed to check a source file through inspecting import statements at the beginning of the file.

We call all the classes that a rule relies on *Characteristic Objects* and describe them using a designed prefix notation [25] *CObject Expression*. Based on characteristic objects, we can filter out rules whose characteristic objects have not been imported before checking a source file, which can reduce time overhead enormously. Details of our technique are elaborated in later sections.

### B. CObject Expression

It must be noted that characteristic objects `MessageDrivenBean` and `SessionBean` of the rule `MDBAndSessionBeanNamingConvention` shown in Figure 2 have a *disjunction* relationship. In order to describe characteristic objects as well as relationships among them, we propose a prefix notation *CObject Expression*.

A CObject expression is comprised of full package names of characteristic objects (full package names facilitate evaluation of the expressions), commas, parentheses and operator symbols. Recursive definition of CObject expression is presented as follows. Given full package name  $\lambda$  of a characteristic object and two CObject expressions  $R$  and  $S$ , we have the following CObject expressions:

- $exist(\lambda)$  denotes *existence* operation that determines whether the characteristic object  $\lambda$  has been imported into a source file.
- $neg(R)$  denotes *negation* operation that reverses the meaning of a Boolean operand.
- $and(R, S)$  denotes *conjunction* operation that performs a logical conjunction on two Boolean operands.
- $or(R, S)$  denotes *disjunction* operation that performs a logical disjunction on two Boolean operands.

CObject expression is compact and easy to use. In addition, the characteristic that operators can be nested within each other makes the notation powerful to express complicated logical relationships. For instance, the characteristic objects of the rule shown in Figure 2 can be described using a CObject expression as follows:  $or(exist(javax.ejb.SessionBean), exist(javax.ejb.MessageDrivenBean))$ , which means that a violation may exist in a source file, if either of the classes `SessionBean` and `MessageDrivenBean` has been imported into the file. To check the expressing capability of CObject expression, we inspected all the rules of PMD and no exceptions were found. The expression may be limited for some unknown rules (e.g. rules whose characteristic objects have *if-then* relationship), but we believe it is applicable in most of cases.

### C. Filtering Rules by Evaluating CObject Expression

Before checking vulnerability rules against a source file  $f$ , our technique first filters the rules according to evaluation results of their CObject expressions. For each rule  $r$ , if the evaluation result is Boolean *true*, we check  $r$  against  $f$ . Otherwise, we discard the rule. In this subsection, we discuss our approach of evaluating CObject expressions.

Since CObject expression is a kind of prefix notation, it can

TABLE I. EXAMPLE OF EVALUATING A CObject EXPRESSION.

Step	Token ( $\eta$ )	Stack ( $T$ )	Action
1	)		Discard
2	)		Discard
3	$O_2$	$O_2$	Push $O_2$ into $T$
4	(	$O_2$	Discard
5	$exist$	$F$	Pop $O_2$ , Evaluate $exist(O_2)$ with $\rho$ , push result $F$ into $T$
6	,	$F$	Discard
7	)	$F$	Discard
8	$O_1$	$O_1, F$	Push $O_1$ into $T$
9	(	$O_1, F$	Discard
10	$exist$	$F, F$	Pop $O_1$ , Evaluate $exist(O_1)$ with $\rho$ , push result $F$ into $T$
11	(	$F, F$	Discard
12	$or$	$F$	Pop two elements $F$ and $F$ , calculate $F \vee F$ , Push result $F$ into $T$
13			Pop and output result $F$

be evaluated using the traditional stack-based approach [25]. In detail, given a CObject expression  $e$ , we scan  $e$  from right to left. For each encountered token  $\eta$ , we perform the following actions according to its type:

- If  $\eta$  is an *operand*, we push  $\eta$  into a stack  $T$ .
- If  $\eta$  is an *operator*, we perform corresponding operations on operands popped from the top of  $T$  and push the results back into  $T$ .

Once all the symbols have been processed, a Boolean value will be left in  $T$ , which is the evaluation result of  $e$ .

It must be noted that, CObject expression subsumes four kinds of operators: *neg*, *and*, *or* and *exist*. The former three are logical operators. As to the last one, we carry out the operation using the *linear search algorithm*. Given a full package name  $\lambda$  of a characteristic object,  $I$  is a set of import statements at the beginning of a source file  $f$ . To distinguish whether  $\lambda$  has been imported into  $f$  (evaluating  $exist(\lambda)$  with  $f$ ), we search  $\lambda$  in  $I$  sequentially. The linear search algorithm is desirable here, because it is easy to implement and the limited size of  $I$  will not cause too much overhead. What should be noted is that two kinds of import statements exist in Java to import an object into a file. The first type like `import javax.ejb.SessionBean` is used to import exactly one object and the second type such as `import javax.ejb.*` is used to import all objects within a package. Consequently, for each characteristic object  $\lambda$ , we should lookup  $\lambda$  itself and  $\lambda^*$  which denotes the second type of import statements of  $\lambda$ .

As an example, we evaluate the CObject expression of rule `MDBAndSessionBeanNamingConvention` shown in Figure 2 with an input program  $\rho$  illustrated in Figure 3. The evaluation process is presented in Table I. For concision, we denote `class javax.ejb.SessionBean` and `javax.ejb.MessageDrivenBean` as  $O_1$  and  $O_2$  respectively. The Boolean value *True* and *False* are denoted by  $T$  and  $F$ . Based on the notations, the CObject expression can be rewritten in a more compact form as  $or(exist(O_1), exist(O_2))$ . From Table I, we find that the evaluation result  $F$  is consistent with our manual inspection. Consequently, it is unnecessary to check rule

```

(1) package pmd_test;
(2)
(3) import java.util.Collection;
(4) import java.util.ArrayList;
(5)
(6) public class Pmd_Test {
(7)     public static void main(String[] args) {
(8)         Collection c = new ArrayList();
(9)         Integer obj = new Integer(1);
(10)        c.add(obj);
(11)
(12)        Integer[] a = (Integer [])c.toArray();
(13)    }
(14) }

```

Figure 3. A program example

### III. EVALUATION

To evaluate our technique, we implemented it in a prototype tool called EPMD, which developed from the open source static analysis tool PMD, and investigated the performance improvement of our technique. The remainder of this section discusses prototype tool EPMD, our subjects, results of experiments, our technique’s runtime overhead and limitations.

#### A. Prototype Tool EPMD

We implemented our technique in an open source static analysis tool PMD and called the extended version EPMD. The primary modifications that we did to PMD are as follows:

- Implementation of a module that evaluates CObject expressions.
- Addition of CObject expressions to the vulnerability rules of PMD.

In order to evaluate CObject expressions, we constructed a lexer and parser with the help of ANTLR [19-22] to recognize tokens and validate syntax of CObject expressions respectively. After that, we evaluate CObject expressions using the stack-based approach elaborated in Section II with input of tokens recognized by the lexer.

#### B. Subjects

We adopt six large-scale open source applications in our evaluation, which are listed in Table II. The selection of these applications is based on the following criteria: (1) applications with available source code; (2) applications of a large code base; (3) applications coming from various application domains. The second requirement is important for our evaluation, because the performance improvement gained by the removal of one unnecessary rule is trivial. In order to achieve accurate and salient results, we should perform experiments on large-

TABLE II. APPLICATIONS USED IN EVALUATION. THE SIZE OF APPLICATIONS COUNTS CODE LINES ONLY, COMMENT LINES AND BLANK LINES ARE ALL EXCLUDED. KLOC: THOUSANDS OF LINES OF CODE.

Subject	Description	# Source File	KLoC
Eclipse 4.2	Software development environment	20602	2675
ElasticSearch v0.19.9	Distributed search engine	2581	230
PMD 5.0	Source code analyzer	945	73
Tomcat 6.0.35	Web server and servlet container	1157	172
SQuirreL SQL Client 3.4.0	Java SQL client	2890	253
JBoss 6.0.0.Final	Application server	6397	494

scale applications, in which case, a large number of unnecessary rules may be pruned away. Furthermore, to avoid biases, our subjects originate from various application domains and are of different size spanning from 73 thousands of lines of code (KLoC) to 2675 KLoC.

#### C. Performance Evaluation with EPMD

To investigate effectiveness of our technique, we analyzed subject applications using EPMD with rule sets  $RS$  and  $RS^*$  respectively. Rule set  $RS$  consists of 45 rules excerpted from PMD, and  $RS^*$  contains all the same elements as  $RS$  except that, each rule of which has an additional property of CObject expression. In accordance with our technique, rules will be filtered when running EPMD with rule set  $RS^*$  but not with  $RS$ . To achieve accurate results, we repeatedly ran EPMD five times for each group of subject and rule set. After that, we compared the average execution time of EPMD running with rule set  $RS$  and  $RS^*$ . Our experiment was conducted on an Intel(R) Core (TM) i3-2100 3.1GHz machine with 3GB of memory running Windows XP. The experimental results are presented in Table III.

From Table III, we find that the execution time with rule set  $RS^*$  is generally less than that of  $RS$  for all applications. The least ratio of reduction is 15.6% when testing on Eclipse. The

TABLE III. TIME OVERHEAD OF EPMD RUNNING WITH RULE SET  $RS$  AND  $RS^*$  RESPECTIVELY. AVG: AVERAGE TIME OVERHEAD; REDUCTION: AVERAGE TIME OVERHEAD REDUCTION.

		1-st Time (s)	2-nd Time (s)	3-rd Time (s)	4-th Time (s)	5-th Time (s)	Avg (s)	Reduction (s)	% Reduction
Eclipse	$RS$	469.8	437.4	444.0	438.9	467.1	451.4	70.4	15.6
	$RS^*$	380.3	380.2	380.3	382.9	381.6	381.1		
ElasticSearch	$RS$	40.7	36.9	37.1	36.7	39.7	38.2	6.6	17.3
	$RS^*$	31.5	31.7	31.7	31.6	31.5	31.6		
PMD	$RS$	9.5	6.6	6.7	6.6	8.6	7.6	3.4	45.3
	$RS^*$	4.1	4.2	4.1	4.2	4.2	4.2		
Tomcat	$RS$	15.5	14.8	14.6	14.6	15.2	14.9	5.4	36.4
	$RS^*$	9.6	9.5	9.3	9.6	9.5	9.5		
SQuirreL SQL Client	$RS$	45.2	31.2	31.4	48.1	31.1	37.4	11.1	29.6
	$RS^*$	26.4	26.5	26.2	25.8	26.8	26.3		
Jboss	$RS$	57.8	49.0	49.2	66.3	49.6	54.4	15.4	28.2
	$RS^*$	38.8	39.2	39.0	39.0	39.1	39.0		

---

```

<![CDATA[
//ForStatement
[count(*) > 1]
[not(ForInit)]
[not(ForUpdate)]
[not(Type and Expression and Statement)]
]]>

```

---

Figure 4. Rule `ForLoopShouldBeWhileLoop` excerpted from PMD

ultimate is 45.3% with PMD. We have an average ratio of time reduction 28.7%. A side effect that may be caused by our technique is the increase of false positives and false negatives. We compared the bug reports generated in each group and found that the bugs reported by EPMD running with rule set *RS* and *RS\** were the same (in terms of the number and content).

The above results, albeit preliminary in nature, strongly suggest that our technique is effective in improving static analysis efficiency, because (1) the subjects that we used are real-world programs originated from various application domains and (2) the rules utilized are also realistic. Moreover, we even believe better results can be achieved when testing on some special subjects or with more vulnerability rules. We will investigate this in the future work.

#### D. Runtime Overhead

The primary runtime overhead of our technique is introduced in the phase of filtering rules (which we have discussed in Section II). In order to filter rules, we evaluated CObject expressions based on the approach applied to prefix notations. Different actions were carried out according to the type of operators, among which, the existence operation is crucial. Let  $L$  and  $M$  be the count of tokens of a CObject expression and import statements of a source file respectively, the time complexity of the stack-based evaluation approach is  $O(L)$ . The existence operation based on the linear search algorithm has a time complexity of  $O(M)$ , because it mainly consists of a loop iterating through all import statements. Therefore, the time complexity of filtering rules is  $O(L \times M)$ . On the other hand, it is quite reasonable to assume  $L$  to be constant, because, in general, the CObject expressions of rules are in limited length. Additionally, it is well known that the number of import statements is trivial relative to lines of code in a source file. Consequently, we believe the runtime overhead incurred by EPMD is within an acceptable range.

#### E. Limitations

Although our technique is effective in improving performance of static analysis, it has limitations. The limitations are mainly caused by the prerequisite that characteristic objects can be distinguished through import statements. The problem is that the assumption cannot be satisfied in some circumstances and the following cases may threaten the validity of our technique.

(1) In Java programs, we can utilize an object via its fully qualified name without corresponding import statements at the beginning of source files. In this case, we cannot determine the

---

```

(1) public class Foo {
(2)     void bar() {
(3)         for (;true;) true;
(4)     }
(5) }

```

---

Figure 5. For loop example which should be simplified to a while loop

use of characteristic objects by inspecting import statements and our technique does not work.

(2) There is one exception to the import rule of Java, that is, all objects in `java.lang` package are imported by default. Thus, any object in `java.lang` package can be used freely without importing explicitly. This case limits the scope that our technique can be used.

(3) It is allowable to import a package without use of any objects belonging to it. Moreover, we cannot ascertain the existence of a specific object according to import statements because a package subsumes multiple objects. What we can determine is the nonexistence of objects in terms of absence of corresponding import packages. This circumstance introduces inaccuracy to our approach.

(4) There exist some rules that have no characteristic objects. Consider the rule `ForLoopShouldBeWhileLoop` illustrated in Figure 4, it is used to detect *for loops* which can be simplified to *while loops*. For instance, the *for loop* (line 3) showed in Figure 5 matches the above rule and it can be simplified to a while loop as *while(true)*. This rule does not possess any characteristic objects but for a keyword *for* which cannot be distinguished through import statements. Therefore our technique is unable to handle this kind of rules.

Though our technique has some limitations above, it is applicable in most of circumstances except a small part of rules discussed in limitations (2) and (4). As to limitations (1) and (3), the cases are rare in applications with good programming practices. Some analysis tools even regard redundant import statements as errors. In terms of the evaluation results, we believe that our technique can work well in practice.

## IV. RELATED WORK

Though some state-of-the-art static analysis tools are efficient enough to satisfy most of applications, performance is an eternal topic and it is still a key challenge of integrating bug-finding tools into development process. To resolve the performance issue, researchers have proposed various kinds of techniques. In this section, due to space considerations, we limit our discussion to several typical techniques.

Incremental analysis [4] can automatically infer what parts of source code have to be reanalyzed after the code has been modified. This approach typically reduces the analysis time substantially, especially for a frequent interactive use of static analysis tools in development process. However, it is difficult to accurately identify the associated code when programs have been changed and in the worst case, a complete reanalysis will be performed.

Nowadays, a widely used approach to improve performance of static analysis tools is to perform less deep analysis, such as

GREP, which performs lexical analysis only [1, 4]. However, these tools have too many false positives, which discount their usability.

Many static analysis tools, such as FindBugs [24] and PMD [23] provide the functionality for users to select vulnerability rules before checking a program. This approach is most similar to our technique. Both them aim to reduce overhead by filtering rules. However, our technique can filter rules automatically.

## V. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a novel rule-filtering algorithm based on characteristic objects. Additionally, in order to describe characteristic objects, we designed a prefix notation COBject expression. By evaluating COBject expressions using the stack-based approach, our technique can determine whether a rule should be checked against a specific source file or not.

We also presented EPMD, a prototype tool developed from PMD that implements our technique. In the evaluation, we performed five groups of test using EPMD on subjects of six real-world applications. The experimental results strongly suggest that our technique is effective in improving static analysis performance.

Though we discussed our technique based on Java language, as a matter of fact, our technique is applicable for programs written in most of mainstream programming languages, such as C/C++ and C#. What is noteworthy is that, not all languages have *import statements* as Java. It is *header file* instead in C/C++. On the other hand, the effectiveness of our technique may be varied for different languages. For instance, the limitations (1) and (2) discussed in Section III will not exist in C/C++ programs, which makes our technique become more useful. Applications of our technique to other programming languages and a further investigation are left to the future work.

## REFERENCES

- [1] D.C. Atkinson and W.G. Griswold, Effective pattern matching of source code using abstract syntax patterns. *Software-Practice & Experience*, 2006. 36(4): p. 413-447.
- [2] N. Rutar, C.B. Almazan and J.S. Foster, A comparison of bug finding tools for Java, in *Proceedings of the 15th International Symposium on Software Reliability Engineering*. 2004. p. 245-256.
- [3] D. Hovemeyer and W. Pugh, Finding bugs is easy, OOPSLA'04 Conference, published as ACM SIGPLAN Notices. 2004: Vancouver, British Columbia, Canada. p. 92-106.
- [4] P. Emanuelsson and U. Nilsson, A comparative study of industrial static analysis tools. *Electron. Notes Theor. Comput. Sci.*, 2008. 217: p. 5-21.
- [5] J.E.M. Araujo, S. Souza and M.T. Valente, Study on the relevance of the warnings reported by Java bug-finding tools. *IET Software*, 2011. 5(4): p. 366-374.
- [6] N. Ayewah, W. Pugh, J. D. Morgenthaler, J. Penix and Y. Q. Zhou, Evaluating static analysis defect warnings on production software, in *Paste'07 Proceedings of the 2007 ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools & Engineering*. 2007. p. 1-7.
- [7] D. Hovemeyer and W. Pugh, Finding more null pointer bugs, but not too many, in *Paste'07 Proceedings of the 2007 ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools & Engineering*. 2007. p. 9-14.
- [8] R. Ploesch, H. Gruber, A. Hentschel, G. Pomberger and S. Schiffer, On the relation between external software quality and static code analysis, in *Proceedings of the 32nd Annual IEEE Software Engineering Workshop*. 2009. p. 169-174.
- [9] M.T. Helmick, Interface-based programming assignments and auto-matic grading of Java programs, in *ITICSE 2007: 12th Annual Conference on Innovation & Technology in Computer Science Education*. 2007. p. 63-67.
- [10] S. Loveland, Using open source tools to prevent write-only code, in *Proceedings of the 2009 Sixth International Conference on Information Technology: New Generations, VOLS 1-3*. 2009. p. 671-677.
- [11] M. Martin, B. Livshits and M.S. Lam, Finding application errors and security flaws using PQL: a program query language. *ACM SIGPLAN Notices*, 2005. 40(10): p. 365-383.
- [12] S. Jarzabek, Design of flexible static program analyzers with PQL. *IEEE Transactions on Software Engineering*, 1998. 24(3): p. 197-215.
- [13] E. Hajiyev, M. Verbaere and O. de Moor, CodeQuest: scalable source code queries with datalog, in *ECOOP'06 Proceedings of the 20th European conference on Object-Oriented Programming*. 2006. p. 2-27.
- [14] M. Alpuente, M. A. Feliu, C. Joubert and A. Villanueva, Using datalog and boolean equation systems for program analysis, in *Formal Methods for Industrial Critical Systems*. 2009. p. 215-231.
- [15] D. Zook, E. Pasalic and B. Sarna-Starosta, Typed datalog, in *Practical Aspects of Declarative Languages*. 2009. p. 168-182.
- [16] J. Whaley, D. Avots, M. Carbin and M. S. Lam, Using datalog with binary decision diagrams for program analysis, in *Proceedings of Programming Languages and Systems*. 2005. p. 97-118.
- [17] O. Panchenko, A. Treffer and A. Zeier. Towards query formulation and visualization of structural search results. in *Proceedings of 2010 ICSE Workshop on Search-driven Development: Users, Infrastructure, Tools and Evaluation*. 2010. Cape Town, South Africa: ACM.
- [18] O. Panchenko, J. Karstens, H. Plattner and A. Zeier, Precise and scalable querying of syntactical source code patterns using sample code snippets and a database, in *2011 IEEE 19th International Conference on Program Comprehension (ICPC)*. 2011. p. 41-50.
- [19] T. Parr and K. Fisher, LL(\*): The foundation of the ANTLR parser generator. *ACM SIGPLAN Notices*, 2011. 46(6): p. 425-436.
- [20] J. Bovet and T. Parr, ANTLRWorks: an ANTLR grammar development environment. *Software-Practice & Experience*, 2008. 38(12): p. 1305-1332.
- [21] S. Liu, R. Zhang, D. Wang, H. Sun, Y. Chen and L. Li, Implementing of Gaussian syntax-analyzer using ANTLR, in *Proceedings of the 2008 International Conference on Cyberworlds*. 2008. p. 613-618.
- [22] G.L. Schaps, Compiler construction with ANTLR and Java-Tools for building tools. *Dr Dobbs Journal*, 1999. 24(3): p. 84-+.
- [23] PMD, 2014. <http://pmd.sourceforge.net>.
- [24] FindBugs, 2014. <http://findbugs.sourceforge.net>.
- [25] Polish notation, 2014. [http://en.wikipedia.org/wiki/Polish\\_notation](http://en.wikipedia.org/wiki/Polish_notation).

# Performance and Usability Evaluation of a Pattern-Oriented Parallel Programming Interface for Multi-Core Architectures

Dalvan Griebler, Daniel Adornes, Luiz Gustavo Fernandes  
Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS),  
School of Informatics (FACIN), Graduate Program in Computer Science (PPGCC),  
Parallel Application Modeling Group (GMAP).

Av. Ipiranga, 6681 - Building 32 - Porto Alegre - Brazil

`dalvan.griebler@acad.pucrs.br`, `daniel.adornes@acad.pucrs.br`, `luiz.fernandes@pucrs.br`

**Abstract**—Multi-core architectures have increased the power of parallelism by coupling many cores in a single chip. This becomes even more complex for developers to exploit the available parallelism in order to provide high performance scalable programs. To address these challenges, we propose the DSL-POPP (Domain-Specific Language for Pattern-Oriented Parallel Programming), which links the pattern-based approach in the programming interface as an alternative to reduce the effort of parallel software development, and achieve good performance in some applications. In this paper, the objective is to evaluate the usability and performance of the master/slave pattern and compare it to the Pthreads library. Moreover, experiments have shown that the master/slave interface of the DSL-POPP reduces up to 50% of the programming effort, without significantly affecting the performance.

*Keywords:* Parallel Programming, Pattern-Oriented, Performance Evaluation, Usability Evaluation.

## I. INTRODUCTION

Parallel programming has become inserted in the daily activities of software developers due to easy access to the multi-core architectures. However, its higher computational power achieved by coupling multiple processing elements on a single chip, increases complexity in parallelism exploitation. Thus requiring higher programming efforts to develop programs with good scalability and high performance.

Initially, the main interest of software designers was to provide flexible libraries. In recent years, the concern has also been to create high-level abstractions including flexibility issues without compromising the performance of the application. Therefore, the challenge is to evaluate the programming interface's usability. This is not a simple task because it requires volunteers to perform the experiments, and becomes laborious for everyone involved [16].

Such challenges have already been targeted in many studies that propose some level of abstraction [17]. On the other hand, [14] proposed an environment to automatically evaluate the usability of parallel language constructions, which does not consider the programming effort. In this paper, we present the master/slave pattern-oriented interface of DSL-POPP for

multi-core architectures in order to evaluate its performance and usability. The contributions are the following:

- We present the master/slave pattern interface of the DSL-POPP;
- We perform a usability experiment comparing the programming effort between DSL-POPP and Pthreads;
- We conduct a performance experiment using four different applications.

This paper is organized as follows: Section II discusses related work; Section III presents the DSL-POPP in a nutshell; Section IV demonstrates the usability experiment; Section V describes the performance experiment; and finally, Section VI presents the conclusions.

## II. RELATED WORK

We consider Delite the first project to face the challenge of using DSLs as a programming environment for simplifying parallel programming. It generates parallel embedded DSLs for an application, whereas programmers implement domain-specific operations by extending the DSL framework [4]. The final result is a high performance DSL for a wider diversity of parallel architectures. In its interface implementation, Delite uses the Scala language. However, one disadvantage is that the lack of some features in the host language may restrict performance optimizations.

The concept of the algorithm skeleton was introduced by Murray Cole [6], who later proposed a skeleton-based programming environment called eSkel. Many libraries and frameworks emerged [8], allowing skeleton constructions in message passing and thread model scenarios. Other related approaches have emerged like the CO2P3S, which is a pattern-based parallel programming framework that creates pre-defined templates through a high-level interface [3], and the TBB (Thread Building Blocks) library that allows skeleton implementation in low-level operations [10]. Recently, the FastFlow template library [1] introduced a precise and optimized implementation of patterns for streaming applications and has proven to be faster than TBB, due to its advanced

parallelism implementation through techniques such as lock-free synchronization [2].

Yet, there are also programming models that are not pattern-based such as OpenMP [5], which have a set of pragma primitives designated to exploit loop parallelism in sequential programs. Charm++ is a parallel language based on the C++ syntax, parallelizing the execution through parallel objects and channel mechanisms [12]. Lastly, Cilk++ is a C language extension that provides keywords for spawn and synchronize threads, and loop parallelism [13]. The drawback of these specialized systems is that pattern-based implementations may require substantial work.

This research differs from these previous related works in it focuses on usability through a pattern-oriented interface, although it continues to focus on high performance and scalable programming. Also, this paper will evaluate the performance of parallel applications in multi-core architectures using statistical method, which differs of studies for performance prediction of parallel patterns [15].

### III. DSL-POPP IN A NUTSHELL

The POPP model is a standard way to design algorithms capable of combining and nesting parallel patterns while remaining compliant with different high performance architectures. It is framed on top of pattern-based approaches, where these patterns are presented as routine, and the skeleton templates are code blocks. The nesting of patterns occurs when a pattern subroutine is called inside a code block, producing a hierarchical composition. Additionally, it is possible to have different pattern combinations, resulting in different parallelism behaviors [9]. However, in this paper we focus on the master/slave, demonstrating how it can be nested in order to achieve levels of parallelism. Therefore, we exemplify in Figure 1(a) its behavior using process illustrations to represent abstract parallelism.

The master is responsible for sending the computational tasks to all slaves. Then, once all tasks have been computed, results are sent back to the master to finalize the whole computation. In this pattern, both the POPP routine and the subroutines can implement their own master and slaves code blocks. For instance, a slave block may have as many slave processes as necessary running parallel, and nesting occurs when a slave or master code block calls a Master/Slave subroutine.

We named active processes those that are used to measure the performance, and control processes those that are only used to represent the skeleton behavior pattern. Through the processes labeled second level, we see the point from which levels of parallelism are achieved. In general, the control process coordinates the computation flow and does not perform significant computations (being CPU idle) while active processes are computing their tasks. For this reason, those processes are not used to measure the speed-up. The levels of parallelism can be achieved by calling subroutines from inside the slave block. Even though the execution sequence

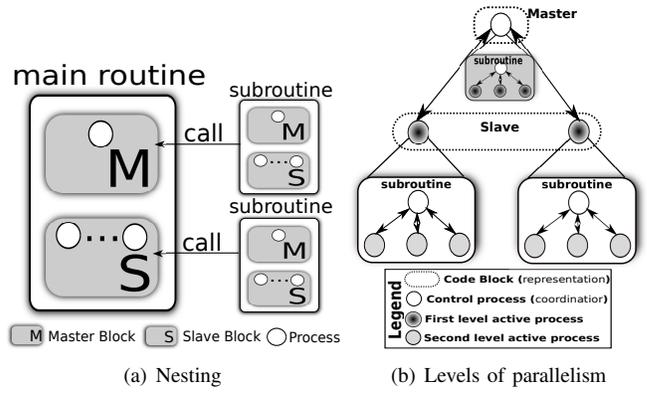


Fig. 1. How to address nested patterns and levels of parallelism through the POPP model for the master/slave pattern.

of the routines is not clearly defined in the example, the idea is that each one executes after the other has finished. In other words, using the example in Figure 1(b), slave block processes do not run parallel with the subroutine called on the master block, but they are on the same parallelism level.

The POPP model is built over the C language and standardized as follows: the specification of routines begins with “\$” and code blocks with “@”. The pattern routine should be declared in a function followed by the return data type and its name. Code blocks should be used inside of the pattern routine, consisting of pure C code, and the skeleton communication occurs through predefined arguments. Also, the code declared inside each block is private and will not be accessible to other code blocks. In the current master/slave interface, in the master block only the buffer and its size have to be given, whereas in the slave block it is necessary to specify the number of threads, buffer, buffer size, and the load balancing policy (Listing 1).

```

1 $MasterSlavePattern int func_name() {
2   @Master(void *buffer, const int size) {
3     // full C code
4     @Slave(const int num_threads, void *
5           buffer, const int size, const policy) {
6       // full C code
7     }
8   }
9 }

```

Listing 1. Master/Slave programming interface.

In the runtime system, the master block creates as many threads as defined in the slave block and waits until all slave threads have finished their work. This is hidden from programmers, since thread creation occurs when the slave block starts and the synchronization is performed automatically at the end of the slave block. Moreover, at the end of a slave block, all slave threads send their work back to the master (using the buffer parametrized to the slave block) in order to allow it to merge all results. This communication procedure, and the load balancing are also hidden from developers.

In fact, slave threads receive their workloads according to the policy argument in the slave block. The implementation of

these policies is automatically generated by the pre-compiler system. Currently, we have only implemented an optimized version of the static load balancing (POPP\_STATIC), where the workload is uniformly divided by the number of threads, and the resulting chunks are then statically assigned to slave threads as they start their computations.

To use the programming interface, developers have to include the DSL-POPP library (*poppLinux.h*) in the source code and compile the program with the DSL-POPP compiler (*popp*). This library includes all routine definitions and code blocks. The compilation process of the source code is depicted in Figure 2. The source-to-source code transformation between DSL-POPP interface and C code is automatically done by the pre-compiler, which is also responsible for checking syntax and semantic errors. Then, the pre-compiler generates the C parallel code using the Pthreads library based on the parallel pattern used. Finally, we use the GNU C compiler to generate binary code for the target platform.

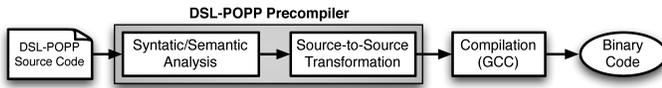


Fig. 2. Compiler overview.

### A. Matrix Multiplication Example

DSL-POPP can be used to design different parallel algorithms [9]. However, we chose the classic matrix multiplication algorithm to illustrate how programmers have to parallelize it to perform the usability experiment. This application consists of some auxiliary functions to load and print the matrix as well as measure the execution time. We intend to present in details only the main parts of the application, since the focus relies on Matrix Multiplication (MM), which has to be implemented with Pthreads (Listing 2) and DSL-POPP (Listing 3).

```

1#include <stdio.h>
2#include <pthread.h>
3#define MX 1000 //matrix size
4long int num_threads=2;
5long int matrix1 [MX][MX], matrix2 [MX][MX],
  matrix [MX][MX];
6double timer() /*return the time in seconds*/
  }
7void attr_val(long int **matrix, long int **
  matrix1, long int **matrix2 /* values
  attribution */)
8void printMatrix(long int **matrix) /* prints
  a matrix */
9void *Thread(void *th_id){
10 long int id= (long int*) th_id;
11 long int i, j, k, end;
12 end=(id*(MX/num_threads))+(MX/num_threads);
13 if(id==num_threads-1)
14   end=MX;
15 for(i=id*(MX/num_threads); i<end; i++)
16   for(j=0; j<MX; j++)
17     for(k=0; k<MX; k++)
18       matrix[i][j] += (matrix1[i][k] *
  matrix2[k][j]);
  
```

```

19 pthread_exit(NULL);
20}
21int main(){
22 double t_start, t_end;
23 pthread_t th[num_threads];
24 void *status;
25 t_start = timer();
26 attr_val(matrix, matrix1, matrix2);
27 int i;
28 for(i=0; i<num_threads; i++)
29   pthread_create(&th[i], NULL, &Thread, (
  void *) i);
30 for(i=0; i<num_threads; i++)
31   pthread_join(&th[i], &status);
32 t_end = timer();
33 printf("EXECUTION TIME: %lf seconds\n",
  t_end-t_start);
34 printMatrix(matrix);
35}
  
```

Listing 2. MM implemented with Pthreads .

As can be observed, we used a very simple strategy to parallelize the MM with Pthreads. The *main* function is responsible for creating auxiliary threads (lines 28 - 31), which will perform the MM in parallel (lines 15 - 18). Therefore, each thread performs the MM on a sub-matrix. The sub-matrix is obtained by dividing the number of rows of the resulting matrix by the number of threads. We used the thread ID to select a different sub-matrix for each thread, avoiding the use of synchronization mechanisms (lines 12 - 14).

```

1#include <stdio.h>
2#include "poppLinux.h"
3#define MX 1000 //matrix size
4long int num_threads=2;
5long int matrix1 [MX][MX], matrix2 [MX][MX],
  matrix [MX][MX];
6double timer() /*return the time in seconds*/
  }
7void attr_val(long int **matrix, long int **
  matrix1, long int **matrix2 /* values
  attribution */)
8void printMatrix(long int **matrix) /* prints
  a matrix */
9$MasterSlavePattern int main(){
10 @Master(matrix, MX){
11   double t_start, t_end;
12   t_start = timer();
13   attr_val(matrix, matrix1, matrix2);
14   @Slave(num_threads, matrix, MX,
  POPP_STATIC){
15     long int i, j, k;
16     for(i=0; i<MX; i++)
17       for(j=0; j<MX; j++)
18         for(k=0; k<MX; k++)
19           matrix[i][j] += (matrix1[i][k]*
  matrix2[k][j]);
20   }
21 t_end = timer();
22 printf("EXECUTION TIME: %lf seconds\n",
  t_end-t_start);
23 printMatrix(matrix);
24 }
25}
  
```

Listing 3. MM implemented with DSL-POPP .

The corresponding DSL-POPP version of the MM is presented in Listing 3. The main routine is composed of a master block (line 9) and a slave block (line 14). The master block does not contribute to the whole parallel computation, since it starts the computation, joins the results of each slave threads, and creates the slaves and waits for them to finish so that it is automatically performed by the pre-compiler. The slave block contains the sequential code of the MM which will be split among independent slave workers (threads).

It is possible to notice that DSL-POPP considerably reduces the amount of code necessary to parallelize the application. Additionally, developers do not have to worry about how to split the work. To evaluate how this will impact the programming effort, we performed a usability experiment using this application in the next section.

#### IV. USABILITY EVALUATION

In this experiment, we only considered the time spent (in minutes) to implement the parallel solution, using it as the metric to evaluate the programming effort when using DSLPOPP compared to Pthreads. In order to do so, we invited M.Sc. and Ph.D. students in Computer Science and asked them to parallelize the matrix multiplication with these approaches. Then, we performed an analysis using the hypothesis statistical test, making the assumption of a null hypothesis ( $H_0$ ) that will be rejected or not. We used 95% reliability and the significance level (max probability to reject the  $H_0$ ) adopted was 5%. This means that to reject the  $H_0$  result has to be less than 0.05 (this value is called  $p$ -value in the literature) [7]. The hypotheses are formalized as follows:

- 1) **Null hypothesis ( $H_0$ ):** the effort in parallel programming with Pthreads is equal to the effort using DSL-POPP (Pthreads = DSL-POPP).
- 2) **Alternative hypothesis ( $H_1$ ):** the effort in parallel programming with Pthreads is greater than DSL-POPP approach (Pthreads > DSL-POPP).

We invited students to answer a questionnaire to evaluate their previous knowledge (the options were: none, low, medium, and high). Based on this questionnaire, we removed the students that did not have the required skills (knowledge in C language, parallel programming and Linux platform). Thus, we split the remaining 20 participants into two groups with equal knowledge. Group 1 parallelized the problem with the DSL-POPP first and then with Pthreads. Group 2 parallelized the same problem but in the reverse order.

In order to avoid external influences, the experiment was carried out in a controlled environment (laboratory), where participants only had access to a user manual (previously reviewed by another researcher) and the original sequential code. Each participant used a desktop machine with Ubuntu Linux installed and no Internet access was allowed. All students had to achieve correct parallel implementations with performance above a minimum threshold (of at least 90% of the linear speedup with two threads) for the experiment to be considered completed.

Table I presents the results obtained from the effort evaluation. For each participant, we present his/her previous experience with Pthreads (taken from the questionnaire) and the time spent to implement the problem using the DSL-POPP and Pthreads. Therefore, it is important to highlight that all participants had never developed applications with the DSL-POPP and most of them had already developed parallel applications with Pthreads. The results demonstrate that the DSL-POPP demands less work from the developers. An exception was found in Group 1 (ID=1), which parallelized the problem faster with Pthreads than with DSL-POPP. This is due to two factors: (i) the participant had already implemented a matrix multiplication with Pthreads and (ii) he had significant experience in developing applications with Pthreads.

We used the Statistical Package for the Social Sciences (SPSS) to analyze the obtained results. The analysis showed a significantly higher level of effort to program with Pthreads than with DSL-POPP. The average time spent to implement the parallel version of the matrix multiplication was 51.45 minutes with Pthreads (the 95% confidence interval was 42.98 to 59.92 minutes) whereas it was 20.70 minutes for the average with DSL-POPP (the 95% confidence interval was 17.59 to 23.81).

For the hypothesis test, there are basically two statistical tests that can be applied: parametric and nonparametric. The parametric test normally requires distributed data and homogeneity of variance. However, the time spent to implement the solution with Pthreads did not follow a normal distribution, thus indicating a non-parametric test. Due to that, we used the Wilcoxon approach for paired sample tests [7]. This hypothesis test is based on the differences between the scores of the two approaches, ranking them positively and negatively. The results are shown in Table II.

TABLE II  
STATISTICAL HYPOTHESIS TEST.

Ranks	N	Mean Rank	Sum of Ranks
Negative Ranks	1 (a)	4.0	4.0
Positive Ranks	19 (b)	10.8	206.0
Ties	0 (c)	-	-
<b>Total</b>	<b>20</b>		

(a) Pthreads  $\dot{}$  DSL\_POPP

(b) Pthreads  $\dot{}$  DSL\_POPP

(c) Pthreads = DSL\_POPP

Wilcoxon test	Pthreads vs. DSL-POPP
Z	-3.771*
Asymp. Sig. (2-tailed)	0.0

\*Based on negative ranks.

We noticed that only one negative rank was found in the statistical test. This means that Pthreads require more effort than DSL-POPP ((b) Pthreads > DSL\_POPP). In order to confirm whether the effort is significantly different between these approaches, we used the significance level (Sig.), which must be less than 0.05 [7]. The SPSS returned the result shown at the bottom (Wilcoxon test) of Table II, concluding that the effort is significantly different. Thus, we can reject the null hypothesis ( $H_0$ ) and based on the mean results, we can accept the alternative hypothesis  $H_1$ , which states that the effort in

TABLE I  
EFFORT EVALUATION RESULTS.

Group 1: DSL-POPP → Pthreads				Group 2: Pthreads → DSL-POPP			
ID	Experience with Pthreads	DSL-POPP Time (min)	Pthreads Time (min)	ID	Experience with Pthreads	DSL-POPP Time (min)	Pthreads Time (min)
1	Medium	29	18	11	High	15	33
2	Medium	23	32	12	Medium	17	54
3	Low	13	30	13	Medium	12	65
4	Medium	25	29	14	Low	15	70
5	Medium	19	27	15	Low	17	60
6	Low	33	65	16	Low	15	71
7	Low	15	39	17	Low	12	61
8	Low	31	81	18	Zero	21	63
9	Zero	28	59	19	Zero	24	61
10	Low	28	45	20	Low	22	66

parallel programming with Pthreads is greater than with DSL-POPP.

### V. PERFORMANCE EXPERIMENTS

This experiment seek to evaluate the performance in order to identify whether there are significant performance differences between DSL-POPP and Pthreads. The metric associated with this experiment is the execution time, which is used to calculate the speed-up and compare the performance results. The hypotheses for this experiment are the follows:

- 1) **Null hypothesis ( $H_0$ ):** the performance of the algorithms implemented with Pthreads is equal to the implemented using DSL-POPP (Pthreads = DSL-POPP).
- 2) **Alternative hypothesis ( $H_1$ ):** the performance of the algorithms implemented with Pthreads is significantly different than implemented with DSL-POPP (Pthreads  $\neq$  DSL-POPP).

The performance tests were carried out on a machine running Ubuntu-Linux-12.04-server-64bits and the architecture was composed of Intel Xeon X3470 (2.93GHz), 8GB of main memory, and 2TB of disk. We chose four well-known algorithms from the literature to be parallelized by one volunteer from the usability experiment. This programmer implemented these algorithms with Pthreads and DSL-POPP, and we certified that the output result was the same as the sequential version in order to guarantee the parallelization correctness. Also, for each sample we performed 40 random executions to measure/compare the performance and efficiency.

Therefore, the first program Estimates an Integral (EI) over a domain of two dimensions using an averaging technique. The second is a Molecular Dynamic (MD) simulation algorithm. The other application sets up a dense Matrix Multiplication (MM), and the last application counts the Prime Numbers (PN) between 1 to N. We used the SLOCCount tool [18] to present an overview of the physical source lines of code and the development effort for this application. The results are presented in Table III. As can be seen, the estimate of this tool demonstrates that the programmer used more lines of code and the development effort was probably more expensive than with DSL-POPP.

TABLE III  
CODE ANALYSIS.

App.	Source Lines of Code (SLOC)			Development Effort Estimate (Min.)		
	Ori.	DSL-POPP	Pthreads	Ori.	DSL-POPP	Pthreads
EI	91	93	135	8322	8760	12702
MD	249	259	352	24528	25404	35040
MM	99	105	209	9198	10074	20148
PN	78	100	142	7008	9198	13578

It is important to highlight that this tool is not equivalent to the usability experiment because it does not effectively evaluate human interaction. Additionally, it does not consider some parallel programming issues, for example, the programmers have to find the pieces of code that can/must be parallelized, study how to split the computation, how to communicate, and how to perform synchronization. Due to this, it only gives an overview of the development effort to show some trends. For example, applying this test over the matrix multiplication application used in the usability experiment, it estimates a development effort for the sequential version of (52 SLOC) 80.18 hours, for DSL-POPP (59 SLOC) 87.36 hours, and with Pthreads (73 SLOC) 109.30 hours.

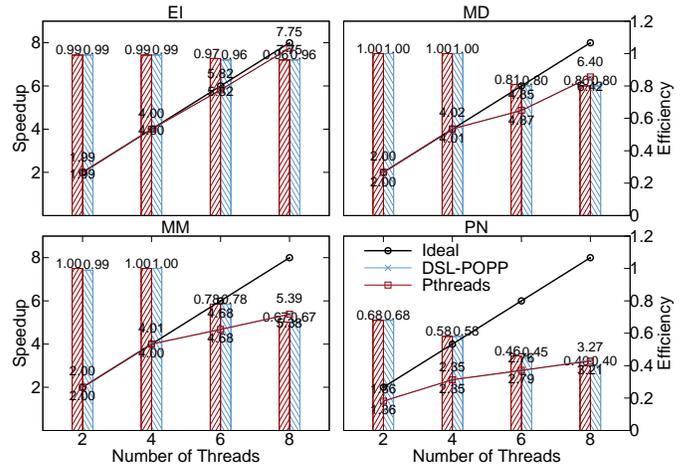


Fig. 3. Speed-up and Efficiency.

The performance and efficiency of these applications are presented in the graphs of the Figure 3. The EI, MD, and

MM achieved ideal speed-ups until 4 threads. However, with 6 and 8 threads they have performance losses due to the use of logical cores. As expected, the PN does not achieve good performance over the other applications, because the algorithm is not embarrassingly parallel. Finally, both speed-up and efficiency presents similar results between DSL-POPP and Pthreads implementations.

Even though it seems there are no significant performance differences according to the graphs of speed-up and efficiency, we performed a significance (Sig) test using the SPSS tool. It was performed with 95% reliability (the probability to reject the  $H_0$  is 5%) over the 40 execution time of each sample so that the Sig. must be  $< 0.05$  for reject the  $H_0$ . The place where the statistics test indicates significant performance differences are in bold (Table IV), thus in these particular cases  $H_0$  is rejected. Overall, all performance tests of MM application presented results without significant differences admitting 95% reliability, which is considered one of the most accurate statistic evaluations for software usability [11].

TABLE IV  
SPSS OUTPUT FOR THE TEST OF SIGNIFICANCE (SIG.)

Threads	EI	MD	MM	PN
2	0.455	0.135	0.059	0.281
4	0.740	0.090	0.174	<b>0.001</b>
6	<b>0.000</b>	<b>0.000</b>	0.837	0.199
8	<b>0.000</b>	0.269	0.381	<b>0.011</b>

## VI. CONCLUSIONS

This paper presented a performance and usability evaluation of a pattern-oriented interface for multi-core architectures. It was performed using software experiments and statistical analysis, whose results demonstrated that the master/slave pattern interface of the DSL-POPP requires less programming effort than Pthreads in the parallelization of a matrix multiplication algorithm. Additionally, we demonstrated that the performance is not significantly affected in four applications, and we discuss the development effort estimated by the SLOCCount tool in order to demonstrate some programming effort trends in these applications.

The comparative analysis with Pthreads library was performed because it is used to implement parallel code generation. Hence, we could see the efficiency of the DSL-POPP for usability and performance issues. However in the future, we intend to evaluate the impact of the parallel programming efforts in applications that can exploit the nested pattern ability, and repeat these experiments for other pattern interfaces available in the DSL-POPP. Also, we plan to compare the performance and usability with other programming interfaces that are not pattern-oriented, such as OpenMP and Cilk.

## ACKNOWLEDGMENTS

Authors wish to thank the research support of FAPERGS (Fundação de Amparo à Pesquisa do Estado do Rio Grande do Sul) and CAPES (Coordenação de Aperfeiçoamento Pessoal de Nível Superior). Additionally, authors would like to thank

the financial support of FACIN (Faculdade de Informática) and PPGCC (Programa de Pós-Graduação em Ciência da Computação).

## REFERENCES

- [1] M. Aldinucci, M. Danelutto, P. Kilpatrick, M. Meneghin, and M. Torquati. Accelerating Code on Multi-cores with FastFlow. In *Euro-Par 2011 Parallel Processing*, volume 6853, pages 170–181, Berlin, Heidelberg, August 2011. Springer-Verlag.
- [2] M. Aldinucci, M. Danelutto, P. Kilpatrick, M. Meneghin, and M. Torquati. An Efficient Unbounded Lock-Free Queue for Multi-core Systems. In *Proc. of 18th Intl. Euro-Par 2012 Parallel Processing*, volume 7484, pages 662–673, Rhodes Island, Greece, August 2012. Springer.
- [3] S. Bromling, S. MacDonald, J. Anvik, J. Schaeffer, D. Szafron, and K. Tan. Pattern-Based Parallel Programming. In *Parallel Processing, 2002. Proceedings. International Conference on*, pages 257–265, British Columbia, Canada, 2002. IEEE Computer Society.
- [4] H. Chafi, A. Sujeeth, K. Brown, H. Lee, A. Atreya, and K. Olukotun. A Domain-specific Approach to Heterogeneous Parallelism. In *Proceedings of the 16th ACM Symposium on Principles and Practice of Parallel Programming*, volume 1, pages 35–46, New York, NY, USA, February 2011. ACM.
- [5] B. Chapman, G. Jost, and R. Pas. *Using OpenMP: Portable Shared Memory Parallel Programming*. Massachusetts Institute of Technology, London, England, 2008.
- [6] M. Cole. *Algorithmic Skeletons: Structured Management of Parallel Computation*. MIT Press, Cambridge, USA, 1989.
- [7] A. Field. *Discovering Statistics Using SPSS*. SAGE, Dubai, EAU, 2009.
- [8] H. González-Vélez and M. Leyton. A Survey of Algorithmic Skeleton Frameworks: High-level Structured Parallel Programming Enablers. *Softw. Pract. Exper.*, 40(12):1135–1160, 2010.
- [9] D. Griebler and L. G. Fernandes. Towards a Domain-Specific Language for Patterns-Oriented Parallel Programming. In *Programming Languages - 17th Brazilian Symposium - SBLP*, volume 8129 of *Lecture Notes in Computer Science*, pages 105–119, Brasilia, Brazil, October 2013. Springer Berlin Heidelberg.
- [10] Intel. Thread Building Block (Intel TBB), Extracted from <https://www.threadingbuildingblocks.org/>, 2014.
- [11] N. Juristo and A. M. Moreno. *Basics of Software Engineering Experimentation*. Springer, Boston, USA, 2001.
- [12] L. V. Kale and S. Krishnan. CHARM++: A Portable Concurrent Object Oriented System Based on C++. In *Proceedings of the Eighth Annual Conference on Object-oriented Programming Systems, Languages, and Applications*, pages 91–108, New York, NY, USA, October 1993. ACM.
- [13] C. E. Leiserson. The Cilk++ Concurrency Platform. In *Proceedings of the 46th Annual Design Automation Conference*, volume 1, pages 522–527, New York, NY, USA, July 2009. ACM.
- [14] V. Pankratius. Automated Usability Evaluation of Parallel Programming Constructs (NIER Track). In *Proceedings of the 33rd International Conference on Software Engineering, ICSE '11*, pages 936–939, New York, NY, USA, May 2011. ACM.
- [15] M. Raeder, D. Griebler, L. Baldo, and L. G. Fernandes. Performance Prediction of Parallel Applications with Parallel Patterns Using Stochastic Methods. In *Sistemas Computacionais (WSCAD-SSC), XII Simpósio em Sistemas Computacionais de Alto Desempenho*, pages 1–13, Espírito Santo, Brasil, October 2011. IEEE Computer Society.
- [16] C. Sadowski and A. Shewmaker. The Last Mile: Parallel Programming and Usability. In *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, volume 1, pages 309–314, New York, NY, USA, November 2010. ACM.
- [17] D. Szafron and J. Schaeffer. An Experiment to Measure the Usability of Parallel Programming Systems. *Concurrency - Practice and Experience*, 8(2):147–166, 1996.
- [18] D. A. Wheeler. SLOCCount, Access from <http://www.cic.unb.br/facp/cursos/ps/slides/doc/SLOCCount.html>, 2014.

# Performance Benchmarking of BPEL Engines: A Comparison Framework, Status Quo Evaluation and Challenges

Cedric Röck, Simon Harrer and Guido Wirtz

Distributed Systems Group, University of Bamberg  
An der Weberei 5, 96047 Bamberg, Germany

E-mail: {cedric.roeck | simon.harrer | guido.wirtz}@uni-bamberg.de

## Abstract

*Despite the popularity of BPEL engines to orchestrate complex and executable processes, there are still only few approaches available which help to find the most appropriate engine for individual requirements. One of the more crucial comparison factors for middleware products in industry are the performance characteristics. There exist multiple studies in both industry and academia testing the performance of BPEL engines, which differ in focus and method. We aim to compare the methods used in these approaches and provide guidance for further research in this area. Based on the related work in the field of performance testing, we created a process engine specific comparison framework, which we used to evaluate and classify nine different approaches that were found via a systematical literature survey. With the results of the status quo analysis in mind, we derived directions for further research in this area.*

**Keywords:** SOA, BPEL, engines, performance testing

## 1. Introduction

Over the past few years, the research concerning the Web Services Business Process Execution Language (BPEL) [21] made huge progress and focused on arising chances and challenges for businesses [5]. Based on service-oriented architectures (SOAs), one of the major trends in the development of business information systems, BPEL steadily became the standard for Web Service based business orchestrations [15]. Strongly connected to the growing popularity is the development of more complex systems, which also leads to an increased error-proneness of the developed software [3]. Therefore, the need to intensively test those SOAs also gains more importance. However, there still is a tremendous deficit in terms of proper testing tool support, which has been considered to be one of the major problems of SOAs [6].

A classic goal of performance testing has always been to compare competing platforms, allowing the selection of the best fitting product for particular needs [28]. Despite the grown acceptance of SOAs, performance testing in this area still

lacks some major aspects. While the number of approaches steadily grows, the majority focuses solely on the evaluation of single Web services instead of middleware components or even complete systems [13]. However, “the middleware used to build a distributed application often determines the overall performance of the application” [8, p. 2] and should therefore be considered at least as carefully as the choice of partner services involved in a process. Looking at performance testing of BPEL processes and their engines, a few studies have been conducted in industry and academia. But there is no standardized benchmark, let alone a commonly agreed upon testing methodology [7]. As a consequence, the current view on the status quo and further research topics is clouded. In this work, we aim to provide a clear view on both, the current state in performance testing of process engines and further challenges in this area.

Our contribution<sup>1</sup> comprises three parts. First, the creation of a framework to compare performance benchmarking approaches of process engines in Section 2. Second, a literature survey of the status quo in performance benchmarking of BPEL engines reusing the previously created comparison framework in Section 3. Third, directions on improving performance benchmarking of process engines are given based on the comparison results in Section 4. Our work concludes in Section 5, which also outlines future work.

## 2. Comparison Framework and its Criteria

Performance testing refers to the usually technical examination of a system under test (SUT) with the aim to analyse and validate a product’s characteristics. It is known for allowing a measurement-based comparison of different products and platforms under similar conditions, hence offering valuable information for purchase decisions [28]. Furthermore, it is also used to identify bottlenecks and verify the requested quality of service (QoS) attributes in nearly finished software, i.e., whether nonfunctional performance requirements are met [29]. According to Koziolok [14], the performance of a software component is influenced by five factors: its implementation, the usage profile, deployment platform, required services, and

<sup>1</sup>For more details, see the accompanying technical report [22].

resource contention. Based on these influences, we created a framework to enable the classification of BPEL engine performance tests. In the following, we present four primary and three secondary criteria, by which performance tests can be analysed and made comparable.

Measurement based performance tests can be executed in several ways depending on the intended purpose of the test. These different strategies establish the **types of performance tests**, namely *baseline*, *stress* and *load* tests [18]. *Baseline tests* measure the response time behaviour of an application with only a single request or user, hence represent the best-case scenario. It can be used for product comparison or as a benchmark baseline for other test types, e.g., for load tests [20, pp. 38-40]. A *load test* verifies the application behaviour in typical and expected situations, including load peaks with multiple concurrent users. It is often used to assure performance related QoS attributes [18] [19, p. 291]. Pushing an application beyond the expected level of load and peak conditions is called *stress testing*. Its goal lies in revealing bugs and unexpected behaviour that only appear in extreme conditions [18]. The results reflect the worst-case scenario and unearths the capacity limits of the system under test [20, pp. 38-40] [19, p. 291].

**Workloads** are defined as the sum of all inputs that are received by the SUT. They are considered to be one of the key aspects for assuring the validity of performance test results [19,25,28,29]. In the context of BPEL engines, workloads consist of the invoked processes and their dependent Web services, but also include the request messages and the strategy for sending these requests. According to [19, pp. 265-266], the workload of software performance benchmarks can be categorised into four *workload types*. *Basic operations* refer to the smallest operations, i.e., all supported and standardized activities in the context of a BPEL engine, and supply fine-grained performance characteristics. *Toy-benchmarks*, usually implementing classic puzzles or being proof-of-work concepts, are of not much use for performance tests. *Kernels* are core parts of real programs. They represent the most important or time consuming parts and provide an intermediate level in terms of granularity and realism. In our context, kernel processes implement patterns, e.g., the workflow patterns [27] that were extracted from a large corpus of real world processes. *Real programs*, or real processes in our case, are often seen as the most important workload type as their results are most accurate for estimating the performance of production systems [2, 25]. Therefore, real workloads are also used in other domains, e.g., for benchmarking SQL databases with TPC-C.

**Workload injection strategies** can be subdivided into a) *continuously* injecting the workload and b) injecting the workload with an *arrival rate*. *Continuously* injected workloads test the engine with a constant load, once reaching the plateau phase. This plateau, however, can either be reached with a big bang (default approach), or *stepwise*, by slowly increasing the number of concurrent users over a specific period [20, p. 41]. The *arrival rate* can be either fixed or dynamic to simulate even more realistic peaks [25].

If the SUT and load generating clients are co-located, one

has to closely observe the system for overloads. Therefore, they are often installed on separate systems, i.e., *distributed*. Moreover, a separation allows the load to be created from distributed clients, providing more realistic situations and assuring the saturation of the SUT.

The evaluation of software performance usually considers multiple metrics. In this study, we use three **performance metrics**: *latency*, *throughput* and *utilization*. The *latency* or response time [8] defines the time span between sending a request and having received the full response. It is influenced by multiple factors, e.g., the execution of partner services, network delays and message parsing [24]. *Throughput* is the most commonly used metric for software performance, defining the number of transactions that are completed within a period. In the context of BPEL engines, the term transaction can refer either to the completion of requests or process instances. The *utilization* reflects the degree to which a capacity is used and refers to many parts of the testbed, e.g., the network, database, server, or load-generating clients [20, p. 29]. As metrics differ in semantics and focus, they depend upon the test types. For example, a baseline test using any throughput metric is meaningless as there is only one active request at a time.

In addition to the primary criteria, we distinguish between three **secondary criteria**. First, the **number of the engines and their license**, being either open source or proprietary. Second, the **setup of the test bed**, being either automated or done manually, which has effects on the experiment's reproducibility. And third, as BPEL engines provide plenty of **configuration options**, we distinguish whether the processes are executed in-memory or not, which we denote as *persistence*.

### 3. Literature Study

The results of our systematic literature study<sup>2</sup>, i.e., the nine approaches and their classifications, are shown in Table 1. Each row and each column refers to an approach and a comparison criterion, respectively. Cells represent the findings, with empty cells denoting the absence of a criterion and cells marked *n/a* denoting that the approach did not provide any data regarding this criterion.

The benchmark conducted by *SOABench* [4] includes three engines and performs load tests using four BPEL processes. Two processes are built with the `flow` activity, the other two use either the `sequence` or the `while` activity. All four processes invoke mocked external services, thus, use the `invoke` activity as well. The performance is measured via latency and throughput metrics. Moreover, *SOABench* features the automated generation, execution and analysis of testbeds, allowing reproducing the test results. Workloads are either injected based on a given arrival rate, or delayed by a *thinking time* between consecutive invocations.

The load test of *OpenESB* [26] is focused on throughput metrics. Its workload is a single process, using all supported BPEL activities, i.e., a toy benchmark, and is injected stepwise.

<sup>2</sup>The method is detailed in the accompanying technical report [22].

**Table 1. Literature analysis, comparing most important performance test factors**

Approaches	Test Type			Workload			Workload Injection				Metrics			Engines		Config	Testbed
	Baseline	Load	Stress	Type	# of Processes	Ext. Service	Distributed	Continuously	Stepwise	Arrival Rate	Throughput	Latency	Utilization	Open Source	Proprietary	Persistence	Automated
SOABench [4]		x	x	basic	4	x	x	x		x	x	x		2	1		x
OpenESB [26]		x		toy	1	n/a			x		x			1			
ActiveVOS [1]		x		real	2		n/a	n/a	n/a	n/a	x			1		x	
Sliver [10]	x			kernel	12	x		x				x	x	2			
Workload model [9]		x		toy	1					x				1			
Intel & Cape Clear [12]		x		real	2	x	x	x			x	x	x		1	x	
Roller [23]		x		real	1	x		x			x				1		
SWoM [17]		x		basic	2	x		x			x	x			3		
FACTS [16]		x		real	1	x	n/a	n/a	n/a	n/a		x		1			

Similarly, throughput metrics are measured in the load test of *ActiveVOS* [1]. The test uses two functional processes, with one being analysed with persistence enabled or disabled.

In [10], *Sliver* is compared with *ActiveBPEL*, using a baseline test measuring request latency and memory utilization. The workload consists of twelve workflow patterns [27], i.e., kernels, realized as BPEL processes, which utilize external services and are invoked in consecutive requests.

Validating their workload model, Din et al. [9] have performed a load test that focused only on the response time behaviour of the used *ActiveBPEL* engine. The tested process uses correlations but does not call external services. The workload is injected by several virtual users over a total duration of two minutes, injecting 20 new processes per second.

The load test of Intel and Cape Clear [12] focuses on latency, throughput and utilization measures of the tested *Cape Clear 7* engine. The two processes implement a typical industrial functionality and invoke external services, while the workload is continuously injected from a set of distributed clients. Additionally, the test focuses on the effect of persistence on the performance metrics.

Verifying Roller’s [23] proposals, he has conducted a load test on one proprietary engine and measured throughput metrics. The tested workload is a single realistic BPEL process, including the invocation of external services, which is continuously called by the testing clients.

Benchmarking three BPEL engines, Langerer et al. [17] have conducted a load test focusing on throughput and latency metrics. The workload, which is continuously injected, consists of two processes. One uses the `assign` activity, the other one calls an external service with the `invoke` activity.

The load test of Liu et al. [16] tests the response time behaviour of a single realistic process, which includes external services and was deployed on the *ActiveBPEL* engine.

#### 4. Discussion and Further Suggestions

This section analyses the findings of our evaluation and discusses which parts of BPEL performance testing should be strengthened in future work. As the approaches differ in many

aspects and follow no common schema, we focus on patterns per criterion, i.e., compare the results column-wise.

The approaches mainly execute load tests, whereas, in addition, Bianculli et al. [4] also applies stress testing. Solely Hackmann et al. [10] perform baseline tests, leaving room for further evaluations, by means of baseline and stress tests.

The workload types differ for all approaches, with four using real workloads, one using kernels, and two using basic and toy each. For each workload, the number of processes also varies, with four approaches using only a single process, three using two, one using four and one using twelve processes. The majority (6/9) uses processes that invoke external services, thus the test always includes the performance characteristics of the `invoke` activity. For the remaining approaches, two do not use external services, while it is unknown for the last one. For all workload types, a larger corpus of processes would help to improve the meaningfulness of the test results. Regarding the basic category, we propose to have a process per feature to be able to compare their performance characteristics. For kernel processes, we suggest to cover more pattern catalogues, whereas for real processes, we propose to use distinct real processes from various use cases in different domains. As external services are a crucial part of BPEL processes, they should not be neglected in further studies as well.

Concerning the workload injection, two approaches did not mention it at all. For some approaches, it is neither stated, nor can it be deduced. As it is a crucial part of a performance evaluation, we advise to explicitly state the chosen strategy.

Regarding the metrics, latency and throughput are used by six approaches each, whereas only two measure utilization. In this context, Intel and Cape Clear [12] provide the most complete approach as they measure all three metric types. Three approaches use two metric types, while the majority (5/9) measures only metrics of a single type. Hence, we propose to focus on the neglected utilization metric, which can reveal interesting characteristics of the engines as well as ensure that there are no system overloads falsifying any results.

The number of engines under test range from one up to three per approach, limiting their relevance for buying decisions. Three approaches compare the performance of multiple

engines, while the remaining six evaluate the performance of a single engine. Only SOABench [4] compares open source with proprietary engines. But as the proprietary engine ActiveVOS incorporates the open source engine ActiveBPEL, SOABench basically test the open source one. Hence, there is no proper performance comparison of open source with proprietary engines, leaving room for further work in this area.

Regarding the configuration opportunities of the BPEL engines, only two approaches [1, 12], both evaluating only a single engine, tested their engines in different configurations, namely either execute their processes in-memory or not. As most engines offer multiple options, it shows that this has been neglected in research, despite its importance. However, when comparing more than one engine, it has to be ensured that all engines equally support the capability.

With only Bianculli et al. [4] allowing to automatically setup the testbed and execute the tests, it is very hard to redo the experiment for all other approaches. Moreover, only [4, 12, 17, 23] published their detailed test setup, processes and tools, which are essential for the repeatability of these tests.

None of the approaches allow analysing the influence of environmental aspects, for instance the system's hardware, database or influences of long-running transactions on the engines' performance. However, modern multi-core systems and solid-state drives provide new challenges and opportunities for differentiation among middleware products.

One additional problem is that none of the approaches takes into account that BPEL engines greatly vary in their degree of support of the BPEL specification [11], i.e., they implement different subsets of the BPEL features. We propose to take these results into account, creating and selecting appropriate workloads for the engines to be compared.

## 5. Conclusion and Future Work

In our work, we created a comparison framework with which existing performance benchmarking approaches of process engines, and BPEL engines in particular, can be classified. We applied our comparison framework onto nine methodically found approaches, revealing their differences and similarities. Based on the findings, we derived guidance for further research in the areas which have been neglected so far.

In future work, we want to apply our comparison framework and method onto other studies targeting other process engines and their languages, and fill in the open gaps that were revealed during this study.

## References

- [1] Active Endpoints Inc. Assessing ActiveVOS Performance. <http://bit.ly/R60NPY>. 2014-01-30.
- [2] A. Avritzer, J. Kondek, D. Liu, and E. J. Weyuker. Software Performance Testing Based on Workload Characterization. In *WOSP*, 2002.
- [3] V. R. Basili and B. T. Perricone. Software Errors and Complexity: An Empirical Investigation. *CACM*, 1984.
- [4] D. Bianculli, W. Binder, and M. L. Drago. Automated Performance Assessment for Service-oriented Middleware: A Case Study on BPEL Engines. In *WWW*, 2010.
- [5] M. Bozkurt, M. Harman, and Y. Hassoun. Testing and Verification in Service-Oriented Architecture: A Survey. *Software Testing, Verification and Reliability*, 2012.
- [6] G. Canfora and M. D. Penta. Testing Services and Service-Centric Systems: Challenges and Opportunities. *IT Professional*, 2006.
- [7] S. Chen, L. Bao, and P. Chen. OptBPEL: A Tool for Performance Optimization of BPEL Process. In *Softw. Comp.*, 2008.
- [8] G. Denaro, A. Polini, and W. Emmerich. Early Performance Testing of Distributed Software Applications. In *WOSP*, 2004.
- [9] G. Din, K.-P. Eckert, and I. Schieferdecker. A Workload Model for Benchmarking BPEL Engines. In *ICSTW*, 2008.
- [10] G. Hackmann, M. Haitjema, C. Gill, and G.-C. Roman. Sliver: A BPEL Workflow Process Execution Engine for Mobile Devices. In *ICSOC*. 2006.
- [11] S. Harrer, J. Lenhard, and G. Wirtz. BPEL Conformance in Open Source Engines. In *SOCA*, 2012.
- [12] Intel and Cape Clear. BPEL scalability and performance testing. Technical report, Intel and Cape Clear, 2007.
- [13] L. Juszczak and S. Dustdar. Script-Based Generation of Dynamic Testbeds for SOA. In *Socially Enhanced Services Computing*. Springer, 2011.
- [14] H. Koziolok. Performance Evaluation of Component-based Software Systems: A Survey. *Performance Evaluation*, 2010.
- [15] T. v. Lessen, D. Lübke, and J. Nitzsche. *Geschäftsprozesse automatisieren mit BPEL [Automating Business Processes with BPEL]*. dpunkt.verlag, 2011.
- [16] A. Liu, Q. Li, L. Huang, and M. Xiao. Facts: A framework for fault-tolerant composition of transactional web services. *Services Computing, IEEE Transactions on*, 2010.
- [17] C. Längerer, J. Rutschmann, and F. Schmitt. Performance-Vergleich von BPEL-Engines [Performance Comparison of BPEL Engines]. Technical report, 2006.
- [18] J. Meier, C. Farre, P. Bansode, S. Barber, and D. Rea. *Performance Testing Guidance for Web Applications: Patterns & Practices*. Microsoft Press, 2007.
- [19] D. A. Menascé. *Capacity Planning for Web Services: Metrics, Models, and Methods*. Prentice Hall, 2002.
- [20] I. Molyneaux. *The Art of Application Performance Testing*. O'Reilly Media, 2009.
- [21] OASIS. *Web Services Business Process Execution Language*, 2007. v2.0.
- [22] C. Röck, S. Harrer, and G. Wirtz. Testing BPEL Engine Performance: A Survey. Technical report, Univ. of Bamberg, 2014.
- [23] D. Roller. *Throughput Improvements for BPEL Engines: Implementation Techniques and Measurements Applied to SWoM*. PhD thesis, IAAS, Stuttgart, Germany, 2013.
- [24] F. Rosenberg, C. Platzer, and S. Dustdar. Bootstrapping Performance and Dependability Attributes of Web Services. In *ICWS*, 2006.
- [25] A. J. Smith. Workloads (Creation and Use). *Com. ACM*, 2007.
- [26] Sun Microsystems. Benchmarking BPEL Service Engine. <http://bit.ly/1jkssHd>. 2014-01-30.
- [27] W. van der Aalst, A. ter Hofstede, B. Kiepuszewski, and A. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 2003.
- [28] E. J. Weyuker and F. I. Vokolos. Experience with Performance Testing of Software Systems: Issues, an Approach, and Case Study. *IEEE Trans. Softw. Eng.*, 2000.
- [29] M. Woodside, G. Franks, and D. C. Petriu. The Future of Software Performance Engineering. In *FOSE*, 2007.

# A Tool for Trade-off Resolution on Architecture-Centered Software Development

Italo Carlo Lopes Silva  
Federal University of Alagoas  
italocarlo@nti.ufal.br

Patrick H S Brito  
Federal University of Alagoas  
patrick@ic.ufal.br

Baldoino Fonseca dos  
Santos Neto  
Federal University of Alagoas  
baldoino@ic.ufal.br

Evandro Costa  
Federal University of Alagoas  
evandro@ic.ufal.br

Hemilis Rocha  
Federal University of Alagoas  
joysesel@gmail.com

## ABSTRACT

The success of a software project is strongly related with architectural design. However, designing the right Software Architecture is a very subjective task and takes a long time, being much influenced by architect's experience and the quality of requirements engineering. A big problem emerges when the trade-off among quality requirements is not properly solved during requirements engineering. The objective of this paper is to present a trade-off resolution process supported by a tool for helping young requirements engineers and software architects on the hard task of specifying the system requirements, detecting and solving trade-offs among them.

## Keywords

Requirements engineering, Trade-off resolution, Software architecture

## 1. INTRODUCTION

Develop software with low cost of production and maintenance, to be delivered in a shorter time with requirement of high quality has become even more urgent and necessary. In this way, one of the key elements of success for the software projects is the architectural design.

Software architectures represent explicitly the structure of systems, and it is one of the earliest artifacts that permits the analysis of system quality attributes, such as dependability, including reliability, availability, security and safety [1].

Although each system has its particularity, it is common that applications that belong to the same domain are to use similar architecture. In this way, the architectural styles are very important, because they define a family of systems in terms of a pattern of structural organization, thus providing meta-models that can be applied in recurrent problems of a domain [5]. However, designing the right Software Architecture is a very subjective task and takes a long time, being much influenced by architect's experience as well as the quality of requirements engineering. A big problem emerges during the architectural design phase when trade-offs amongst quality attributes have not been identified and properly managed during requirements engineering.

Each alternative solution satisfies different functional and non-functional requirements to varying extents. Selecting a solution among multiple alternatives involves managing

trade-offs among requirements, with respect to stakeholders preferences and consequences of alternatives on the requirements [2].

The main problems when managing trade-offs among requirements and deciding over alternative design solutions are [2]: (1) Extensive Data Collection, (2) Manual prioritization, (3) Incomparable Scales and (4) Scalability.

The objective of this paper is to present a trade-off resolution process with tool support for helping young requirements engineers and software architects on the hard task of specifying the system quality requirements, detecting and solving trade-offs among them. The tool has a rule-based architecture which uses an expert system engine for identifying trade-offs and provide directions for managing them. The rules database can be seen as a knowledge repository that contains technical solutions based on patterns and good experiences of specialists.

The proposed tool is presented in details, including its requirements, its software architecture and detailed design, as well as the structure of the rules for representing the specialist's knowledge.

The rest of the paper is organized as follows. Section 2 presents some related work. Section 3 presents the trade-off management process proposed in this paper. Section 4 presents the tool developed for supporting the aforementioned process. Finally, Section 5 presents some concluding remarks and directions of future work.

## 2. RELATED WORK

Faced with the typical absence of reliable quantitative data, some Requirements Engineering (RE) techniques, such as Tropos [6] and i\* [15] treat quality attributes as soft goals, thus enabling to reason about the partial satisfaction of such goals by using qualitative labels such as partially satisfied, sufficiently satisfied, partially denied, and fully denied. Although it is considered a good evolution, the subjectivity of classification and evaluation of quality attributes could produce conflicting quality requirements. When conflicts emerge among quality attributes, the management of the trade-offs becomes a critical issue [13].

Elahi and Yu [2] present a semi-automated tool that uses Even Swaps process [7] for decision making related to conflicting requirements. Although this tool addresses the same focus of our approach, we focus on managing trade-off involving specifically software quality attributes. Besides, we

also aim to clarify the meaning of quality attributes and the relationship among them, in order to improve the confidence of the stakeholder on the final quality attribute specification.

García-Mireles *et al.* [4] present a conceptual framework for dealing with software quality trade-offs. Such solution presents a set of activities for managing trade-offs based on a systematic comparison of CMMI and ISO-12207 specifications. Our work is compliant with García-Mireles *et al.*'s approach and can be seen as a tool-supported instance of such conceptual framework.

### 3. A PROCESS FOR MANAGING TRADE-OFFS AMONG QUALITY ATTRIBUTES

The Trade-offs management is an important task, which can influence many activities of software development. In terms of quality attributes, the trade-off management is specially important in the context of architectural design. Figure 2 presents an overview of the proposed process, which also contextualizes the trade-off management activities with architectural design activities.

The first activity, named 01-Prepare the Environment focuses on the persistence of specialist's knowledge related to requirements engineering. This activity should be executed at the beginning, after the deployment of the proposed solution, but can also be re-executed anytime, in order to update the knowledge database with expertises. The second activity, named 02-Specify Quality Requirements consists on the utilization of the proposed tool using the existing specialized knowledge. These activities will be detailed in the next subsections.

#### 3.1 Registering the Quality Attributes

The first thing to be done in whole process is the register of quality attributes (Activity 01.1). In the context of this paper, we have registered the quality attributes proposed by ISO/IEC 9126 [3]. The main idea consists in using these information to collect user's preferences about the desired quality requirements.

These registered information about the quality attributes will be used to collect the attributes' priorities, thus the end users won't need to know technical terms or details about a particular domain or technology. In this sense, we considered adopting a mapping between natural language and quality requirements, where the engineers and architects will associate each quality attribute with a text about this attribute. The end user must understand this text easily, reducing the effort to collect the priority of a particular requirement correctly.

Additional information such as features, advantages and disadvantages could be registered to improve the understanding about a specific requirement. Each registered quality attribute will be associated to an element in a knowledge repository. This association is very important, because once the priorities of requirements were filled (Activity 02.1), the tool will retrieve the elements which are associated with each attribute and will fill properly the working memory with their priorities.

#### 3.2 Registering the Trade-off Patterns

After registering the quality attributes, it is necessary to identify potential trade-offs among them (Activity 01.2). The potential trade-offs, called patterns, consist on well known

scenarios in which two or more quality attributes conflict each other. For example, on a web application, the software architect noticed that when the number of concurrent requests increases, the application becomes slower. This scenario could generate a pattern exemplifying a case when scalability and performance conflict each other.

The identification of such trade-off patterns can be done both based on previous experiences and on the specialized literature. In the context of this paper, we have registered trade-off patterns based on real scenarios reported by literature [11].

Once the stakeholder confirms the priorities for each quality attribute, they will be used as input to the working memory of trade-off management process (Activity 02.3), using for this the association made in the Register Quality Attributes activity (Activity 01.1), that will be responsible to identify and solve possible trade-offs among two or more requirements.

#### 3.3 Registering Rules for Prioritize Requirements

Then, in order to finish the environment preparation, it is necessary to define rules to support trade-offs management (Activity 01.3). Such rules are described through a list of easy questions, whose answers reflect priorities over the requirements.

For example, in the context of the trade-off scenario of the hypothetical web application, a helpful question could be: "Would you limit the number of concurrent request in order to provide a better quality of service in terms of performance?". Each registered trade-off scenario should be associated to at least one question and each question should have an impact associated to its answers, which can increase or decrease the weight of the quality attributes, thus impacting on its priority, as can be seen in Figure 1 that shows the meta-model for representing the rules. For simplifying the diagram, most attributes have been omitted. It is important to emphasize that the association between TradeOffPattern and Domain allows the definition of either domain-specific and general-domain trade-off patterns.

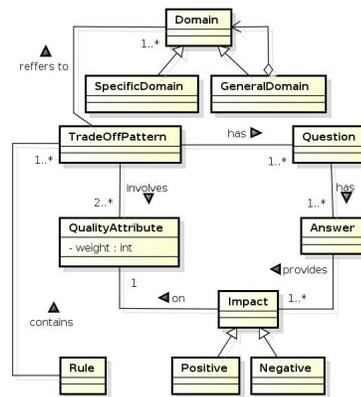


Figure 1: Metamodel Describing Rule Structure

#### 3.4 Eliciting the Quality Requirements

Firstly, the activity named Specify Quality Requirements (Activity 02) starts with the stakeholder trying to identify the quality attributes which his system must meet (Activ-

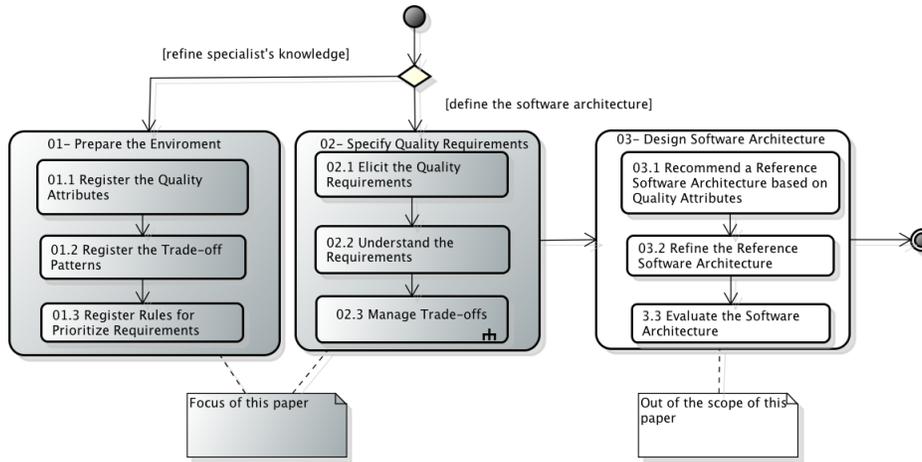


Figure 2: Overview of the Proposed Process

ity 02.1). The tool will retrieve all the quality requirements persisted in the database and will show a list with information about each one of attributes that were registered previously (Activity 01.1).

This tool will give 100 points to the stakeholder to distribute among the attributes which he is think that is important for his software. All attributes start with the same priority, that is zero. The importance of a requirement will be measured using the points associated for each attribute.

Once the stakeholder confirms the priorities for each quality attribute, they will be used as input to the working memory of trade-off management process (Activity 02.3), using for this the association made in the activity named Register the Quality Attributes (Activity 01.1), that will be responsible to identify and solve possible trade-offs among two or more requirements.

### 3.5 Understanding the Requirements

For each quality attribute selected, it is important to provide extra details in order to validate properly the chosen quality attributes (Activity 02.2). This activity will give support the previous activity (Activity 02.1), providing extra information about each one of registered attributes. Anytime, when the stakeholder has some doubt or the information is not sufficiently clear about a requirement, he will can use this resource to clarify any misunderstood about this item. Such details can be, for example, a detailed description of the quality attribute with an illustrating example, its features, advantages and disadvantages.

### 3.6 Managing Trade-offs

Finally, after having the real list of quality attributes aimed for the application, might be necessary to manage trade-off among them (Activity 02.3). For this, it is necessary to follow five sequential steps: (1) Identify trade-off patterns among the selected quality attributes; (2) Ask questions related to the identified trade-off patterns; (3) Compute answer's impact and update the weight attribute (positive and negative); (4) Sort quality attributes based on the weight attribute; (5) Classify requirements following a predefined taxonomy.

The taxonomy for classifying quality attributes aims to

reduce subjective interference. Analyzing existing guidelines [9], we have designed a taxonomy for classifying quality attributes in four possible levels: (1) **Mandatory**, for those attributes whose bad functioning could preclude the use of the system; (2) **Important**, for those attributes that the stakeholder would be willing to pay more for them; (3) **Desirable**, for those attributes that the stakeholder wants, but would not be willing to pay more for them; and (4) **Not Important**, for those attributes that the stakeholder does not require, but can be implemented if do not impact on cost. After having the quality attributes sorted by importance (weight), the stakeholder should classify in the four categories proposed. A possible way for automating this task is to establish a threshold value of weight. But in case of keeping trade-off even after the aforementioned resolution process, the software architect should be warned about the problem and solve it manually.

## 4. A TOOL FOR SUPPORTING THE PROPOSED PROCESS

A generic trade-off resolution tool for supporting the process presented in Section 3 has been developed. We have used the Java programming language and MySQL database using JPA framework for persistence. During the tool specification, we opted to perform the two first activities of the execution process at the same time.

After selecting the desired quality attributes, the user should confirm in order to manage trade-offs. The initial weight of the quality attributes is zero, thus indicating that all of them initially have the same priority. Alternatively, the user can choose to manually set the initial priority. For this, 100 points are given to the user, who can distribute these points over the quality attributes. When the priorities were filled, all the information will be passed to a working memory of the expert system that manages trade-offs (Activity 02.3). After the trade-off management process, such importance order is updated, based on the final weight.

For providing more details about the design of tool, Figure 3 presents its software architecture, which follows an heterogeneous architectural style which combines data-centered and rule-based characteristics.

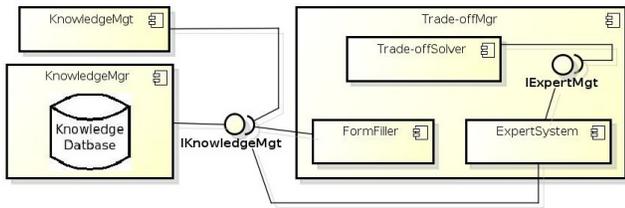


Figure 3: Architecture of the Developed Tool

The **ExpertSystem** component consists on a rule-based system that emulates the decision-making ability of a human expert [10]. Such systems are designed to solve complex problems by reasoning about knowledge by using an inference engine, which is a computer program that tries to derive answers from a knowledge base. The developed tool reused an existing rule-based inference engine called Inabit [14, 12], an Intelligent Agent Building Tools. Inabit may be seen as a kind of rule-based expert system shell, but it may be also seen as a software framework by providing a set of facilities to be used by the developers of intelligent applications. Therefore, the main idea is to provide effective support with facilities to both domain expert/knowledge engineer and developer. Inabit provides various features commonly used in a typical expert system, such as (i) inference by using goal driven reasoning (backward chaining) or data driven reasoning (forward chaining), (ii) basic explanation facilities, (iii) facilities to create and edit a given knowledge base.

For this, The **FormFiller** component will have the task of passing the stakeholder's answers regarding quality attributes to the **ExpertSystem** component to be inserted in the working memory of framework. With all these information, the **Trade-offSolver** component will use the framework, running the forward chaining reasoning in the knowledge stored to identify the trade-offs problems. Once these trade-offs were identified, the tool will try to solve them presenting questions associated to the trade-off patterns, as presented in Section 3.

## 5. CONCLUSION AND FUTURE WORK

This paper presented an approach to support the requirements engineering phase of a software. The proposed solution comprises a systematic process supported by a tool for helping young requirements engineers and software architects on the hard task of specifying the system requirements, detecting and solving trade-offs among them. The tool has a rule-based architecture which uses an expert system engine that behaves as a requirement engineer, thus keeping the technical knowledge and experience at the software company. Although other existing approaches address the same target, we focus on managing trade-off involving specifically software quality attributes. Besides, we also aim to clarify the meaning of quality attributes and the relationship among them, in order to improve the confidence of the stakeholder on the final quality attribute specification.

As an immediate future work, the approach should be evaluated in real scenarios and domains, with a high number of volunteers. Furthermore, it is also being evolved in order to link trade-off resolution of quality attributes to design decisions related to the choice of architectural styles during the architectural design phase. In addition to supporting

the architectural design by recommending reference architectures, this integration also aims to increase the quality of trade-off resolution, since the trade-offs are better resolved when contextualized with design artifacts, such as software architecture and source-code [8].

## 6. REFERENCES

- [1] L. Bass and R. Kazman. Architecture-based development. Technical Report 007, Carnegie Mellon University, Pittsburgh,PA, apr. 1999.
- [2] G. Elahi and E. Yu. A semi-automated decision support tool for requirements trade-off analysis. In *IEEE 35th Annual COMPSAC*, pages 466–475, 2011.
- [3] I. O. for Standardization. Iso standard 9126: Software engineering - product quality, parts 1, 2 and 3, 2001.
- [4] G. A. García-Mireles, M. Á. Moraga, F. García, and M. Piattini. A framework to support software quality trade-offs from a process-based perspective. In *EuroSPI*, pages 96–107, 2013.
- [5] D. Garlan and M. Shaw. An introduction to software architecture. Technical Report 166, Carnegie Mellon University, Pittsburgh,PA, jan. 1994.
- [6] P. Giorgini, J. Mylopoulos, E. Nicchiarrelli, and R. Sebastiani. Formal reasoning techniques for goal models. *J. Data Semantics*, 1:1–20, 2003.
- [7] J. S. Hammond, R. L. Keeney, and H. Raiffa. *Smart choices : a practical guide to making better life decisions*. Broadway Books, USA, 2002.
- [8] K. Henningsson and C. Wohlin. Understanding the relations between software quality attributes - a survey approach. In *In Proceedings of 12th International Conference for Software Quality*, Ottawa - Canada.
- [9] J. Horkoff and E. Yu. A qualitative, interactive evaluation procedure for goal- and agent-oriented models. In *In Proceedings of CAiSE Forum 2009*, 2009.
- [10] P. Jackson. *Introduction To Expert Systems*. Addison Wesley, Harlow, England, 3rd edition, 1998.
- [11] F. Martensson. Trade-off examples inside software engineering and computer science. In L. Lundberg, M. Mattsson, and C. Wohlin, editors, *Software quality attributes and trade-offs*. Blekinge Institute of Technology, Karlskrona, Sweden, 2005.
- [12] Rafael Rocha. Inabit: A java rule engine and agent framework. <http://code.google.com/p/inabit/>, [September 2013].
- [13] W. N. Robinson, S. D. Pawlowski, and V. Volkov. Requirements interaction management. *ACM Comput. Surv.*, 35(2):132–190, June 2003.
- [14] R. Rocha, E. Costa, P. Brito, et al. Improving construction and maintenance of agent-based applications through an integration of shell and software framework approaches. In *XIX ENEA - Brazil*, page 12, 2012.
- [15] E. S.-K. Yu. *Modelling strategic relationships for process reengineering*. PhD thesis, Toronto, Ont., Canada, Canada, 1996. UMI Order No. GAXNN-02887 (Canadian dissertation).

# GreenRM: Reference Model for Sustainable Software Development

Marcello Thiry and Liliane Frez

Applied Computing Graduate Program  
University of Vale do Itajaí - UNIVALI  
Florianópolis, Brazil  
{thiry, lili.frez}@univali.br

Alessandra Zoucas

Department of Knowledge Management  
Federal University of Santa Catarina - UFSC  
Florianópolis, Brazil  
alessandra.zoucas@gmail.com

**Abstract**— Information technology and the environment have a relationship until then signaled by the effects caused by excessive energy consumption, greenhouse gas emissions and the e-waste. In our paper we present the GreenRM reference model for sustainable software development. We focus on the model structure definition and the benefits identified with its implementation. We also present a comparison between the model processes and attributes with the environmental requirements of ISO/IEC 14001 to demonstrate its compatibility. In addition we present a model early assessment in three Brazilian software organizations to evaluate the model technical and financial feasibility.

**Keywords:** *Green IT; Process Reference Model; Sustainable Development; Green Software Engineering.*

## I. INTRODUCTION

When we think of Information Technology (IT), words like innovation and solution comes to mind. However, rarely, or never, we think on relationship between IT and the environment. This relationship is discussed through green IT, which has emerged as a subject centralizing this discussion bringing to the IT market the importance of sustainable development and the growth focused on preservation of the environment. Environmental problems associated with the IT industry are mainly related to the large power consumption, the emission of carbon dioxide and lack of treatment of e-waste [1]. In software organizations we observed that environmental issues are the same, mainly due to technological resources used and the infrastructure required. In the software life cycle we can analyze issues related to the energy consumption during the execution of programs or the energy consumption to maintain the infrastructure of the software factory (computers, servers, and others), printing documents on requirements gathering phase or the use of disposable media (CD, DVD) on the installation phase [2], the great demand for acquisition of new equipment or electronic components with higher performance and the hardware change that become obsolete when the software evolves [3].

On this research we analyze organizational changes for a software organization implement processes that contribute to the improvement and reduce of environmental problems within their organizational setting. This paper presents the GreenRM reference model to support software organizations in the effort to achieve process improvement but considering environmental

quality with the introduction of environmental management. The GreenRM can stimulate competition in the software market through sustainable image, allowing organizations to be qualified and valued from an environmental quality model.

## II. THE GREENRM REFERENCE MODEL

GreenRM is a reference model based on the concepts of maturity and capability and was developed based on two pillars, the PRO2PI-MFMOD method framework [4][5] and the environmental standard ISO/IEC 14001 [6]. The PRO2PI-MFMOD framework was used to establish a reference model construction method for the GreenRM development. The ISO/IEC 14001 was used as a support to identify relevant processes and practices already consolidated.

During the PRO2PI-MFMOD framework instantiation, some activities and techniques were performed: 1) Gathering environmental needs in IT and definition of the target audience; 2) Literature Review; 3) Reference model structure definition (based on ISO/IEC 15504 [7]); 4) Detailed analysis of selected models; 5) Hold meetings with experts from the ISO/IEC 15504 to ensure structure compatibility between models; 6) Development strategy review; 7) A preliminary model version was evaluated with three software organizations. These activities were developed from the "Sequential Practices" described in the framework. The techniques employed in the development were: Translation process areas of an existing model; Application questionnaire; Literature Review; Analysis of related work and Peer Review. The PRO2PI-MFMOD framework instantiation allowed the development of a reference model based on definitions and concepts of recognized and consistent models in Software Engineering. With this, we expect a greater acceptance in software organizations that already have implemented maturity and capability models for software quality. The major benefits in implementing the GreenRM model are the integration of environmental management with the business goals, financial returns arising from green solutions like energy conservation, the sustainable image, changes on processes organizational for the environmental certification ISO/IEC 14001 and the change on culture and awareness.

### A. Green Levels

The green levels developed on GreenRM are the model maturity levels. Each maturity level has a set of required

processes. In addition, the processes were also classified in three categories: main processes, organizational processes and supporting processes.

The first green maturity level (G1 - Environmental sustainability is managed) defines four processes: GSM - Green Solution Management (main); DCM - Data Center Management (main); EWM - Electronic Waste Management (main); and ERM - Environmental Risk Management (supporting). The second green maturity level (G2 - Environmental sustainability is institutionalized) defines four processes: GAQ - Green Acquisition (organizational); GEI - Green Solution Evaluation and Improvement (supporting); GSD - Green Software Development (main); and GIR - Green Software Installation and Retirement (main). The third green maturity level (G3 - Environmental sustainability is continuously improved) defines three processes: GMS - Green Solution Measurement (organizational); OTG – Organizational Training for the Green Solution (organizational); EKM - Environmental Knowledge Management (organizational).

The green level G1 ensure an organized management of environmental initiatives performed by the organization. These initiatives are identified as green solutions and are treated at the first process of the level. By means of assuring the environmental management of specific solutions for the data processing center and for the electronic waste, the second and the third processes of the level are responsible for these issues. To help identifying these green solutions, the last process of the level permits the risk and environmental impacts analysis. The green level G2 ensure that the environmental processes be incorporated to the remained organizational processes. In this level, the auditing process is established to assure that the green solutions are been performed as in order to meet the organization environmental goals. Focusing on the accomplishment of the environmental requirements related to the software lifecycle, two processes are defined. These two processes focus on environmental practices directed related to the development, installation and the retiring phases of the software. Still at this maturity level, a process for green acquisition is also defined, demanding that supplier’s selection and evaluation be based on environmental criteria. The green maturity level G3 permits to analyze and measure the benefits gained with the application of green solutions through measurement and guarantees that the knowledge acquired with green solutions implementation be managed and disseminated.

**B. Capability Levels**

For the GreenRM capability dimension (model evaluation), three capability levels and eleven process attributes (PA) defined for the levels from 1 to 3 are presented. The ISO/IEC 15504 and ISO/IEC 14001 standards contributed significantly to the development of these process attributes. Most of these attributes is related to the environmental management needs specified at the ISO/IEC 14001.

The level 0 is not included in any type of indicator because it represents a non-implemented process or a faulty one in accomplishing the outcomes. The capability level 1 represents the process is managed: PA1 - the process achieves its defined outcomes; PA2-formal process documentation is established, maintained and communicated to everyone involved; PA3-the

process is planned and monitored; PA4-required roles and responsibilities to perform the process are identified; PA5-personnel performing the process are competent on the basis of appropriate education, training and experience; and PA6-the process is specified on the organization environmental policy. At capability level 2, it is defined that the process is institutionalized. The process attributes are then evaluated: PA7-a process improvement group is formally established to continuously discuss and revise the process; and PA8-the process outcomes are revised at different hierarchical levels of the organization, including higher level management. And finally, capability level 3 represents that the process is measured and evaluated continuously. To evaluate this level, the process attributes are demonstrated: PA9-achieved outcomes by the process performing allows demonstrating environmental benefits; PA10-process measurements are collected and analyzed in order to support the decision making about the environmental solutions; and PA11-improvement opportunities derived from technological innovations and environmental solutions are identified, evaluated and selected to support the business' goals achievement.

The process attributes PA2, PA4, PA6, PA8 e PA10 were identified from the analysis of the environmental requirements demanded by the ISO/IEC 14001 standard. By achieving these attributes, it is possible to guarantee the existence of the environmental policy and the formalization of processes in this policy, defining the roles and responsibilities involved, the analyses of administration in the decision making, as well as measurements for identifying corrective and preventive actions.

**C. A Process Documentation Excerpt**

As an example of the processes documentation, we present in Table I, the G2 maturity level process GSD (Green Software Development). This process allows the analysis of environmental practices in the software development lifecycle.

TABLE I. GSD (GREEN SOFTWARE DEVELOPMENT) PROCESS DESCRIPTION

<b>Process ID</b>	GSD
<b>Process Name</b>	Green Software Development
<b>Process Purpose</b>	Provide the environmental performance on the activities related to the software development phase. The improvement actions allow the insertion of environmental practices on the activities that belong to the software lifecycle.
<b>Process Outcomes</b>	As a result of the successful implementation of the software: <ol style="list-style-type: none"> <li>1. Software documentation is developed in electronic media and tools for traceability are established.</li> <li>2. The useful life of the software is analyzed at the requirements survey phase, considering future needs and current hardware limitations.</li> <li>3. Non-functional requirements aiming at meeting environmental needs are specified.</li> <li>4. The prototypes elaborated at the requirements survey phase are reused on the software implementation.</li> <li>5. Modularization practices are used in the software project, when necessary.</li> <li>6. The software components and methods are projected aiming their reutilization.</li> <li>7. Automation tools are used in the software implementation stage.</li> <li>8. Unit tests are performed in the end of each implementation.</li> <li>9. Automatized tests are planned and performed according to the frequency established in the planning.</li> </ol>

	10. Performance tests are planned and performed according to what was previously planned.
Base Practices	<p><b>GSD.BP1: Develop software documentation in electronic format.</b> Software documentation should be developed in electronic format, facilitating the control and organizing the artifacts and security of information. Besides, it decreases or avoids the need for paper use on the software development. [Outcome 1]</p> <p><b>GSD.BP2: Establish tools that allow traceability.</b> Tools that allow the traceability of artifacts generated during the software development should be applied, since it is through traceability that it is possible to measure the impact of changes and, consequently, to measure the effort required. The traceability facilitates the maintenance of artifacts and can decrease the need for printing documents. [Outcome 1]</p> <p><b>GSD.BP3: Analyze the useful life estimated for the software.</b> The software should be specified and developed taking into consideration the current and future characteristics of the hardware, because it is necessary to avoid the need for hardware replacement with the software implementation. [Outcome 2]</p> <p><b>GSD.BP4: Specify the non-functional requirements that aim at environmental issues.</b> Non-environmental requirements which describe characteristics that care for environmental quality of the software should be defined. For example: not using shiny colors, the energy consumption estimated for the software in use or not, and so on. [Outcome 3]</p> <p><b>GSD.BP5: Reuse the prototype developed at the requirements' phase on the software implementation.</b> The prototype should be reused on the software implementation and, with that, energy, time and effort spent on the requirements' phase are also reused. [Outcome 4]</p> <p><b>GSD.PB6: Use modularization practices in the software project.</b> Modularization simplifies the software project and may turn the implementation less complex and, consequently, less time will be spent to perform it. [Outcome 5]</p> <p><b>GSD.PB7: Project components and methods allowing reutilization.</b> Reutilization of the components and methods has several benefits, but one of them is energy and effort saving [Outcome 6]</p> <p><b>GSD.PB8: Use automation tools during software implementation.</b> Tools for automatic code generation and analysis allow the reduction of the time necessary to execute the task manually and the source code standardization. [Outcome 7]</p> <p><b>GSD.PB9: Perform unit tests.</b> Unitary tests may minimize the error detection in stages after the implementation and, therefore, perform a prominent role in the consolidation of a flawless software. The less flaws, the less necessity of alterations and rework. [Outcome 8]</p> <p><b>GSD.PB10: Plan automatized tests.</b> Automatized tests reduce the necessity of people in manual tests. Moreover, they allow the reapplication of test cases and standardization of test processes. It is necessary to plan the activities, resources and effort used in these tests. [Outcome 9]</p> <p><b>GSD.PB11: Plan performance tests.</b> Performance tests allow the identification of compatibility problems between software and hardware, so software may not work properly on current or legacy systems that present less capacity. It is necessary to plan the activities, resources and effort used in these tests. [Outcome 10]</p> <p><b>GSD.PB12: Execute automatized and performance tests.</b> Having the planning made, it is possible to perform the activities related to the automatized tests. It is necessary to register the test execution and inform the interested ones the problems detected by the test. [Outcome 9, 10]</p>

All expected process outcomes are related to environmental base practices. It is also possible to notice that these expected outcomes consider software quality aspects. Thus, we have concluded that environmental actions employed on the green software development process also contribute for software quality in general. This allows the organization to improve its development processes aimed at environmental quality and, at the same time, to improve the software itself. In this sense, we

have understood that environmental quality complements software quality.

When implementing the GreenRM in software organizations, this process is mandatory and also should be evaluated in relation to the process attributes defined in each capability level.

### III. THE ISO/IEC 14001 AND THE GREENRM

The Brazilian government has been prioritizing organizations that implement the environmental management system or any of the environmental requirements defined by the ISO/IEC 14001 standards. In some public notices for hiring software services - Petrobrás (Brazilian Oil Company) is a good example - it is possible to find items that demand the meeting of at least one requirement in the standard.

To support the ISO/IEC 14001 implementation in software organizations, the GreenRM has been developed to provide a path for this environmental certification. The standard describes the requirements that the environmental management system should meet; however, there is no script or guide that helps the application of these. The GreenRM supports the implementation of these requirements through the establishment of a model that presents in details what should be done and provides a complementary implementation guide that allows a more detailed understanding of each environmental process.

The ISO/IEC 14001 standard presents generic requirements, not considering specific aspects of information technology and software engineering. The GreenRM tries to fill this gap, establishing specific processes and practices to improve software development organizations. The conception of a model based on ISO/IEC 14001 qualifies and brings a strong basis to the model processes. Through Table II evaluation it was possible to realize that GreenRM meets the environmental requirements specified in the standard. However, it is necessary a previous evaluation of the organization before the certification because there are elements such as the documents content which were inserted only as work products in the GreenRM. This includes specifically the item '4.4.4 Documentation'.

### IV. GREENRM ASSESSMENT

To evaluate the model, an interview process to assess technical and financial feasibility of the GreenRM implementation has been done in three software organizations (Table III) with the participation of software engineering and environmental management experts.

TABLE II. SOFTWARE ORGANIZATIONS EVALUATED

A. Software development organization with 20 years of existence; 900 workers; MPS C (Brazilian Model equivalent to CMMI-DEV ML 3).
B. Software development organization with 34 years of existence; 700 workers; CMMI-DEV ML 2.
C. Software development organization with 10 years of existence; 200 workers; CMMI-DEV ML 2.

Concerning the technical feasibility, the three organizations considered the model a feasible one. However, the Green Acquisition and Green Solution Measurement processes have

been questioned by the organization A. The interviewee explained that the Green Acquisition process implementation would be problematic because of difficulties in finding and choosing suppliers from environmental criteria, taking into consideration that there might be a reduced number or even no organizations that put environmental actions into practice, and this would limit their choice of good suppliers. For the Green Solution Measurement, the interviewee has considered this a more difficult process to implement because, nowadays, difficulties already took place when measurements are done in the organization.

Regarding the financial feasibility, the model was considered feasible for the three organizations. Excepting for the Data Processing Center process evaluated as unfeasible by organization C, since it has its infrastructure outsourced. However, in situations like that, where an organization can justify the absence of a data center, the GreenRM model allows the exclusion of the DCM process. The evaluation process also identified how the results have been met for each process: N-Not Achieved; P-Partially Achieved; L-Largely Achieved; and F-Fully Achieved.

Through this collection, it has been identified the reaching level of the process in relation to the PA 1 attribute - in which the process reaches its defined outcomes. This was the only process attribute evaluated, since the evaluation aimed at identifying the processes adherence to the organizations reality. The evaluation of the PA 1 process has been considered as a first step, only to identify whether the organization executes a given result or not. Besides, it was not possible to evaluate the other attributes of the process, because the organizations did not implement the model.

To determine the process capability, the ISO/IEC 15504-3 evaluation method considers that each process attribute is assessed on a four-point (N-P-L-F) rating scale: Not achieved (0 - 15%); Partially achieved (>15% - 50%); Largely achieved (>50% - 85%); and Fully achieved (>85% - 100%). The rating is based upon evidence collected against the practice indicators, which demonstrate fulfillment of the process attribute.

Through this analysis, it has been identified that organization A and C have not obtained processes evaluated at the 'Fully achieved' level, while organization B has obtained two processes in this level. Regarding the 'Largely achieved' level, it has been identified that organization A has obtained two processes, organization B has obtained five processes and organization C has obtained one process in this level. For the 'Partially achieved' level, it has been identified that organization A and C have obtained two processes, while organization B has not obtained any process in this level. And, concerning the 'Not achieved' level, the organization A has obtained seven processes, organization B has obtained four processes and organization C has obtained eight processes in this level.

From the result of this evaluation, it was possible to compare the evaluated organizations. The organization B stands out from the others, which made possible to conclude that this organization will succeed best when implementing the model, taking into account that organization B holds more processes in the 'Fully achieved' and 'Largely achieved' levels

than the others. Therefore, on the comparative presented by the graph, the organization B is in the first position, organization A is in the second and organization C is in the third position.

In most cases, the actions applied by the organizations are motivated by their own interests, since most of the times there are no support from the Brazilian government in environmental projects for this area. Thus, we understand that investments in establishing projects such as the GreenRM will be possible when the government starts to grow interest and demand on the public hiring processes, providing the appreciation of software organizations that implement environmental actions.

## V. CONCLUSION

Our research demonstrates that GreenRM is a model that unites the concepts of Green IT in software development organizations with the ISO/IEC 14001 environmental requirements, according to the reality of these organizations. The GreenRM can be used as a guide for the environmental certification and for the application of Green IT practices. The main goal is not just support practices related to infrastructure but also improve the environmental sustainability in the software development process. The model has been evaluated in three software organizations obtaining supportive results. However, we realize that these results cannot be generalized. It is necessary to expand the evaluation considering a larger number of software organizations and also evaluate the model with a group of Green IT experts from Brazil and abroad.

We also are negotiating to work in a pilot for the model implementation in a software organization. The results of this implementation will allow a real verification of the GreenRM, including a complete review of the model. In addition we are also working with a metrics model to support the implemented practices measurement allowing to verify not just if the organizations are achieving the outcomes defined by the GreenRM, but if the implementation achieves real environmental benefits to this organizations.

## REFERENCES

- [1] S. Murugesan, "Harnessing Green IT: Principles and Practices," IEEE Computer Society, vol. 10, January/February 2008, pp. 24-33.
- [2] Johann, Timo; Dick, Markus; Kern, Eva; Naumann, Stefan: Sustainable development, sustainable software, and sustainable software engineering: An integrated approach. In: IEEE (eds.): SHUSER 2011. 2011 International Symposium on Humanities, Science and Engineering Research, June 6-7, 2011, Kuala Lumpur, Malaysia, 2011, pp. 34-39.
- [3] C. Sahin, F. Cayci, J. Clause, F. Kiamilev, L. Pollock, and K. Winbladh, "Towards Power Reduction Through Improved Software Design," Proceedings of IEEE EnergyTech, 2012.
- [4] C. F. Salviano, A. Zoucas, J. Silva, A. Alves, C. von Wangenheim and M. Thiry, "A Method Framework for Engineering Process Capability Models" Proc. 16th Conf. European Systems and Software Process Improvement and Innovation, Publizon, 2009, pp. 6.25-6.36.
- [5] C. von Wangenheim, J. Hauck, A. Zoucas, C. F. Salviano, F. McCaffery and F. Shull, "Creating Software Process Capability/Maturity Models," IEEE Software, July/August 2010.
- [6] ISO/IEC 14001:2004, "Environmental Management systems requirements for guidance: International Standards for Business, Government and Society". 2008.
- [7] ISO/IEC 15504 (Parts 1-5. 2003:2012), "Information technology: Process assessment". 2012.

# A Multicriteria Approach to Project Portfolio Selection

Using Multiobjective Optimization and Analytic Hierarchy Process

Everton Gomedede and Rodolfo M. Barros  
 Computer Science Department  
 State University of Londrina  
 Londrina, Brazil  
 evertongomedede@gmail.com, rodolfo@uel.br

**Abstract**—This paper presents an approach to solve project portfolio selection problem (PPSP) in the presence of limited resources, multiples criteria, software projects, constraints, functions to be optimized, interdependent projects, and scenarios with a large number of projects available. For this purpose, it is divided into two phases, one for (i) optimization using the multiobjective algorithm NSGA-II and another (ii) post-optimization using the Analytic Hierarchy Process (AHP). Among their contributions, we can name (i) a solution to the combinatorial analysis  $2^n$  and (ii) the structure of a hierarchy of criteria derived from subjective aspects.

**Keywords**- Software Project Portfolio Selection, Software Engineering Decision Support, NSGA-II;

## I. INTRODUCTION

Project Portfolio Management (PPM) is an important issue for organizations that want to establish a process of selection and prioritization of projects focused on alignment to the corporate strategies. This means managing the set of programs and/or projects as a systemic whole, and enables appropriate allocation of resources, whether financial, human and technological, enabling an integrated investment management [1].

The PPM includes activities such as identification, evaluation, selection, prioritization, balancing, among others, whereas support the consistency of the strategy aligned to the organizational vision, mission, and values [1]. This relation is shown in Fig. 1.

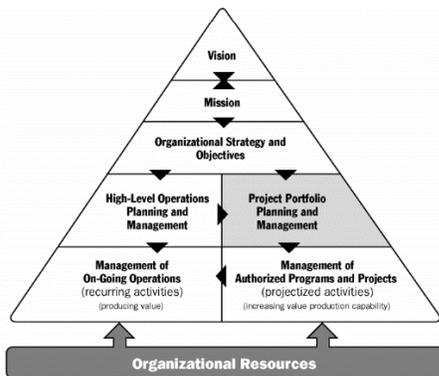


Figure 1. An organizational context of portfolio management [1].

<sup>1</sup> A Project Portfolio (PP) is a collection of programs and/or projects managed group with intention to achieve one or more strategic objectives [1].

The Project Portfolio Selection Problem (PPSP) aims to choose a set of programs and/or projects, considering not only the constraints and characteristics of each one, but also the relations among them, optimizing one or more objectives [21]. Considering the process shown in Fig. 2, the selection activity is contained in stage “Selection”, where the alternatives are evaluated and chosen [1].

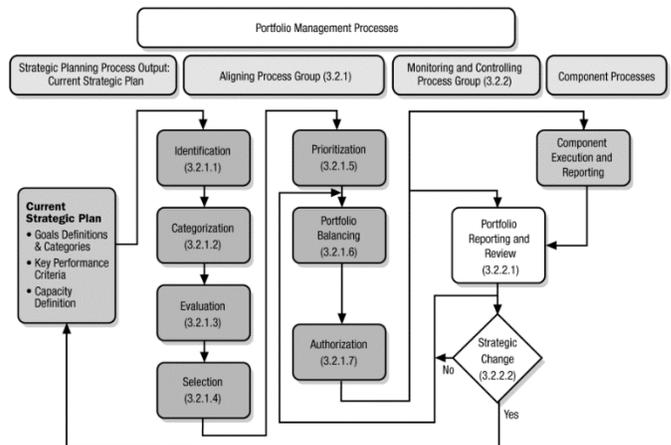


Figure 2. A process to systematic management of a project portfolio [1].

To select the projects that should be implemented by an organization is the main component of a PPM [2]. Levine [3] states that many companies strive to make their projects succeed well, not knowing, if these are the right projects to be executed. Another aspect addressed by the author is the fact that companies take excessive risks with projects or continue running projects that probably will not reach their goals. Thus, valuable resources are being expended without needing instead of being targeted for more interesting projects for organizations.

This paper presents an approach to solve PPSP consisting of two phases: (i) optimizes two objective functions (risk and return) subject to a set of constraints generating a set of optimal portfolios, and (ii) a hierarchy of qualitative criteria that captures difficult mapping mathematics information [11, 12] allowing a single project portfolio to be selected.

## II. RESEARCH PROBLEM & CONTRIBUTIONS

The research question that guided this work can be described as follows: “How to select an optimal project portfolio considering quantitative and qualitative criteria in a scenario with multiple objectives, multiple constraints, and a high number of possible combinations?”. This issue has generated some contributions that can be classified as:

- *Mathematical description of problem*: encode the problem in terms of their objective functions and constraints to solve combinatorial analysis  $2^n$ .
- *Hierarchy of criteria for project portfolio selection*: a structure with the criteria commonly present in the context of selection of project portfolios.

## III. FUNDAMENTALS & RELATED WORKS

### A. Multiobjective Optimization (MOO)

Problems with multiple objectives arise in a natural fashion in most disciplines and their solution has been a challenge to researchers for a long time. Despite the considerable variety of techniques developed in *Operations Research* (OR) and other disciplines to tackle these problems, the complexities of their solution calls for alternative approaches. [4].

The use of *Evolutionary Algorithms* (EAs) to solve problems of this nature has been motivated mainly because of the population-based nature of EAs which allows the generation of several elements of the Pareto optimal set in a single run. Additionally, the complexity of some *Multiobjective Optimization Problems* (MOPs) (e.g., very large search spaces, uncertainty, noise, disjoint Pareto curves, etc.) may prevent use (or application) of traditional OR MOP-solution techniques [4].

### B. Nondominated Sorting Genetic Algorithm II (NSGA-II)

The NSGA-II *Multiobjective Evolutionary Algorithm* is based on a classification of hierarchical dominance frontiers. This method is employed an elitist strategy of reinsertion in population, to ensure which any solution of the *Pareto Optimal Set* is found in any generation, it will be retained until the final population [5]. The NSGA-II works with a parent population  $P$  to generate an offspring population  $Q$  similar to conventional EAs. In the first iteration, generates a population  $P_0$  that is subjected to the classification of dominance. Each solution has a fitness value equal to the level of their frontier. Using the operators: *selection* by tournament, *crossover*, and *mutation*, obtained an offspring population  $Q_0$ .  $P_0$  and  $Q_0$  are size  $N$ . Both populations,  $P_0$  and  $Q_0$ , are join in a total population  $R_0 = 2N$ .

For subsequent generations ( $t = 1, 2, \dots$ ), the algorithm works with the *total population*  $R_t$ . Every generation is classified in hierarchical frontier of dominance, obtained frontiers  $F_1, F_2, \dots$ , where  $F_j$  is the first frontier, with all *non-dominated* solutions of the current  $R_t$ . The reintegration of the total population  $R_t$ , into a new population  $P_{t+1}$  of parents, is made in order to select the  $N$  solutions of  $R_t$  that are at a higher level of dominance. Thus, the formation of  $P_{t+1}$  starts with solutions  $F_1$  followed by solutions  $F_2$  and so forth.

Each  $F_i$  set must be inserted in  $P_{t+1}$  while  $P_{t+1} + |F_i| \leq N$ . Inserting the solutions of a frontier  $F_j$  such as  $|F_j| > N - P_{t+1}$ , the algorithm chooses the solutions of  $F_j$  that are better spread,

i.e., the reinsertion of population from one generation to another is made considering the *best individuals* among *parents* and *offspring*. These individuals are classified into dominance frontiers and new population is formed selecting individuals of the first frontier until population size is reached. This measurement is given by the crowd distance [5]. Fig. 3 illustrates the population scheme of NSGA-II.

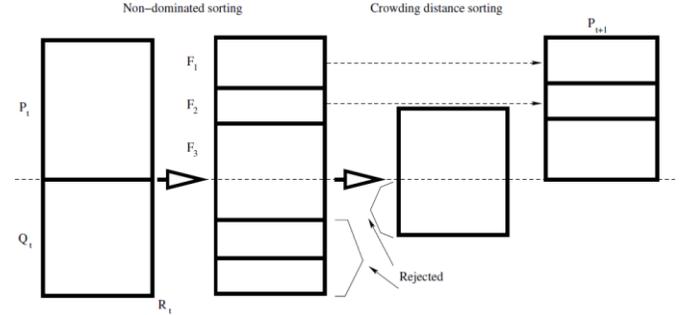


Figure 3. The population scheme of NSGA-II algorithm [5].

The NSGA-II presented a method of frontier diversity called crowding distance [5]. The distance of crowd of a solution  $i$  ( $d_i$ ), represents the estimate of the perimeter formed by the cuboid of which vertices are their nearest neighbors. The cuboid size  $i$  is directly proportional to distance of solution  $i$  of their neighbors. Extreme solutions in each objective will have a cuboid of infinite size. Fig. 4 shows the distance from the crowd for the solution  $i$ .

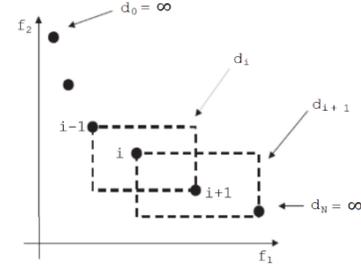


Figure 4. Crow distance used in NSGA-II [5].

Once obtained crowd distances, the sets  $F_j$  are ordered decreasingly by their distances. Finally, it generates  $|Q_{n+1}|$  from  $|P_{n+1}|$  using the operators of tournament selection by crowd, crossover, and mutation [5].

The multiobjective selection in NSGA-II is performed by tournament crowd. The NSGA-II incorporates a small modification in the tournament selection method (crowd tournament) [5]. A solution  $i$  is considered winner of a tournament against  $j$  solution if:

- Solution  $i$  has a higher level of non-dominance:  $i$  frontier  $< j$  frontier
- If both solutions are in the same frontier, but  $i$  crowd distance is greater than  $j$ , or  $d_i > d_j$

Subsequently, the operators of crossover and mutation are applied, as employees in EAs. At the end of each generation the population  $P_t$  and  $Q_t$  are inserted as previously described (Fig. 3) in an elitist strategy to obtain new parents population  $P_{t+1}$ . After reaching a pre-specified number of generations, the algorithm is

stopped and the frontier of non-dominated solutions of the current population is returned as the final solution of the EA.

### C. The Analytic Hierarchy Process (AHP)

The AHP was first proposed by Thomas L. Saaty and their main feature is the *pairwise comparison* of a hierarchy of criteria and alternatives [6]. It is often used to analyze problems of multicriteria decision-making [6, 7, 11, 12]. The AHP divides the overall problem in evaluations of minor importance, while maintaining the participation of these problems *small* in global decision, decomposing the structure of problem into a hierarchy (containing *criteria* and *alternatives*) [6].

Saaty states that hierarchy is an abstraction of a system structure to study the functional interactions of each components and their impact on the total system [6, 7]. The most creative part of the decisions that have *significant effect* on the result is the *modeling of the problem*. In AHP, a problem is structured as a hierarchy, and subsequently undergoes a process of comparison.

The *paired comparison* is an important component of the AHP. Two criteria (or alternatives) are compared using a nine-point scale, where one (1) means importance “*equal*”, three (3) the importance is “*low*”, five (5) clearly indicates “*superior*”, seven (7) is “*very*” important, and nine (9) denotes “*extremely*” important [6, 7]. Even numbers (2, 4, 6, 8) can be used to indicate intermediate values, if necessary. If there are  $n$  criteria to be considered, then  $n(n-1)/2$  pairwise comparisons should be made. Afterwards, an  $n \times n$  matrix is constructed and the weights of each entity (*local* and *global*) are obtained [11, 12].

The *consistency* of the matrix can be verified by the following indexes: *consistency index* (CI) and *consistency ratio* (CR). They are defined in equations (1) and (2) with  $\lambda_{max}$  is the main value (*Eigen*) and IR random consistency index, as shown in Tab. 1. For consistency, *CI* and *CR* should be less than 0.1 to AHP analysis be considered consistent [11, 12].

$$CI = (\lambda_{max} - n) / (n - 1) \quad (1)$$

$$CR = CI / RI \quad (2)$$

According to Saaty [7], the benefit of this method is that as the values of the paired comparisons judgments are based on experience, intuition, and on physical data, AHP can deal with *qualitative* and *quantitative* aspects of a decision problem.

TABLE I. CONSISTENCY INDEX RANDOM (RI)

$n^a$	1	2	3	4	5	6	7	8	9	10
RI <sup>b</sup>	0	0	0,58	0,9	1,12	1,24	1,32	1,41	1,45	1,49

a. Dimension of the matrix (n) and b. Random consistency index (RI).

A *well-constructed hierarchy* is a good model of reality, and can bring some advantages [6, 7]. First, the hierarchical representation of a system can be used to describe *how changes* in priorities at the highest levels *affect* the priority of the lowest levels. The hierarchy also allows obtaining an *overview* of a system, since lower levels of criteria to their purposes at the highest levels. Finally, hierarchical models are *flexible* and *stable*: stable because small changes are small effects; whereas

flexible because additions to a well-structured hierarchy does not disturb the overall performance [6, 7, 11, 12].

### D. Related Works

There are works in the literature that deal with project portfolio selection problem (PPSP). According to Wang [10] these works can be classified into:

- *Scoring Models*
- *Mathematical Models*
- *Financial Ratios Models*
- *Probabilistic Models*
- *Pricing Options Theory*
- *Strategic Approaches*
- *Hierarchical Approaches*
- *Behavioral Approaches*

In recent years, *Heuristic Methods* were used to solve PPSP. Iamratanakul [8] published a literature review related to this topic, classifying the portfolio selection models in a taxonomy that comprises different types of approaches, one of them is the *Heuristic Approach*. The evolution of published works that address the problem of portfolio selection can be seen in paper published by Metaxiotis & Liagkouras [9]. The Fig. 5 summarize the evolution of heuristics methods to portfolio selection and other subjects.

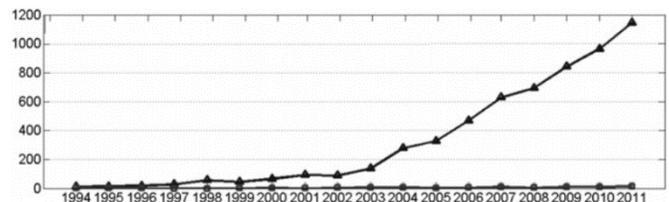


Figure 5. Evolution of publications related to heuristic methods [9].

The line with *circles* represents the evolution of publications related to heuristic methods for *portfolio selection*. The line with *triangles* represents the evolution of the publications of heuristic methods in *several contexts*.

Among recent works (2010-2014) we can mention; in [13] the authors propose an approach for *multiobjective heuristic* search technique to support a selection of project portfolio in scenarios with a large number of available projects. In [14] the authors propose an alternative that uses *fuzzy logic* with a heuristic to choose an optimal portfolio. The same authors present a variation of this alternative in [15] adding a data mining subsystem. The work presented in [16] uses an *evolutionary algorithm* for selecting an optimal portfolio based on a single objective function. In [17] the authors propose a tool that identifies a set of portfolios (*Pareto Optimal*) within a cost range allowing the realization of interactive analysis. In [18, 19], the authors present a tool which implements a heuristic algorithm. The Tab. 2 shows the comparison of related work with our approach.

TABLE II. RELATED WORKS COMPARATION

Criteria	Related Works						
	[13]	[14]	[15]	[16]	[17]	[18,19]	<i>Our</i>
<i>Objective Functon</i>	●	○	○	○	●	●	●
<i>Restrictions Set</i>	●			●	●	●	●
<i>Heuristic Search</i>	●	●	●	●	●	●	●
<i>Optimal Set</i>	●	○	○	○		●	●
<i>Structurated Decision</i>							●
<i>Post Optimization Selection</i>	●						●
<i>Portfolio Selection</i>	●	●	●	●	●	●	●
<i>Criteria Set</i>							●
<i>Process in Phases</i>							●
<i>NP-hard</i>	●			●	●	●	●

● Meets strongly ○ partially meets and no symbol, no answer. Criteria developed according to elements often present in publications [8, 9, 13, 14, 15, 16, 17, 18, 19].

The work presented in this article is intended to cover the gaps between the current models.

#### IV. TWO PHASES MULTICRITERIA APPROACH

##### A. The Project Portfolio Selection Problem (PPSP)

PPSP is to determine in what ways the available designs can be combined to *maximize* the return, considering a set of *constraints* while *minimizing* the risks involved [21].

Harry Markowitz<sup>2</sup> [20] defines *two fundamental* characteristics of a portfolio: their expected *return* and their variance, representing the *risk* of the portfolio. PPSP can be formally defined as:

$$\text{Maximize } \bar{R}_p = \sum_{i=0}^n X_i \cdot E(R)_i \quad (3)$$

$$\text{Minimize } \sigma_p = \sqrt{\sum_{i=0}^n w_i^2 \sigma_i^2 + \sum_{i=0}^n w_i w_j \rho_{i,j} \sigma_i \sigma_j} \quad (4)$$

$$\text{Subject } \sum_{i=0}^n C I_i \leq \text{budget} \quad (5)$$

$$\max(\text{payback}) \leq \text{payback} \quad (6)$$

$$P_i \text{ depends } P[1..n] \quad (7)$$

$$P_i \text{ exclude } P[1..n] \quad (8)$$

$$P_i \exists! E(R) \quad (9)$$

$$E(R) > 0 \quad (10)$$

##### B. Function Return

Equation (3) represents the objective function to be maximized [20, 21]. The first feature of the portfolio, its return expected  $\bar{R}_p$ , is simply the weighted average of the returns of individual projects that comprise it, where:

- $X_i$  is the percentage invested in the project  $i$ .
- $E(R)_i$  is the percentage invested in the project  $i$ .

##### C. Risk Function

Equation (4) represents the function that must be minimized [20, 21]. The key feature of this equation is the risk, as measured by their variance, where:

- $W_x W_y$  represent, respectively, the share of  $x$  and  $y$  in the project portfolio
- $\sigma_x^2 \sigma_y^2$  represent the variance of  $x$  and  $y$  projects, respectively, with respect to the risks identified
- $\rho_{x,y}$  covariance between  $x$  and  $y$  projects

The covariance  $\rho_{x,y}$  is given by Person Covariance [15] in Equation (11).

$$\rho_{x,y} = \frac{n \sum x_i y_i - (\sum x_i)(\sum y_i)}{\sqrt{[n \sum x_i^2 - (\sum x_i)^2] * [n \sum y_i^2 - (\sum y_i)^2]}} \quad (11)$$

##### D. Constraints

Equations (5) and (6) ensure that the *total of investment* on portfolio and the *return period* (payback) are not higher than expected [21]. Equations (7) and (8) show the *dependencies* and *exclusionary* between projects. This means that when selecting a project, you must also select their dependents and/or eliminate mutually exclusive [21]. In (9), it ensures that a project should exist only once within the portfolio. Finally, Equation (10) ensures that the final return is greater than zero [21]. The search space is given by  $2^n$ , where  $n$  is the number of projects available for selection<sup>3</sup>.

##### E. Structure of Two Phase Approach

Considering the different stages of the project portfolio selection process [21], a decision structure of two phases was created (Fig. 6), aiming (i) to generate a *set of optimal solutions* and (ii) allowing that *one of the solutions* is selected by a structured method.

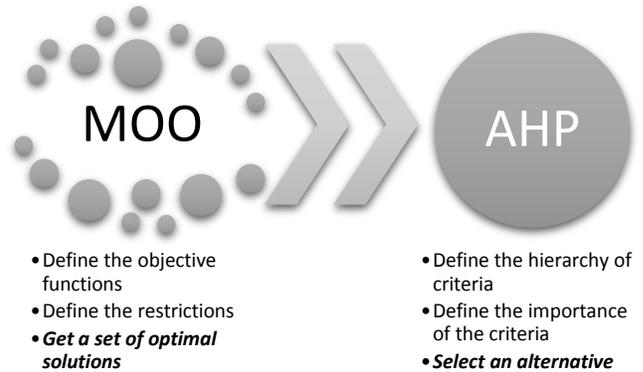


Figure 6. Structure of the two-phase approach.

The first phase (MOO) uses the NSGA-II algorithm with the objective functions and constraints explained above to generate a set of optimal solutions (Pareto Front). After that, a hierarchical structure is used to select a single solution (AHP) considering a set of criteria.

##### F. Hierarchy of Criteria

A total of 34 criteria to assess the values of project portfolios in the relevant literature were researched. These criteria were

<sup>2</sup> Was awarded with Prize in Economic Sciences in Memory of Alfred Nobel 1990.

<sup>3</sup> This search space disregards the restrictions imposed on the problem. This is one of the types of problems that can be classified as *NP-hard* [21], polynomial methods which would take considerable time to test all solutions.

organized into two groups: (i) *endogenous* criteria with focus on creating internal value to the organization (these criteria should express factors that are within the control of the organization), (ii) *exogenous* criteria intrinsically related with environment being beyond control of the organization [21]. This classification (and criteria) can be seen in Tab. 3.

TABLE III. SELECTION CRITERIA PROJECT PORTFOLIO

<i>Endogenous</i>	<i>Exogenous</i>
Fit with corporate strategic objectives	<b>Position of the related technology in its own life cycle</b>
Profitability	<b>Environmental and safety consideration</b>
Capability of research team	<b>Dealing with international Sanctions</b>
Financing capacity	<b>Public support for development</b>
Impact on enhancing Innovation	<b>Barriers to copy or imitation</b>
Contents of technical plan	<b>Market volume opened by Research result</b>
Serving as infrastructure	<b>Competition intensity</b>
Technological connections	<b>Benefits for human life</b>
Extensibility of results and Span of application	<b>Impact on firm prestige</b>
Appropriateness for research cost	Potential for progress
Equipment support	Market Dynamics
Appropriateness for research project timing	Potential for research product growth
Impact on enhancing Firm Productivity	Impact on societal stakes
Advancement of related Technology	Number of stakeholders
Research gap to corporate core business	Impact of related technology on competitive issues
Quality Improvement	
Impact on employees learning and growth	
Experience accumulated in the field	
Synergy with other projects	

The criteria *bold* has a higher representation [21].

The criteria of Tab. 3 are used *often* in specialized literature [21], providing a structure to allow for a meaningful analysis. An example of the result of the hierarchy can be seen in Fig. 7.

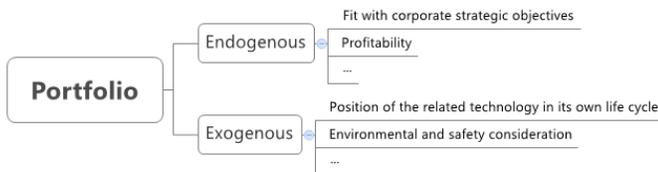


Figure 7. Example of hierarchy of criteria.

## V. EXPERIMENTS & RESULTS

### A. Considerations and Initial Parameters

In PPSP, each portfolio is represented by a *binary sequence*, where each position of the sequence is that the project is present or not. An individual may be represents by  $S = [0, 1, 1, 0, 1]$ . To ensure a good quality and diversity of the initial solution set, the initial population of NSGA-II was generated *random way* [5].

The initial population size was estimated empirically, starting with 10 individuals, increasing by 10 until the result of the algorithm was not changed reaching number (rounded) of 100 individuals. This size remains the same for population during the iterations.

The stopping criterion of the genetic algorithm might vary according to user's choice. One way is to (i) define a number of generations that must be created. Another way is (ii) run it until it is a population where individuals have the evaluation function to be reached [5]. The criterion used was (iii) *convergence*, i.e., there is no significant improvement in the solution for a given number of generations. The Tab. 4 summarizes these parameters.

TABLE IV. PARAMETERS USED IN NSGA-II ALGORITHM

<i>Parameter</i>	<i>Value</i>
Amount of Projects	42 software projects
Initial Population	100 individuals
Population Size	100
Function to be Maximized	Equation (3)
Function to be Minimized	Equation (4)
Constraints	Equations (5), (6), (7), (8), (9) and (10)
Representations Scheme	Binary Encoding
Selection Operator	Crowding Tournament
Crossover Operator	Partially Mapped Crossover (PMX)
Mutation Rate	0.01
Stop Condition	100 springs without improve over solutions

### B. Results and Analysis of Experiments

The data used in experiments was got from a set of 42 software projects originated from strategic planning of a midsize company. These experiments were carried out considering the *real* scenario and scenario *simulations*, where we tested variations in values of constraints.

The Pareto optimal set is displayed in Fig. 8. This set contains portfolios that were constructed by the combination of projects available, optimizing the functions represented by Equations (3) and (4) and respecting constraints expressed by equations (5), (6), (7), (8), (9) and (10). The solutions contained in the set can be considered optimum (from the viewpoint of each objective function). Variations on the constraints (Equations (5) and (6)) alter the set of solutions (i) reducing the amount of available portfolios and (ii) reducing the efficient frontier (solutions of lower *return* and lower *risk*), confirming the theory of Markowitz [20].

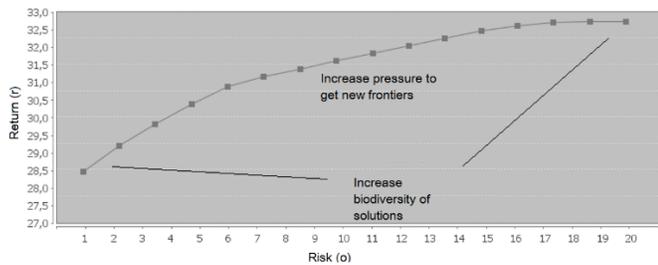


Figure 8. Efficient frontier generated by MOO using NSGA-II.

The Fig. 8 shows two adjacent goals of heuristic search (i) search for *new frontiers* and (ii) increase the *biodiversity* of the solutions in order to obtain a greater number of alternatives available for post-optimization step [5].

With a set of optimal solutions, one can choose among them. This step is important because you can use the *tacit knowledge*, *experience*, and *intuition* of experts [6, 11, 12]. For experiment, we used the criteria of Tab. 3 with 7 people (including functional managers, project managers, and directors) and the paired comparison obtained by *Delphi* method [11].

The result was the selection of a *portfolio* contained in the Pareto optimal set, with a tendency for higher risk/return and being accepted, by those involved, as the best portfolio.

## VI. CONCLUSIONS & FURTHER WORK

The *models* and *algorithms* [5] discussed here aimed to bring an *approach* to organizational decision-making and give a new dimension to the *project portfolio selection*. This approach has better results when there are *several constraints* to be satisfied and/or the problem is large (several projects available) to be solved in deterministic or polynomial way (*NP-hard*) [21].

Importantly, the *selection* of a project portfolio assumes a *broader* and more *complex* understanding than the single use of a particular method [11, 12, 22, 23]. It presupposes that the decision on a portfolio is the result of *negotiation*, *human aspects* and *strategic* analysis. The approach of this work encourages and guides decision-making, but should not be used as the sole method.

Among the contributions of the approach can be mentioned (i) a solution to the *combinatorial analysis*  $2^n$  and (ii) the structure of a *hierarchy of criteria* derived from subjective aspects that allow the selection of a single portfolio.

In future work, one can explore a *larger set of constraints* and *other functions to be optimized*, such as minimizing the costs of the portfolios generated by transforming PPSP from 2 to 3 or  $N$  goals, added data sources, such as Master Data Management (MDM) [22] to get more precision information.

## REFERENCES

- [1] PMI (2013). The Standard for Portfolio Management / Project Management Institute. — 3rd ed. Project Management Institute, Inc.
- [2] Killen, C. (2007). Managing the new product development project portfolio: a review of the literature and empirical evidence. *Management of Engineering and Technology*, pages 5–9.
- [3] Levine, H. A. (2007). *Project Portfolio Management: A practical guide to selecting projects, managing portfolios and maximizing benefits*. Jossey-Bass, San Francisco, 1st edition.
- [4] Carlos A. Coello Coello, Gary B. Lamont, and David A. Van Veldhuizen, *Evolutionary Algorithms for Solving Multi-Objective Problems (Genetic and Evolutionary Computation) Second Edition* - Springer-Verlag New York, Inc. Secaucus, NJ, USA ©2006.
- [5] Deb, K., & Pratap, A. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2), 182–197.
- [6] Saaty, T. L. (1980). *The Analytic Hierarchy Process*. New York: McGraw-Hill International.
- [7] Saaty, T. L. (2005). *Theory and Applications of the Analytic Network Process: Decision Making with Benefits, Opportunities, Costs, and Risks*. Pittsburgh: RWS Publications.
- [8] Iamratanakul, S. (2008). Project portfolio selection: From past to present. *Management of Innovation and Technology*, 287–292.
- [9] Metaxiotis, K., & Liagkouras, K. (2012). Multiobjective Evolutionary Algorithms for Portfolio Management: A comprehensive literature review. *Expert Systems with Applications*, 39(14), 11685–11698. doi:10.1016/j.eswa.2012.04.053
- [10] Wang, Z., & Yu, Y. (2011). Information entropy method for project portfolio selection. *2011 Eighth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, 2618–2622.
- [11] Gomedes, E., Barros, R. M. (2012) “Utilizando o Método Analytic Hierarchy Process (AHP) para Priorização de Serviços de TI: Um Estudo de Caso.” In: VIII Simpósio Brasileiro de Sistemas de Informação, São Paulo, p. 408-419, 2012.
- [12] Gomedes, E., Proenca JR., M. L. and Barros, R. M. (2012) “Networks Baselines And Analytic Hierarchy Process: An Approach To Strategic Decisions.” In: IADIS International Conference Applied Computing, p. 34-41, 2012.
- [13] Barros, M., & Costa, H. (2012). Multiobjective optimization for project portfolio selection. *GECCO Companion '12 Proceedings of the Fourteenth International Conference on Genetic and Evolutionary Computation Conference Companion*, 1541.
- [14] Danmei, Z., & Tie, Z. (2008). A novel R&D project portfolio selection decision approach based on fuzzy logic and heuristics scheduling. *Control and Decision Conference, 2008. CCDC 2008. Chinese*, 144–147.
- [15] Danmei, Z., Xingtong, W., & Rongrong, R. (2010). A Heuristics R and D Projects Portfolio Selection Decision System Based on Data Mining and Fuzzy Logic. *2010 International Conference on Intelligent Computation Technology and Automation*, 118–121.
- [16] Kremmel, T., Kubalik, J., & Biffi, S. (2010). Multiobjective evolutionary algorithm for software project portfolio optimization. *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*.
- [17] Lourenço, J., Morton, A., & Costa, C. B. e. (2012). PROBE—A multicriteria decision support system for portfolio robustness evaluation. *Decision Support Systems*, 4668.
- [18] Mira, C., Feijao, P., & Souza, M. (2013). A project portfolio selection decision support system. *10th International Conference on Service Systems and Service Management (ICSSSM)*, 725 – 730.
- [19] Mira, C., Feijao, P., Souza, M. A., Moura, A., Meidanis, J., Lima, G., ... Freitas, I. T. (2012). A GRASP-based Heuristic for the Project Portfolio Selection Problem. *2012 IEEE 15th International Conference on Computational Science and Engineering*, 36–41.
- [20] Markowitz, H., 1952. Portfolio selection. *Journal of Finance* 7, 77–91.
- [21] Abbassi, M., Ashrafi, M., & Sharifi Tashnizi, E. (2014). Selecting balanced portfolios of R&D projects with interdependencies: A Cross-Entropy based methodology. *Technovation*, 34(1), 54–63.
- [22] Gomedes, E. ; Barros, R. M . Master Data Management e Data Warehouse - Uma Abordagem Arquitetural para a Melhoria do Processo de Decisão. In: 8ª Conferência Ibérica de Sistemas e Tecnologias de Informação - CISTI'2013, 2013, Lisboa. 8ª Conferência Ibérica de Sistemas e Tecnologias de Informação - CISTI'2013, 2013.
- [23] Gomedes, E. ; Barros, R. M . A Non Intrusive Process to Software Engineering Decision Support focused on increasing the Quality of Software Development. In: The 25th International Conference on Software Engineering and Knowledge Engineering, 2013, Boston - USA. The 25th International Conference on Software Engineering and Knowledge Engineering, 2013.

# Design and Development of a Mobile Classroom Response System for Interactive Problem Solving\*

M. Muztaba Fuad

Department of Computer Science  
Winston-Salem State University  
Winston-Salem, NC 27110, USA  
fuadmo@wssu.edu

Debzani Deb

Department of Computer Science  
Winston-Salem State University  
Winston-Salem, NC 27110, USA  
debd@wssu.edu

**Abstract**— It is common for students to multi-task and use their mobile devices while in class for studying or other activities. This research aims to leverage this situation by developing a mobile classroom learning software to help students solve interactive problems in their mobile devices in order to improve their class engagement and problem solving skills. This paper presents the design and development of the mobile classroom response software to communicate, collaborate and evaluate in-class interactive problem solving activities using mobile devices. The software facilitates various pedagogical approaches that reported to enhance student learning. The software is designed to be easy-to-use and maintainable. MRS is extensible and can render interactive problems developed by third party developer. Software quality matrices for the developed code and the user interface are presented to justify above stated objectives.

**Keywords**—*Mobile Learning Software; Software Architecture, Client-server System, Extensibility.*

## I. INTRODUCTION

Mobile technology has brought tremendous potential and opportunities for educators to enable and deliver learning in ways that could not have been accomplished before. There have been an increasing number of studies related to the research and development of learning software intended for mobile computing devices. Some of these studies focus on supporting several interesting pedagogical approaches while utilizing mobile learning software such as slide annotation, collaborative note taking, student submissions, grading, polling etc. This research envisions utilizing mobile learning environment to help students solve interactive problems in order to improve their class engagement and problem solving skills. Toward this goal, this paper present the design, development and planned incorporation of the mobile classroom response software (MRS) designed to communicate, collaborate and evaluate in-class interactive problem solving activities using mobile devices.

MRS is a client-server software that allows the faculty to dynamically prompt the students with interactive problems synchronized with the lecture material in their mobile computing devices (Clients). Students are able to actively interact with the problem and send their answer back to the faculty computer (Server). MRS then performs grading of the exercises automatically, by comparing the student made

sequence of steps with the correct sequence of steps. The other important features supported by MRS are immediate and context-sensitive feedback, anonymous question, polling, summarized grading etc. Currently MRS software and associated problem solving activities are being deployed in CS and IT courses at Winston-Salem state University (WSSU). By adopting MRS in the classroom, this research expects to increase student engagement and improve their problem solving capabilities and therefore prepare them better to enter the computing workforce. MRS is designed to be user friendly and extensible so that faculty and students can use the system with ease and can extend it to any domain.

## II. BACKGROUND AND RELATED RESEARCH

To improve student learning in the STEM disciplines, traditional pedagogical approaches are not enough to transfer critical knowledge to students. A feedback driven evidence based teaching and learning technique has to be devised and implemented to improve student retention rates in the STEM discipline. In that regard, evidence-based instructional practices were incorporated in a sophomore-level CS course for the last few offerings, where at the end of a lecture, students were immediately prompted with in-class problem solving activities. Intervened student's performance data and their response to this pedagogy reveal the potential of this approach in order to enhance student learning. This finding encourages us to extend this model by scattering the questions/problems throughout the class period by synchronizing with the content covered. Observation about student's frequent use of mobile devices during class further inspires us to extend the above idea of asynchronous problem solving to mobile devices to engage students more to class activities.

Different studies [1]-[2] have found the benefits of using mobile devices in classrooms and in recent years, there has been a plethora of work [3]-[4] etc. performed to incorporate mobile devices in classrooms. There are also several commercial products [5]-[6] etc. for different mobile platforms that provide similar functionalities. Classroom Presenter [7], Ubiquitous Presenter [8] and DyKnow [9] are notable research initiatives that utilize tablets to create more active, student-centered lecture environment while supporting various effective pedagogies. However, there are distinct differences

---

\* Supported by NSF fund # 1332531

between the MRS and similar systems. Most importantly, MRS facilitates interactive problem solving. In STEM courses, students need to actively solve problems by interacting with the problem in a hands-on approach. Students cannot develop skills such as synthesizing a problem and critical thinking only by using multiple-choice or true-false questions. We argue that by presenting the problems as interactive entities, where students can actively play with the problem; student’s critical thinking and problem solving skills can be improved.

### A. Interactive Problem

An interactive problem is one, where students have to devise the answer following a set of steps and by following a particular algorithm/process. In each step, students have to make key choices that will have impact on the next step of the interaction. During these interaction steps, students can go back and forth and change their answer. This will allow them to see what is the affect of different selection on the result and how every piece fits together. Problems can be started bottom up or at the middle to give students different perspective on the problem and assess their problem solving skills. Only after the student traverse each of the steps or the allotted time to answer a problem runs out, the results of their interactions performed at each step are then sent back to the server as the answer. Each problem has a rubric that not only grade final answer but also partial answers to gauge student’s problem solving skills and thinking models.

## III. SYSTEM DESIGN

The MRS software is designed as a client-server application. Client device needs the corresponding client app installed in them to interact during the class. The server (or the faculty computer) hosts the questions, manage users and process the results for display and for grade calculation purposes. It also has the required data analysis component to tabulate user responses and produce easy to interpret reports.

### A. Server

Figure 1 shows the lifecycle of the server. The server is designed as a multi-process, multi-threaded entity to satisfy simultaneous invocation from users and to provide real time response to in-class activities. Usually when a user initiates a check-in to the system, the server validates the identity of the user and set a role for that specific user. Roles are basically privilege levels that allow a user to interact with the system with certain accessibility. The client app also allow students to send anonymous feedback/questions to faculty and vote on questions that faculty will choose to review at the end of the class. Separate thread of the server constantly monitors whether students wants to initiate a feedback/question session. In that case, it sends the current pool of feedback/ questions to the corresponding client. Once a response from the client is received, it is matched with the current pool and if a match is found, the priority of that entry is increased. Otherwise a new entry is created with the newly initiated feedback/question. Once the server receives answers back from all the clients, it does corresponding analysis of the data and produce

appropriate summarized representation as specified by the faculty. The server also maintains the score of every student, which can be used later to calculate student’s grade.

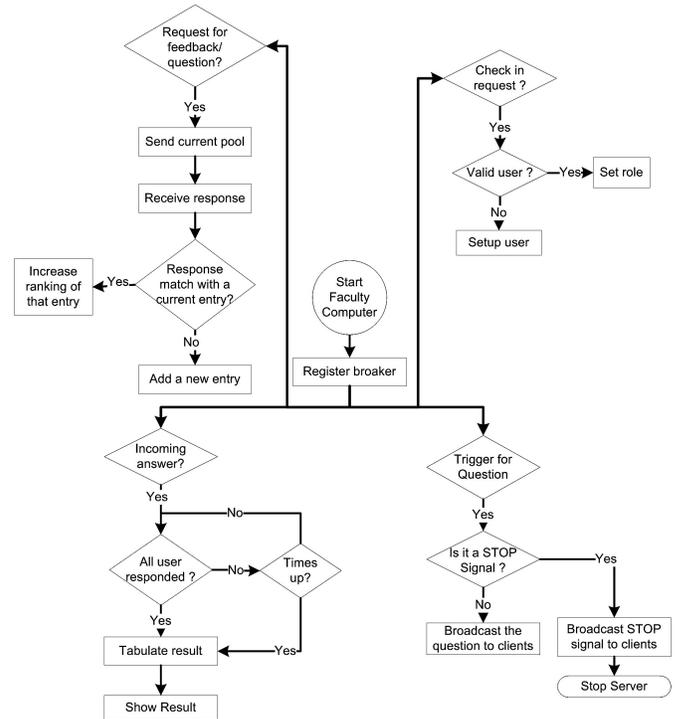


Figure 1. Server lifecycle.

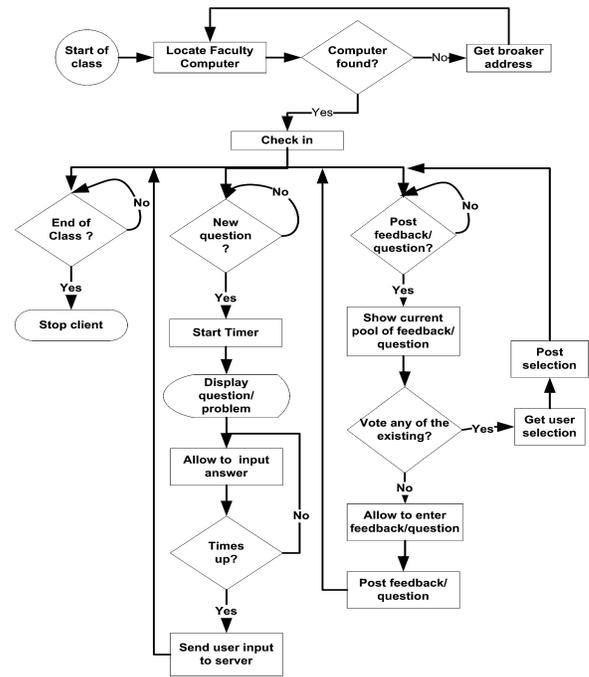


Figure 2. Client lifecycle.

### B. Clients

Figure 2 shows the life cycle of the client. Once a student checks in, the app shows a standard screen where students can only submit new feedback/question or vote on one. If the app receives a new question from the server, the corresponding

activity that will render the given question into interactive entity will be executed. Every problem has a set amount of time to answer (which the faculty can assign) and a visible timer starts counting down to allow the students to see how much time they have left to answer the question. Once the answer of the student is received (or the timer runs out) and sent to the server, the app goes back to its root screen. At any time during the class, students can initiate a session to post a feedback/question or vote on an existing one anonymously.

#### IV. SYSTEM DEVELOPMENT

The current implementation of the server is in Java and the client is on Android. The server's components are:

- User interface: To manage and monitor the system.
- Encoder and Decoder: To store, transmit and convert the question and answer in the server and to client applications. A lightweight XML format is developed for encoding.
- Result processing: Data processing capabilities built into this module to process answers back from the clients and to produce proper representation.
- Blackboard integration: To import student information and export student score to course management system.
- Trigger management: To synchronize question broadcasting and answer propagation and timing management.
- Meta-language: A Meta language is developed to describe the questions and to render them in a general way across different platforms. It is XML based and has similarity with the encoding scheme mentioned above.
- Question editor: To easily add, delete and edit questions.
- User management: To manage user roles and information.

The client app has platform specific components and also the following:

- User Interface: To allow students to interact with the given problem and to initiate/vote anonymous questions/feedback.
- Networking module: This module is responsible for transmission and reception of questions, their answers and student feedback/questions and any other server messages. To properly communicate with the server, an asynchronous communication and synchronization protocol is developed to satisfy timely execution of problems and propagation of answers back to the server.
- Render module: This module render the encoded question sent from the server in the client device. This module determines which Android activity should be initiated for a question, initialize the activity with proper parameters as sent from the server and keep in consideration the target screen dimension, resolution and orientation for rendering a question.
- Log-in module: This component is responsible for discovering the server, checking user information and setting appropriate role of the user. This module allows the client to work per session basis and will automatically logs out, if it receives a stop signal from the server.

##### A. Extensibility

Building each possible problem type within the client is impossible and not practical. This will not only make the client overly bulky to run in mobile platform, but will also

make it domain centric. To overcome this, the activity (or the android app) that facilitates interactive problems is completely separated from the application logic of the MRS client software. Figure 3 shows the life cycle of such interactive problem app. This approach makes it possible to integrate and execute any interactive problems developed by third party developers into the MRS software. The renderer module of MRS software can run any android activity (and app) as long as following conditions are met:

- Name of the package for the target app should be same as the question type in server.
- The incoming question will be delivered to the app through an Android Intent using the encoding scheme. Therefore the developer should use the decoding method in the project library to work with the given question.
- Answers from the interactive app should be returned to MRS using an Android Intent and using the encoding method in the project library.

Constrained with above conditions, anyone can develop their own interactive problem apps in any domain and use the MRS software to incorporate interactive problem solving in class. Third party developer can also decide on the level of interactivity they want to incorporate in their apps and the way to tackle Android's activity life cycle for their app.

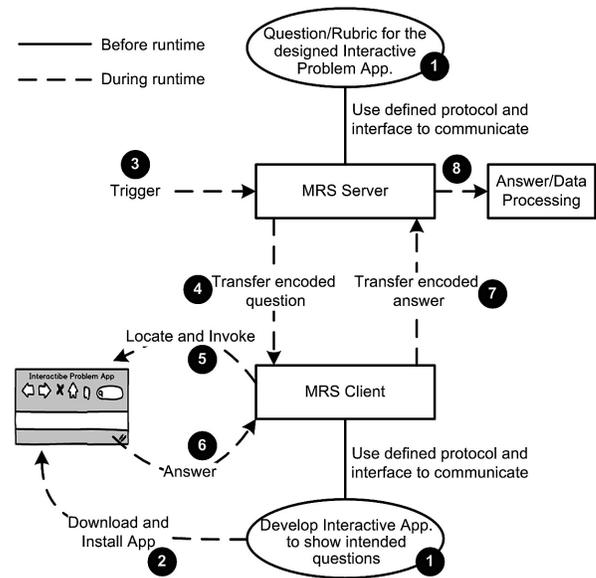


Figure 3. Lifecycle of interactive problem app.

##### B. Communication

A prefix-based communication scheme is devised for all messages, where different parts of the message are separated by a dedicated delimiter symbol. Any acknowledgement is piggy-backed with an outgoing message to minimize transmission overhead. The prefixes distinguish each message and the server or the client process an incoming messages according to the prefix. The size of the prefixes and the delimiter symbol are kept small in order to reduce the transmission overhead. Underneath the custom transport layer protocol, UDP is used to transfer packets between clients and

server. Since each of the messages occupied single packets, there is no need to sequence them, saving us the overhead for sequencing and blocking communication.

### C. Software Matrices

To examine the quality and maintainability of the code, we use a static code analysis tool named STAN [10]. STAN analyzes the structure of the code and visualizes the design, to measure quality of the code and to report design flaws. STAN supports a set of selected metrics, suitable to cover the most important aspects of structural quality. Below we present two such measures that show the overall quality of the code.

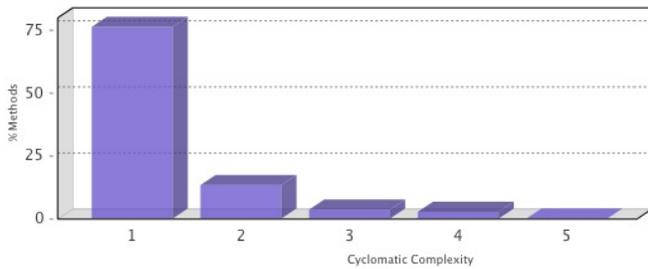


Figure 4. McCabe Cyclomatic Complexity.

Figure 4 shows McCabe Cyclomatic Complexity [11] of the code. We used this measure as it can be accurately calculated from the static call graph and determines code's structural complexity. Studies [12]-[13] show a correlation between a program's Cyclomatic Complexity, where code with higher Cyclomatic Complexity have higher probability of inducing errors during later maintenance. In that regard, it is evident from Figure 4 that the Cyclomatic Complexity values for MRS falls within the low risk and simple program range [14] and therefore the software is easy to maintain and extend. We also evaluated the code using object-oriented metrics found in [15]. Table I shows the values calculated by STAN for the developed code. The value for each of the matrices falls within desired ranges, which justify our claims regarding extendibility, manageability and maintainability.

TABLE I. QUALITY MATRICES.

Chidamber & Kemerer Metrics	Average
Weighted Method Per Class (WMC)	9.2
Depth of Inheritance Tree (DIT)	2
Number of Children (NOC)	1
Coupling between Objects (COB)	4
Response for a Class (RFC)	6.87
Lack of Cohesion in Methods (LCOM)	7.93

### V. MRS AT WORK

One of the goals of the system is to provide users with easy to use interfaces and seamless user experience. In that regard, all user interfaces were made intuitive and easy to use. Before the start of the class, faculty has to setup in-class interaction (importing student info, questions, their answers/rubrics and setup type of analysis, kind of report etc.) in the faculty computer using a graphical user interface. Once the interaction for the class is setup, students logged into the system and the class is in progress; the faculty can use a

trigger (mouse click or a designated key in the keyboard) to broadcast a question to students. On the other hand, when a student initiates a check-in to the system, the clients first locates the faculty machine (MRS server) and once a session is established, credentials are validated so that the client can start accepting questions. After a successful login, the client shows the root screen, where the students can post anonymous questions/feedback to the faculty. Once a question is received the client invokes the corresponding activity to render the given question as described in Section IV.A. As mentioned earlier, each question has multiple interactive screens, which students can traverse back and fourth. Once the faculty machine receives answers back from all the clients, it does corresponding analysis of the data and displays it accordingly. Currently, several interactive problems are being developed with a planned deployment in the class of Fall, 2014. Currently MRS is operational and it's scalability, responsiveness and reliability parameters have been tested and they all fell within accepted ranges.

### VI. CONCLUSIONS

In this paper, we present the design and development of MRS software that is targeted towards supporting classroom interaction using mobile devices. More specifically the goal is to facilitate interactive problem solving, submission, grading polling etc. by using the software. The architecture is presented and elaborated to exemplify the communication, maintainability and extendibility aspects. The result acquired from software quality data and the user interface shows that the system is easy-to-use, extendible and maintainable.

### REFERENCES

- [1] Mockus, L. and Edel-Malizia, S., "The Impact of Mobile Access on Motivation: Distance Education Student Perceptions", 17th Annual Sloan Consortium International Conference on Online Learning, 2011.
- [2] Jones, A., & Issroff, K., "Motivation and mobile devices: exploring the role of appropriation and coping strategies", Research in Learning Technology, Vol. 5, No. 3, 2007.
- [3] Roberts, J., Harvesting fragments of time. Mobile learning pilot project. Technical report, McGraw-Hill, 2003.
- [4] Young, J. Mobile College App: Turning iPhones Into 'Super-Clickers' for Classroom Feedback, Chronicle of higher education, 2008.
- [5] Top Hat Monocle, 2013, <https://www.tophatmonocle.com>.
- [6] E-Clicker, Apple App Store, 2013.
- [7] Anderson R., et. al., "Classroom Presenter: Enhancing Interactive Education with Digital Ink", Computer, Vol. 40, No. 9, pp. 56-61, 2007.
- [8] Griswold, W., Simon, B., "Ubiquitous presenter: fast, scalable active learning for the whole classroom", IITCSE, pp. 358, 2006.
- [9] Berque, D. "An evaluation of a broad deployment of DyKnow software to support note taking and interaction using pen-based computers", Journal of CSC, Vol.21, No.6, pp. 204-216, 2006.
- [10] Structure Analysis for Java (STAN), <http://stan4j.com>, 2014.
- [11] McCabe, T., A Software Complexity Measure, IEEE Transactions on Software Engineering, Vol. 2, pp 308-320, 1976.
- [12] Watson, A. and McCabe, T. Structured Testing: A testing methodology using cyclomatic complexity metric, NIST special publication, 1996.
- [13] Clark, M., "Measuring Software Complexity to Target Risky Modules in Autonomous Vehicle. Systems." AUVSI North America Conference, 2008.
- [14] C4 Software Technology Reference Guide, Software Engineering Institute, Carnegie Mellon University, 1997.
- [15] Chidamber, S.R., Kemerer, C.F., A Metrics Suite for Object-Oriented Design. IEEE Transactions of Software Engineering, pp. 476-493, 1994.

# Towards Automatic Consistency Checking between Web Application and its Mobile Application

Xiangping Chen<sup>1,2,3</sup>

<sup>1</sup>Institute of Advanced Technology, Sun Yat-sen University, National Engineering Research Center of Digital Life, Guangzhou, 510006, China

<sup>2</sup>Shenzhen Engineering Laboratory for Mobile Internet Application Middleware Technology, Shenzhen, 518060, China

<sup>3</sup>Research Institute of Sun Yat-sen University in Shenzhen, Shenzhen, 518057, China  
chenxp8@mail.sysu.edu.cn

Zhensheng Xu<sup>1,2</sup>

<sup>1</sup>School of Information Science and Technology, Sun Yat-sen University, National Engineering Research Center of Digital Life, Guangzhou, 510006, China

<sup>2</sup>Research Institute of Sun Yat-sen University in Shenzhen, Shenzhen, 518057, China  
xuzhensh@mail2.sysu.edu.cn

**Abstract**—With the increasing usage of mobile devices in daily life, a lot of websites delivered another mobile application version of application with the same services. If the mobile application is inconsistent with the web application, it may imply possible error or misleading information in the mobile application. This paper proposes an approach for automatic consistency checking between mobile application and its web application. Our approach starts from the generation of page and link information by crawling content from web application and simulating user actions in Android application. Page matching and consistency checking algorithms are proposed and used to find out inconsistency in page, element and link. The validation tool is used in 3 mobile application containing more than 300 pages. The result shows that our approach can find out inconsistency with acceptable precision and recall.

**Keywords**—Consistency checking, Black-Box, Mobile Application

## I. INTRODUCTION

With the increasing usage of mobile devices in daily life, the number of mobile applications increased very quickly. By July 2013, the number of applications in Google play has rose through 1 million<sup>1</sup>.

With this trend, a lot of websites delivered another mobile application version with the same services. Almost all the top 500 global sites<sup>2</sup> provide its corresponding android application<sup>3</sup>.

Because the web application version and mobile application version are provided for the same target users, the functional requirements of the mobile application are already fixed before implementation. The requirement is usually the same as the web application, or subset of the requirements of the web application. In addition, these functionalities are required to be provided in the same way. For example, if a user can access the list of international

news by clicking the element “News” in the homepage, it is improper to provide the list of international news through the “Information” element in the interface of mobile application. In addition, the list of international news should be the same.

For a web application containing hundreds and thousands pages, the implementation of its mobile application is time-consuming with a lot of repetitive work. There may rise inconsistency problem that the mobile application include: (1) error page that shows information which is not included in the web application; (2) error element that displays misleading information; (3) error link that is linked to the wrong page. Although, not all the inconsistencies exist between a mobile application and its web application are faults. For example, a mobile application may provide a special sale for mobile users only. However, inconsistency information is still useful to make sure that extra functionalities are added.

Current works[1-5] focus on consistency checking of different software models in an application. The problem in this paper is how to find out the consistencies between two implementations with the same design model/requirements. Some works proposed testing methods and tools[7-10] for Android applications. However, they do not notice the consistency problem.

In this paper, we propose an approach for automatic consistency checking between mobile application and its web application. We abstract both web application and mobile application as a set of pages and link relationships between pages. Our approach starts from the generation of page and link information by crawling content from web application and simulating user actions in Android application. We define the consistency relationship between two applications and propose page matching and consistency checking algorithm to find out inconsistencies in page, element and link. The tool is used in 3 mobile applications containing more than 300 pages. The result shows that our approach can find out inconsistencies with acceptable precision and recall. Our approach can help software

<sup>1</sup><http://www.techweb.com.cn/world/2013-07-25/1312444.shtml>

<sup>2</sup><http://www.alexa.com/topsites>

<sup>3</sup> <https://play.google.com/store>

engineers and maintainers in finding possible errors during development and maintenance.

The remainder of the paper is organized as follows: Section 2 gives an illustrative example to show inconsistency problem; Section 3 gives the definition of consistency and a general overview of our solution, the definition of consistency and a general overview of our solution, which is further detailed in the Section4. Section 5 presents the evaluation of our approach; Section 6 discusses the limitations of our work, Finally, Section 7 discusses some related works before Section 8 concludes our work.

## II. AN ILLUSTRATIVE EXAMPLE

For the OSChina website (<http://www.oschina.net>), its android application is delivered to provide the same functionality as its website. When we click the button with text “推荐阅读” in the main interface of mobile application, we can see the interface shown in Fig 1(a).

In the home page of the web application, we can find a list of articles, as shown in Fig 1(b). These articles are listed without detail information. In the mobile application, the author, the type of paper, publishing time and number of reviews of an article is listed in the interface.



Figure 1. corresponding pages in mobile application and web application

If we click “最新推荐博客文章” in the home page of the web application, we can see the same list as we see in the home page of OSChina. The related information for these articles is not provided in this page. The related information about an article is provided in different ways for mobile application users and web application users. The inconsistency problem may confuse or mislead users.



Figure 2. Corresponding page in the linked page

## III. APPROACH OVERVIEW

### A. The Consistency Requirements between Web Application and Mobile Application

A web application or mobile application can be abstracted as  $AP = (P, L)$  where  $P$  is the set of web pages,  $L$  is the set of link between web pages. A page in  $P$  includes a list of elements. The elements are divided into two types: elements with link and element without link.  $P = (E_l, E_e)$ .  $link = \langle p_s, e, p_t \rangle$ ,  $link \in L$ ,  $e \in P_s, E_l$ . A link is a link from a web page  $p_s$  to another web page  $p_t$ , and related to an element  $e$  in the web page  $p_s$ .

We define the consistency between a web application and its mobile application based on the following assumptions:

- (1) There exist a web application and a mobile application. The mobile application is developed to provide the same or subset of the services provided by the web application.
- (2) A page in the mobile application contains less information than a page in the web application. As a result, when implementing the mobile application based on the web application, the developer divides the contents in a web page into one or more pages in the mobile application.
- (3) The mobile applications are implemented considering the user habit of web application. However, because a page in the web application may be implemented as more than one page in the mobile application, there are links between pages which are developed for the same page of web application. If these links are neglected, the traces of user action to acquire the same information in both mobile application and web application should be the same.

For a web application  $AP_{web} = \langle P_{web}, L_{web} \rangle$  and a mobile application  $AP_{mobile} = \langle P_{mobile}, L_{mobile} \rangle$ :

- (1)  $\forall p_i \in P_{mobile}, \forall p_j \in P_{web}$

$$p_i.e = p_j.e \rightarrow description(p_i.e) = description(p_j.e)$$

Elements in the page of a web application are different from those in the page of a mobile application because they are developed using different programming language. Two elements of different kinds can be viewed as equal when

they are used to provide the same information or functionality.  $description(e)$  for the element  $e$  cannot be complete. For example, for an element which is a picture, it is hard to generate its description. The extracted description from the application may affect the accuracy for finding a possible consistent page.

$$(2) \forall p_i \in P_{mobile}, \forall p_j \in P_{web}$$

$$consistent(p_i, p_j) \rightarrow p_i.E_l \in p_j.E_l \wedge p_i.E_e \in p_j.E_e$$

We assume that a page in a mobile application contains less information and functionality than its corresponding page in the web application version. A page  $p_i$  in a mobile application is consistent with a page  $p_j$  in its web application when all the elements included in  $p_i$  are also included in  $p_j$ .

$$(3) \forall l_i \in L_{mobile}, \forall l_j \in L_{web}$$

$$consistent(l_i, l_j) \rightarrow$$

$$consistent(l_i.p_s, l_j.p_s) \wedge l_i.e = l_j.e \wedge consistent(l_i.p_t, l_j.p_t))$$

A page in a web application may consistent with multiple pages in the mobile application version. We define that a link  $l_i$  in a mobile application is consistent with a link  $l_j$  in a web application when (1) their source pages are consistent; (2) the target pages are consistent; (3) the related elements are equal.

The consistency relationship between  $AP_{web}$  and  $AP_{mobile}$  is defined as:

$$(1) \forall p_i \in P_{mobile}, \exists p_j \in P_{web}, consistent(p_i, p_j)$$

$$(2) \forall l_i \in L_{mobile}, (\exists l_j \in L_{web}, consistent(l_i, l_j)) \vee$$

$$(\exists p_j \in P_{web}, consistent(l_j.p_s, p_j) \wedge consistent(l_j.p_t, p_j))$$

For each page  $p_i$  belongs to  $P_{mobile}$ , there exists a corresponding consistent page  $p_j$  in  $P_{web}$ ; For each  $l_i$  in  $L_{mobile}$ , if  $l_i$ 's source and target pages are consist with the same page in  $AP_{web}$ , there exists a corresponding consistent link  $l_j$  in  $L_{web}$ .

It should be noted that inconsistency between web application and mobile application do not imply an error in the implementation. A lot of mobile applications provide specific functionalities for mobile user only. However, the inconsistencies are worth of notice during development and evolutional process for software engineers.

### B. Approach Overview

Because the implementations of an application in different platforms are different, the first step of our approach is to generate the description of the mobile application and web application using the same model. The description includes page and link information.

Considering consistency requirement in implementing a mobile application for an existing web application, inconsistency may occur at the page level and the element level. Our approach first searches all possible pages in web applications to find out a possible consistent page for a page in mobile application. Because the link is related to an element in the page, the consistency in the element level considers both elements with and without a link. Our approach compares the elements and the links in sequence.

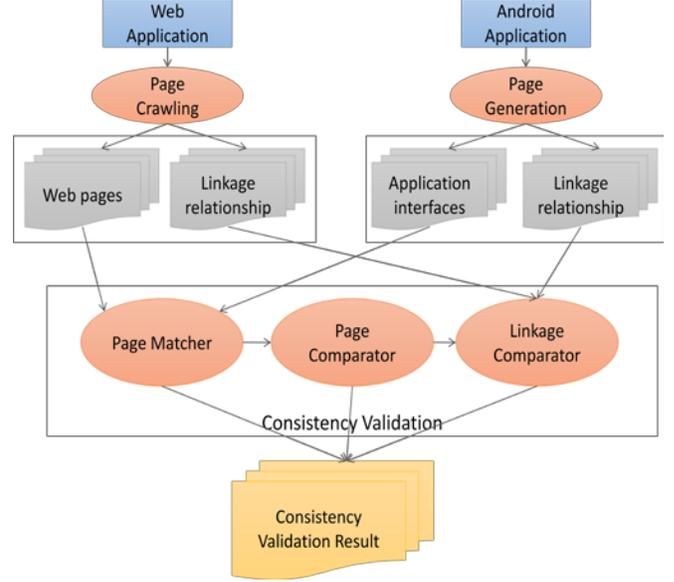


Figure 3. Approach Overview

## IV. CONSISTENCY CHECKING

### A. Generating Page and Link Information

In our approach, we view the application as a black-box and simulate user actions to generate the page and link information. The generation starts from the homepage or main interface of the application.

For each page, we generate descriptions for its elements. Because the descriptions are used in the following comparison, they should be described in the same way. Text and image are two of the most important information for user. In real application, the pictures used in web applications and mobile applications are usually different because they are used for different size of displayers. For the image information, understanding and generating the description of a picture is too hard. As a result, we only use text information as the description of page elements. For an element in page  $p$ , it is recorded using the page id, the text information, and its related link. Text and link can be left blank if the element does not contain text information or it is an element without a link. Element with no text and link information will be deleted because the comparison is meaningless.

The required input for next step, searching possible corresponding page, is a page and all its related pages. As a result, width-first search algorithm is better when exploring the pages. When exploring the pages, our approach records the related elements and the ids of source pages and target pages for links. The search omitted a link when the link's target page is not included in the application or it is already explored before. The search stops when all the pages are explored or the search depth reaches a pre-defined value.

Because the programming language for web applications and mobile applications are different, they have different types of components in the pages. Our approach employs existing parser/tool to extract information from the application.

For the web application, our algorithm traverses pages from the home page and uses a Java HTML parser Jsoup[11] to extract information from each page. Jsoup provides APIs for fetching web page and extracting the text and link information from a page. Because a page in the web application is standalone, we can traverse the pages by using a width first search. During information extraction, we do not deal with dynamic content which is generated during execution of Javascript. These complex situations will be considered in our future work.

For an android application, our approach simulates user actions to traverse all possible pages. Because a page (interface) in the android application is not a standalone page, fetching a page require the simulation of user actions to access this page from main interface. If the traverse is carried out using a width-first search, all the user actions simulated for a page will be repeated carried out when traversing its related pages. As a result, our algorithm traverses page in Android application using the depth-first search.

We employs an Android test automation framework Robotium [12] to carry out the simulated user actions. Robotium is an Android test automation framework that support for native and hybrid applications. In real application, traversing pages in Android application is time-consuming. For example, we extract 90 pages from an Android application in about 1 hour. The max depth for the depth first search can be changed according to how complex the application is.

### B. Page Consistency Checking

Based on the extracted page and link information, our approach traverses all the pages in the mobile application and finds out their corresponding pages. For a page  $p_i \in P_{mobile}$ , its corresponding page is a page  $p_j \in P_{web}$ , when the possibility for consistency( $p_i, p_j$ ) is higher than the possibility of consistency( $p_i, p_j'$ ),  $p_j' \in P_{web}$  and  $p_j' \neq p_j$ .

We suppose that the corresponding page of the main interface of a mobile application is the main page of the web application. Page matching starts from finding corresponding pages linked from the main interface. Because we omitted link whose target page is explored before. For a page  $p$ , there is only one link whose target page is  $p$ .

For pages  $p_i, p_i' \in P_{mobile}$ ,  $p_j \in P_{web}$ , if  $\exists l \in L_{mobile}, l.p_s = p_i \wedge l.p_t = p_i'$ ,  $corresponding(p_i) = p_j$ ,  $corresponding(p_i') \in \{p_j\} \cup \{p | p = e.link.p_s \wedge e \in p_j.E_l \wedge p \in P_{web}\}$ .

The matching is carried out in the set of all possible matched pages  $P_{possible} = \{p | p = e.link.p_s \wedge e \in p_j.E_l \wedge p \in P_{web}\}$ . For  $p_j' \in P_{possible}$ ,  $e \in p_i$

$$(1) \text{matched}(e, p_j') = \begin{cases} 1, \exists e' \in p_j', e.text = e'.text \\ 0, \forall e' \in p_j', e.text \neq e'.text \end{cases}$$

$$(2) \text{matched}(p_i', p_j') = \sum \text{matched}(e, p_j'), e \in p_i'$$

Based on the number of matched elements, we can find the corresponding page by comparing text information.

$$\text{corresponding}(p_i') = \{p | \text{matched}(p_i, p) \geq \text{matched}(p_i, p_j') \\ p \in P_{possible} \wedge p_j' \in P_{possible} \wedge p \neq p_j' \wedge \text{matched}(p_i, p) \neq 0\}$$

If the number of matched elements between pages  $p_i'$  and  $p_j'$  are all zero, there is no corresponding page for  $p_i'$ .

### C. Element and Link Consistency Checking

We divided the inconsistencies into three types. For a page  $p$  in mobile application, there is :

- (1) Page inconsistency: no corresponding page  
 $corresponding(p) = \phi$
- (2) Element inconsistency  
 $\exists e, (e \in p.E_e \wedge e \notin corresponding(p).E_e)$
- (3) Link inconsistency  
 $\exists l, l.p_s = p \wedge l.e \notin corresponding(p).E_l \wedge \\ corresponding(l.p_t) \neq corresponding(p)$

The inconsistent pages can be found in the page consistency checking process. An inconsistent page includes information and functionalities which are not included in the web application. Developers can use this result to see if mobile app specific functionalities are implemented, and analyze the degree of coupling between these new functionalities and existing functionalities.

Element inconsistencies are recorded in the page consistency checking stage when the algorithm tried to compute the number of matched elements. An element inconsistency indicates possible differences in the way to provide information in different applications. The example introduced in section 2 is a classic example of element inconsistency. In some cases, the developer may use image and text to show the same information in different applications. This may bring some undetectable consistency relationship and reduce the accuracy of our method.

Link inconsistency is validated based on the corresponding relationship between pages. Because there may exist more than one page in the mobile application with the same corresponding page in the web application, the links between these pages are not matched to a corresponding link in the web application. The checking filters this kind of links. Other inconsistent links are caused by link inconsistent.

## V. EXPERIMENTS

We use our consistency tool to validate the following android applications with their web applications. We choose three application in our experiment. Because we view both web application and Android application as black-boxes, the first step of our experiment is to find out the website address and download the Android application package file (APK).

TABLE I. EXPERIMENT PROJECTS

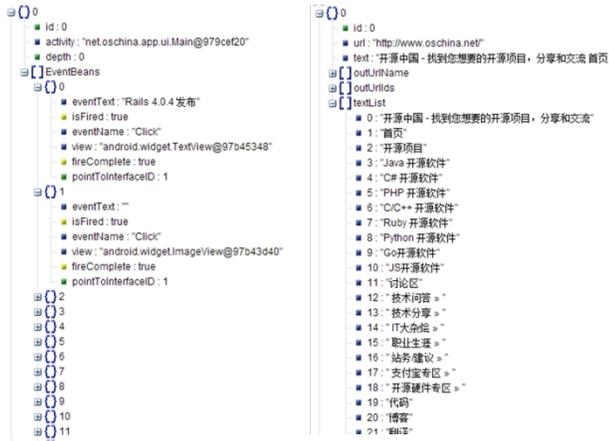
Name	Website address	Android app Size (M)	Version
oschina	http://www.oschina.net	2.2	1.7.6.5
zol	http://www.zol.com.cn	7.2	3.5.2
chinadaily	http://www.chinadaily.com.cn	3.3	3.2.0

Because the generation of page and link information from the android application is time-consuming, we pre-defined the depth during explosion. Table II shows the page size and execution time in extracting page and link information. The pre-defined depth for explosion of Android application is 3.

TABLE II. THE SIZE AND EXECUTION TIME OF GENERATION

Project name	Number of pages (web app)	Execution Time	Number of pages (Android app)	Execution Time
oschina	4070	23min	145	50min
Zol	12430	107min	116	31min
chinadaily	1727	23min	104	36min

The extracted page and link information is saved in a tree structure. Fig 4(a) shows part of the generated information of the main interface of Android application, and Fig 4(b) shows part of the generated information of the home page of web application in the project of oschina.



(a) Android app (b) web app  
Figure 4. Example of generated information

Based on the page and link information generated, our tool compare the consistency relationship between Android applications with their web applications. Table III shows the inconsistency problem we found for each projects.

TABLE III. NUMBER OF INCONSISTENT PAGES

Project name	page inconsistency	Element inconsistency		Link inconsistency	
		pages	elements	pages	links
oschina	4	139	1680	13	76
Zol	33	58	613	15	25
chinadaily	17	58	85	16	36

We choose 30 pages from each Android application randomly, and manual validate the consistency relationship with the web application. The result shows the precision and recall of our consistency validation results. The average precision of our result is 75.4% and 72.2% respectively for inconsistency in the element and link level, 47.2% for inconsistency in the page level.

TABLE IV. VALIDATION RESULT

Project name	Precision Page inconsistency	Recall page inconsistency	Precision element inconsistency	Precision link inconsistency
oschina	1/4	1/2	187/243	40/49
Zol	2/5	2/4	138/174	5/7
chinadaily	13/17	13/19	14/20	7/11
average	47.2%	56.1%	75.4%	72.2%

The precision in checking inconsistencies between pages is relatively low. Because when our algorithm finds a page without corresponding page, it stops matching all its related pages. A page inconsistency usually involves more than one page. The recall of page inconsistency is caused by the matching of some meaningless elements in the page, for example, logo in the page.

VI. DISCUSSION AND LIMITATION

During the experiment, we found some important problems that have not yet resolved in this paper.

First, for the application running in the internet, the content in both web application and mobile application is changing all the time. The changes of content during the process of information extraction bring inconsistency problem. Our approach will generate confusing result if the application is updating during consistency checking.

Second, our approach employs a testing tool Robotium[12] to extract page and link information from Android application, as introduced in section 4. In the implementation, our tool supports extraction of information for pre-defined and simple user-defined view types. User-defined types are now widely used in the implementation of Android application, our approach cannot work well in complicated user-defined type.

VII. RELATED WORK

Consistency checking is widely used in the field of model-driven software engineering and development. Inconsistencies between multiple views in the same phase or different phases of software lifecycle, may bring follow-on errors and unnecessary rework.

Most works [1][2][3] focus on detecting consistency for UML models because UML is widely used and the consistency of models affects the quality of generated code. These works focus on both intra and inter level consistencies. Some works are proposed considering different aspects in consistency validation. Blanc[4] proposed a meta-model independent consistency validation method. In his approach, model is represented as sequences of elementary construction operations, structural and methodological consistency rules can then be expressed uniformly as logical constraints on such sequences. Eged's work [5] focus on an automated approach for detecting and tracking inconsistencies in real time. The performance of his method is not noticeably affected by the model size and common consistency rules but only by the number of consistency rules. Vierhauser et al.[6] proposed an approach for incremental consistency checking for product lines. It can be used to check the consistency of variability models and the consistency of the models with the underlying code base.

To the best of our knowledge, there is not an existing method for consistency checking between a mobile

application and its corresponding web application. Most works focus on the testing of mobile applications, which provide basis for our approach.

Hu et al.[7] present an approach for automating the testing process for Android applications, with a focus on GUI bugs. The approach detects GUI bugs by automatic generation of test cases, feeding the application random events, instrumenting the VM, producing log/trace files and analyzing them post-run. Azim et al. [8] propose two strategies: targeted exploration and depth-first exploration, for the automatic exploration of the Android applications so as to improve the coverage at two granularity levels: activity (high-level) and method (low-level). Amalfitano et al. [9] proposed an automated technique that tests Android apps via their Graphical User Interface (GUI). Its tool Android Ripper[10] is based on a user-interface driven ripper that automatically explores the application's GUI with the aim of exercising the application in a structured manner. In addition, there are useful testing tools for Android application. Troyd[11] allows developers to write Ruby scripts that can drive the Android application. Robotium[12] is an Android test automation framework that support for native and hybrid applications. Our implementation of consistency validation tool is built based on Robotium.

### VIII. CONCLUSION

In this paper, we propose our approach based on the inconsistency problem in developing and using mobile application in daily life. This paper proposes an approach for automatic checking consistency between web application and its mobile application. Our approach starts from the generation of page and link information by crawling content from web application and simulating user actions in Android application. Page matching and consistency checking algorithms are proposed and used to find out inconsistency in page, element and link.

Because the number of inconsistency found in our experiment is too large for comprehension. In the future work, we will try to define inconsistency pattern based on mining the inconsistency data. Repetitive inconsistency simplified as a pattern will improve the usability of our approach for real software development.

### ACKNOWLEDGMENT

This effort is supported by National Natural Science Foundation of China(No. 61100002), the NSFC Guangdong Joint Fund (No. U1201252), Shenzhen Technology R&D

Program for Basic Research(No.JC201105201042A), the Special Foundation for Development of Internet Industry of Shenzhen(No. 20120731105450), the Fundamental Research Funds for the Central Universities, Science and Technology Planning Project of Guangdong Province (No.2012B010900089).

### REFERENCES

- [1] Lucas F J, Molina F, Toval A. A systematic review of UML model consistency management. *Information and Software Technology*, 2009, 51(12): 1631-1645.
- [2] Usman M, Nadeem A, Kim T, et al. A survey of consistency checking techniques for uml models. *Advanced Software Engineering and Its Applications*, 2008. ASEA 2008.: 57-62.
- [3] Egyed A, Letier E, Finkelstein A. Generating and evaluating choices for fixing inconsistencies in UML design models. *23rd IEEE/ACM International Conference on Automated Software Engineering*, 2008: 99-108.
- [4] Blanc X, Mounier I, Mougénot A, et al. Detecting model inconsistency through operation-based model construction. *ACM/IEEE 30th International Conference on Software Engineering, ICSE08*, 511-520, 2008.
- [5] Egyed A. Automatically detecting and tracking inconsistencies in software design models. *IEEE Transactions on Software Engineering*, 2011, 37(2): 188-204.
- [6] Vierhauser M, Grünbacher P, Egyed A, et al. Flexible and scalable consistency checking on product line variability models. *Proceedings of the IEEE/ACM international conference on Automated software engineering*. ACM, 2010: 63-72.
- [7] Azim, Tanzirul, and Iulian Neamtiu. Targeted and depth-first exploration for systematic testing of android apps. *Proceedings of the 2013 ACM SIGPLAN international conference on Object oriented programming systems languages & applications*. ACM, 2013.
- [8] Amalfitano, Domenico, et al. Using GUI ripping for automated testing of Android applications. *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*. ACM, 2012.
- [9] Amalfitano, Domenico, et al. A toolset for GUI testing of Android applications. *Software Maintenance (ICSM)*, 2012 28th IEEE International Conference on. IEEE, 2012.
- [10] Hu, Cuixiong, and Iulian Neamtiu. Automating GUI testing for Android applications. *Proceedings of the 6th International Workshop on Automation of Software Test*. ACM, 2011.
- [11] Hu C, Neamtiu I. Automating GUI testing for Android applications. *Proceedings of the 6th International Workshop on Automation of Software Test*, 2011: 77-83.
- [12] Google Code. Robotium. January, 2014. URL <http://code.google.com/p/robotium/>.
- [13] Jinseong Jeon and Jeffrey S. Foster. Troyd. January, 2012. URL <https://github.com/plum-umd/troyd>.
- [14] Android GUI Ripper. URL <http://wpage.unina.it/ptramont/GUIRipperWiki.htm>

# Mobile Applications: The Paradox of Software Estimation

Laudson Silva de Souza

Department of Informatics and Applied Mathematics  
Federal University of Rio Grande do Norte  
Natal, Brazil 59078-970  
Email: laudyson@gmail.com

Gibeon Soares de Aquino Jr.

Department of Informatics and Applied Mathematics  
Federal University of Rio Grande do Norte  
Natal, Brazil 59078-970  
Email: gibeon@dimap.ufrn.br

**Abstract**—There is a global trend towards the increase of the number of users connected to the network via mobile devices which, consequently, will create an increasing demand for information, applications and content for such equipments. New ways to use existing aplicações are emerging. In particular, systems that were once accessed via web interfaces through personal computers physically located in offices, universities or homes are providing new ways to access from mobile devices which, in turn, have different requirements and capabilities than the personal computers. Hence, the main objective of this paper is to present an estimation model for mobile applications.

**Keywords**—Quality, Estimating, Software Engineering, Mobile Applications

## I. INTRODUCTION

The remarkable growth in the use of mobile devices in several areas, such as health, work, education and public safety. Come along with the accession of all kinds of mobile applications, but those interested in the area awakens a huge discussion on the approach to estimating software, for this private context. For although there was no specific method to handle the particularities of this area. And the software to be developed for mobile applications, are estimated with traditional methods of estimates, ie, facing desktop applications. This new technological scenario is changing old habits and creating new ways of society to access information and interact with computer systems [1], [2] e [3].

According to Gartner, 1.75 billion people own mobile phones with advanced capabilities and foresees further growth in the use of this technology in the coming years [4]. The ITU<sup>1</sup> estimates that there are more than six (6) billion mobile subscribers worldwide. There is a worldwide trend towards increasing the number of users connected to the network via mobile devices, which consequently produce an increasing demand for applications and content for such equipment. With ways of using information systems emerging, such as the access that were previously performed through Web interfaces for personal computers in offices, universities or homes, are providing new ways to access from mobile devices, which turn have different requirements and capabilities of personal computers.

There is a global trend towards the increase of the number of users connected to the network via mobile devices which,

consequently, will create an increasing demand for information, applications and content for such equipments. New ways to use existing aplicações are emerging. In particular, systems that were once accessed via web interfaces through personal computers physically located in offices, universities or homes are providing new ways to access from mobile devices which, in turn, have different requirements and capabilities than the personal computers. Hence, the main objective of this paper is to present an estimation model for mobile applications, denominado “MEstiAM” (Estimation Model for Mobile Applications) proposed in [5].

## II. PRESENTING THE MODEL

According to the proposed [5], which conducts a literature review covering the main estimation methods, but also conducts a systematic review of the characteristics of mobile applications and conducts a survey to ratify the specific characteristics of context finally proposes the method previously cited as a specific method of estimation for the design of mobile application development.

### A. Approaching the Model

The MEstiAM model is totally based on FISMA method, which in its original usage proposes a structure of seven classes of the Base Functional Component or BFC (Base Functional Component) type, which is defined as a basic component of functional requirement. The seven classes used to account for the services during the application of the method are [6]: interactive navigation of the end user and query services (q); interactive input services from end users (i); non-interactive outbound services for the end user (o); interface services for another application (t); interface services for other applications (f); data storage services (d) and algorithmic manipulation services (a).

The identification for each class name BFC previously mentioned, with a letter in parenthesis, is used to facilitate the application of the method during the counting process, because each of the seven classes BFCs are composed of other BFC classes which, at the time of calculating, these BFCs “daughter” classes are identified by the letter of their BFC “mother” class followed by a numeral, as can be seen in Figure 1.

The unit of measurement is the point of function with the letter “F” added to its nomenclature to identify the “FiSMA”,

<sup>1</sup>International Telecommunication Union

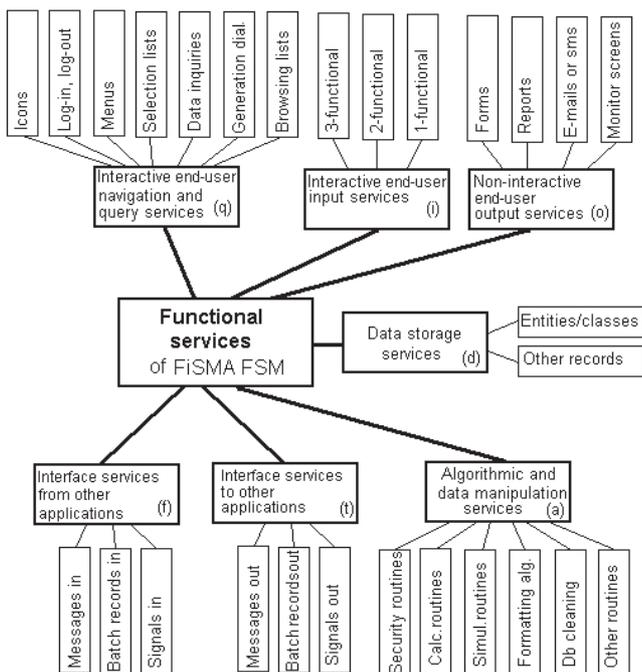


Fig. 1. Types of BFCs classes of the base model.

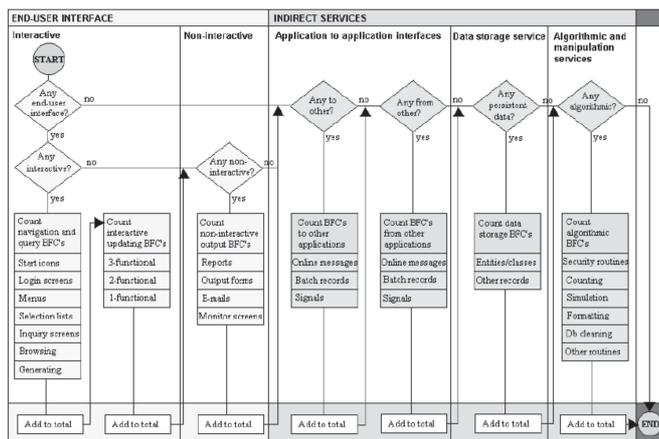


Fig. 2. Representation of the measurement process of the base model.

resulting in FfP (*FiSMA Function Point*) or Ffsu (*FiSMA functional size unit*). The measurement process generally consists of measuring the services and end-user interface and the services considered indirect [6], as can be seen in Figure 2.

Figure 2 shows the process of measuring the base model, in which it defines each step and sum of each BFC class of the model. Briefly, the process of counting should be done as follows. Identify:

- How many types of BFCs does the software have?
- Which are they? (identify all)
- What are they? (provide details of each BFC identified).

After doing this, it is necessary to add each BFC root using

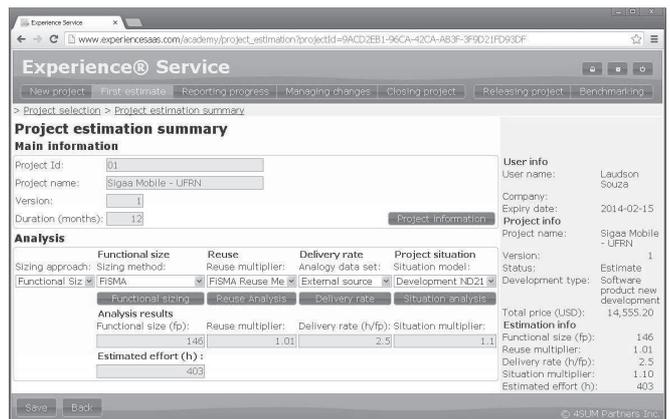


Fig. 3. Final Report of FiSMA applied to Sigaa Mobile.

the formulas pre-defined by the method and their assignments. Finally, the formula of the final result of the sum is the general sum of all the BFCs classes.

### B. Applying the Model

The FiSMA method can be applied manually or with the aid of the Experience Service<sup>2</sup> tool, which was the case, provided by FiSMA itself through contact made with senior consultant Pekka Forselius and with the chairman of the board Hannu Lappalainen.

When using the tool, it is necessary to perform all the steps of the previous subsection to obtain the functional size. Figure 3 shows the final report after the implementation of the FiSMA on a real system, the Management of Academic Activities Integrated System (Sigaa) in its Mobile version, developed by the Superintendence of Computing (SINFO) of the Federal University of Rio Grande do Norte (UFRN).

After the application of FiSMA, the functional size of the software is obtained and from this it is possible to find the effort using the formula: Estimated effort (h) = size (fp) x reuse x rate of delivery (h/fp) x project status; the latter is related to productivity factors that are taken into account for the calculation of the effort. However, of the factors predefined by the FiSMA regarding the product, only 6 (six) are proposed, in which the basic idea of the evaluation is that "the better the circumstances of the project, the more positive the assessment". The weighting goes from - - to + +, as follows:

#### Caption:

- (+ +) = [1.10] Excellent situation, much better circumstances than in the average case;
- (+) = [1.05] Good situation, better circumstances than in the average case;
- (+ / -) = [1.0] Normal situation;
- (-) = [0.95] Bad situation, worse circumstances than in the average case;

<sup>2</sup><http://www.experiencesaas.com/>

- (- -) = [0.90] Very bad situation, much worse circumstances than in the average case.

#### Productivity factors:

- Functionality requirements - compatibility with the needs of the end user, the complexity of the requirements.
    - (- -) Complex and critical application area (thousands of FPs), multiple users and multicultural system.
    - (-) Interoperable application area with some complex characteristics, requiring special understanding from users and developers.
    - (+/-) Partly automated, integrated application area and a medium size application (between 600 and 1000 FPs) with standard security requirements.
    - (+) Application area mostly automated and application with less than 5 interfaces with other systems; there are specific security requirements.
    - (+ +) Very mature application area, simple and easy, a small stand-alone application (less than 200 FPs) for a small group of users.
  - Reliability requirements: maturity, tolerance to faults and recovery for different types of use cases.
    - (- -) Malfunctions may put in danger human lives and cause significant economic or environmental losses.
    - (-) The software is part of a large real-time system where all the failures of operation will cause problems to many other applications.
    - (+/-) Not more than 2 hours of downtime is acceptable, but the system recovery routines are appropriate.
    - (+) Need for non-continuous operation, but daily.
    - (+ +) Need for periodic operation. Pausing for a few days will not cause any damage to the organization.
  - Usability requirements: understandability and easiness to learn the user interface and workflow logic.
    - (- -) A large number of different types of end users around the world.
    - (-) 2 or 3 different types of users with different skills.
    - (+/-) A large number of end users with equal abilities.
    - (+) No more than tens or hundreds of homogeneous users in perhaps more than one location.
    - (+ +) Only a few users, all located on one site.
  - Efficiency requirements: effective use of resources and adequate performance in each use case and under a reasonable workload.
    - (- -) Complex database with millions of data records and transactions per day, thousands of simultaneous end users.
    - (-) Large database, hundreds of simultaneous end users, critical response most of the time.
    - (+/-) Large database, less than millions of data records and less than hundreds of simultaneous end users.
    - (+) Medium database in volume and structure, simple and predictable data requests from some simultaneous end users.
    - (+ +) Simple and small database without simultaneous end users or complex data requests.
  - Maintainability requirements: lifetime of the application, criticality of fault diagnosis and test performance.
    - (- -) Very large strategic software (over 20 years of lifetime) in a volatile area of business, with frequent changes in laws, regulations and business rules.
    - (-) Large software (10-20 years of lifetime), and frequent changes in laws, regulations and business rules.
    - (+/-) Medium size software (5-10 years of lifetime), monthly changes in laws, regulations and business rules.
    - (+) Small software, rarely changes (2 to 5 years of lifetime).
    - (+ +) Temporary software (less than 2 years of lifetime), without modifications.
  - Portability requirements: adaptability and instability to different environments, to the architecture and to structural components.
    - (- -) Software users are located in many types of organizations, with various platforms (hardware, browsers, operating systems, middleware, protocols, etc), various versions and various update frequencies.
    - (-) The software must operate on some different platforms (hardware, browsers, operating systems, middleware, protocols, etc) and in various versions of each of them.
    - (+/-) Each version of the software must run on multiple versions of a given platform (hardware, browser, operating system, middleware, protocols, etc), and the frequencies of update of the users are quite predictable.
    - (+) The software must run on a given platform (hardware, browser, operating system, middleware, protocols, etc), but the use of system-level services is limited because the upgrade process is partial.
    - (+ +) Software must be run on a particular platform (hardware, browser, operating system, middleware, protocols, etc), but the upgrade process is completely controllable.
- Among the productivity factors mentioned above, only the “Portability Requirement” factor fits in harmony with the “Portability” characteristic regarding both hardware and software. However, none of the other factors discusses the characteristics of mobile application, in other words, after obtaining the functional size of the software and applying the productivity factors related to the product to estimate the effort, this estimate ignores all of the characteristics of mobile applications, judging that the estimate of traditional information systems is equal to the mobile application. However, with the proposal of the creation of new productivity factors, which would be the specific characteristics of mobile applications, this problem will be solved, as presented below.
- Performance Factor:
    - (-) The application should be concerned with the optimization of resources for a better efficiency and response time.
    - (+/-) Resource optimization for better efficiency and response time may or may not exist.
    - (+) Resource optimization for better efficiency and response time should not be taken into consideration.

- Power Factor:
  - (-) The application should be concerned with the optimization of resources for a lower battery consumption.
  - (+/-) Resource optimization for lower battery consumption may or may not exist.
  - (+) Resource optimization for a lower battery consumption should not be taken into consideration.
- Band Factor:
  - (-) The application shall require the maximum bandwidth.
  - (+/-) The application shall require reasonable bandwidth.
  - (+) The application shall require a minimum bandwidth.
- Connectivity Factor:
  - (-) The application must have the maximum willingness to use connections such as 3G, Wi-fi, Wireless, Bluetooth, Infrared and others.
  - (+/-) The application must have reasonable predisposition to use connections such as 3G, Wi-Fi and Wireless.
  - (+) The application must have only a predisposition to use connections, which can be: 3G, Wi-fi, Wireless, Bluetooth, Infrared or others.
- Context Factor:
  - (-) The application should work offline and synchronize.
  - (+/-) The application should work offline and it is not necessary to synchronize.
  - (+) The application should not work offline.
- Graphic Interface Factor:
  - (-) The application has limitations due to the screen size because it will be mainly used by cell phone users.
  - (+/-) The application has reasonable limitation due to the screen size because it will be used both by cell phone and tablet users.
  - (+) The application has little limitation due to the screen size because it will be mainly used by tablet users.
- Input Interface Factor:
  - (-) The application must have input interfaces for touch screen, voice, video, keyboard and others.
  - (+/-) The application must have standard input interfaces for keyboard.
  - (+) The application must have any one of the types of interfaces, such as: touch screen, voice, video, keyboard or others.

The proposed factors take into account the same weighting proposed by FiSMA, but only ranging from - to +, in other words:

- (+) = [1.05] Good situation, better circumstances than in the average case;
- (+ / -) = [1.0] Normal Situation;
- (-) = [0.95] Bad situation, worse circumstances than in the average case.

The functional size remains the same, thus affecting only the formula used to obtain the effort, which will now consider in its “project situation” variable the new productivity factors specific for mobile applications.

The validation process was as follows, was raised the total effort expended in developing the Sigaa Mobile project, ie, we obtained the actual effort. After we applied the method of estimation FISMA, in his original proposal thus obtaining an estimate of effort. Then we applied the method MEstiAM also generating an effort estimate finally the comparative analysis between the three estimates generated was performed to verify which method is closer to the actual effort spent. As can be seen in Table I.

TABLE I. ANALYSIS OF ESTIMATES OF SIGAA MOBILE

Real Effort Spent	MEstiAM Model	FiSMA Model
860 h	792 h	403 h

As can be seen in Table I, the proposed method, MEstiAM, which is closest to the actual effort spent. You FISMA model, it was very much desired for the new model and the actual effort expended.

### III. CONCLUSION

Based on these results, it was evident the difference between a specific method for this context MEstiAM and other methods. Since they fall short by not taking into account the particularities of mobile applications, making it partially ineffective method in this situation.

Finally, it is concluded that it is entirely appropriate and feasible to use the method MEstiAM where it should take into account all peculiarities of such applications. Putting up finally a division that actually there are considerable differences in the design development of mobile applications.

### REFERENCES

- [1] L. Naismith, M. Sharples, G. Vavoula, P. Lonsdale *et al.*, “Literature review in mobile technologies and learning,” 2004.
- [2] G. Macario, M. Torchiano, and M. Violante, “An in-vehicle infotainment software architecture based on google android,” in *Industrial Embedded Systems, 2009. SIES '09. IEEE International Symposium on*, 2009, pp. 257–260.
- [3] T. Liu, H. Wang, J. Liang, T.-W. Chan, H. Ko, and J. Yang, “Wireless and mobile technologies to enhance teaching and learning,” *Journal of Computer Assisted Learning*, vol. 19, no. 3, pp. 371–382, 2003.
- [4] I. GARTNER. (2013) Gartner says worldwide mobile phone sales declined 1.7 percent in 2012. egham, uk: Gartner, 2013. [Online]. Available: <http://www.gartner.com/newsroom/id/2335616>
- [5] L. S. de Souza and G. S. de Aquino Jr., “The applicability of present estimation models to the context of mobile applications,” in *ENASE*, 2014, p. 9.
- [6] F. S. M. A. FiSMA. (2004) Fisma functional size measurement method version 1-1. [Online]. Available: <http://www.fisma.fi/in-english/methods/>

# An Evolutionary Methodology for Optimized Feature Selection in Software Product Lines

Xiaoli Lian LiZhang

State Key Laboratory of Software Development Environment  
School of Computer Science and Engineering  
Beihang University  
Beijing, China

[lianxiaoli@cse.buaa.edu.cn](mailto:lianxiaoli@cse.buaa.edu.cn)

[lily@buaa.edu.cn](mailto:lily@buaa.edu.cn)

**Abstract**—Feature modeling is the primary technology to capture and document the commonalities and variability among all of the members in a product line. Individual products are customized by selecting features according to the requirements. The work of feature selection is complex because of: 1) the complex dependencies and constraint relationship amongst features; 2) the multiple competing and conflicting non-functional requirements (NFRs); 3) the constraints to NFRs; 4) the explicit functional requirements. To select optimized feature set that conforms to the feature relations and satisfies both the functional and non-functional requirements and the related constraints, an evolutionary algorithm template which employs multi-objective optimization algorithms to optimally select features in SPLs, is proposed. In the experiments, two different algorithms are designed based on our template. Empirical results show the remarking performance of our algorithms on time especially when the feature models are large and complex.

**Keywords**-Product Line Engineering;feature selection;multi-objective optimization;non-functional optimization

## I. INTRODUCTION

Product Line Engineering<sup>[1]</sup> has been shown an effective approach to product software for decreasing the cost, improving the product quality and accelerating time to market by many organizations such as Boeing, Bosch Group, Nokia and so on. Feature model is a popular way to express the commonalities and variability among all of the members in a product family. The tailored products are derived by selecting and removing features from feature model.

In realistic feature model, non-functional requirements (NFRs) are always demanded besides the functional ones (FRs). These NFRs are also conflicting and have to be traded-off. In addition, massive features have constraints and dependencies relation with other ones, e.g., 86% features declare constraints in Linux and eCos. So a newly selected feature would often result some other ones obsolete and these have to be reselected.

Multiobjective Evolutionary Optimization Algorithms (MEOAs) aims to optimize more than one objective in the presence of trade-offs. It is a proper way to deal with the competing NFRs in feature selection. Unfortunately, the random crossover and mutation of MEOAs inevitably destroy the feature dependencies and constraints. So in the current paper, a repair

operator is designed to revise the destructive effects by the random operations during evolutionary process.

The main contributions of this paper can be summarized as: (1) We design a uniform representation for different kinds of relations amongst features; (2) We design *SolutionRevise*, a common solution revise operator, which can be applied to different MEOAs and revise the feature selection that violates the feature model constraints; (3) We design an evolutionary algorithm template named MOOFs and show the remarking performance by two algorithms named MOOF<sub>HD</sub> and MOOF<sub>es</sub>.

## II. FEATURE MODEL

FIG.1 illustrates a simple feature model for mobile phone product line. The rectangles indicate features and the line between them are their relations. From FIG.1, we can see two basic kinds of relations amongst features:

1. Relationships between a parent feature and its child features, includes *mandatory*, *optional*, *alternative* and *or*.
2. Cross-tree (or cross-hierarchy) constraints typically state that one feature *excludes* or *requires* another one.

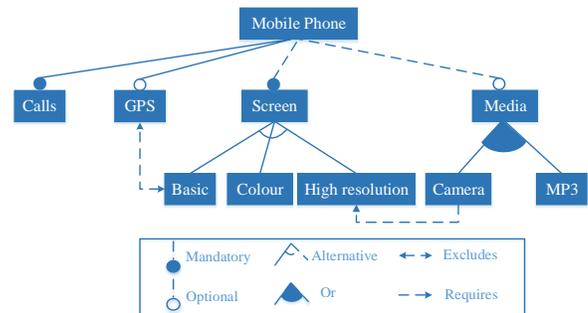


FIG. 1 FEATURE MODEL FOR MOBILE PHONE PRODUCT LINE

There can be group cardinality of the form  $[m,n]$  in feature groups, which declare that at least  $m$  and at most  $n$  members of a feature group can be selected. Cross-tree constraint ratio (CTCR) is defined in [2] as the ratio of the number of features in the cross-tree constraints to the number of features in the feature tree and it is a common indicator to indicate the complexity of a feature model.

To deal with the NFRs, some researches extended the feature model with feature attributes<sup>[2][3][4][5]</sup>. In our paper, we assumed

This work was partially supported by National Natural Science Foundation of China under Grant No.61370058 and State Key Laboratory of Software Development Environment under Grant No. SKLSDE-2014ZX-17.

the NFRs have been quantified. The classification and quantification of the NFRs are not our concerns in this paper.

### III. METHODOLOGY

#### A. Preprocessing

##### 1) Feature model constraints and functional requirements representation and non-functional requirements

From section II, we can see there are different kinds of relations amongst features. These different relations restrict the existence of features in different way. To simplify the treatment for different constraints, we encode them in a uniform format firstly. We defined them as *rule* and the Chomsky grammar can be seen in Definition 1.

**Definition 1.** A *rule* is defined as a quadri-tuple  $(\mathbf{N}, \Sigma, \mathbf{P}, \mathbf{S})$ , where the non-terminal set  $\mathbf{N} = \{rule, A, B, G, Min, Max, N, N_+\}$ , the terminal set  $\Sigma = \{\varepsilon, [, ], ,, <, >, \{, \}, *, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, featureID\}$ , the start symbol  $\mathbf{S} = \{rule\}$ . The following are the rewrite rules in  $\mathbf{P}$ .

$$\begin{aligned} rule &:= <A, B > \\ A &:= \varepsilon | B \\ B &:= featureID | G \\ G &:= [featureID(, featureID)^+ ] \{Min, Max\} \\ Min &:= 0 | N_+ N^* \\ Max &:= Min | * \\ N &:= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 \\ N_+ &:= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 \end{aligned}$$

A *rule* can be expressed as  $\langle A, B \rangle$  which is a relation that A requires B. A and B are marked *first* and *second* respectively in the following sections. Both A and B can be a single feature or a feature group. Taking one ternary cross-tree constraint  $\sim a$  or  $\sim b$  or  $c$  for example, the *rule* can be expressed as  $\langle [a, b] \{2, 2\}, c \rangle$  which means that  $c$  must be contained when both  $a$  and  $b$  are selected.

In consideration of clear expression, we state the condition that a feature group is satisfied.

**Definition 2.** A feature group is satisfied when the number of feature members selected is between *minRange* and *maxRange*.

In addition, A can be  $\varepsilon$  when B must be satisfied in any case. Through analyzing the explicit functional requirements, we found they can be described as rule with the  $\varepsilon$  first. If one feature noted as  $f$  is required, there should be a *rule*  $\langle \varepsilon, f \rangle$  which means that  $f$  must be selected anyhow.

The feature selection conforming to all of the rules is called as “correct” solution in this paper. Otherwise, the selection is “incorrect”.

Just as the discussion in section II, we assumed the NFRs have been quantified by the attributes of features and the values have been assigned in some way. The constraints for NFRs were described as equalities or inequalities, e.g.,  $cost \leq 400$ .

##### 2) Feature chromosome encoding

The feature chromosome is encoded as binary string. The length of the binary string is equal with feature number. If one feature is selected, the corresponding bit is 1; otherwise the bit set is 0.

#### B. MOOFs: an Multi-Objective Optimization algorithm template for Feature selection

Based on the idea of MEOAs, we designed our algorithm template *MOOFs* listed in Algorithm 1. In *MOOFs*, we did not confine any concrete method such as population initialization, the fitness function, the crossover operator, and the mutation operator and so on. The selection of these operators depends on the certain application.

---

##### Algorithm 1: *MOOFs*

---

**Input :**  $a$  (population size)  
 $N$  (maximum number of generations)  
 $k$  (fitness scaling factor)  
 $R$  (rule set)

**Output:**  $A$  (Pareto set approximation)

Initialize  $P$   
 $p' = SolutionRevise(p)$ , for all  $p \in P$   
 $P^R \leftarrow p'$

**Repeat**  
 calculate fitness values of  $P^R$   
 select two individuals from  $P^R$  to fill mating pool  $M$   
 apply mutation operator to  $M$  and get offspring  $O$   
 $O' = SolutionRevise(O)$   
 $P^R \leftarrow O'$   
 Select and remove individuals from  $P^R$

**Stop until**  $m \geq N$  or other stopping condition  
 Return  $A$  that is the Pareto Front approximation in  $P^R$

---

#### C. SolutionRevise: an algorithm to revise feature selections

According to the type of first and second in a rule defined in section III.A.1), the rules can be grouped into five categories:  $\langle \varepsilon, single\ feature \rangle$ ,  $\langle \varepsilon, feature\ group \rangle$ ,  $\langle single\ feature, single\ feature \rangle$ ,  $\langle single\ feature, feature\ group \rangle$  and  $\langle feature\ group, single\ feature \rangle$ . The type  $\langle feature\ group, feature\ group \rangle$  does not exist, because a feature group must have a single feature parent. So their relationships can be depicted as  $\langle single\ feature, feature\ group \rangle$ . On the basic of this classification, we defined two principles to select or remove a feature.

---

##### Algorithm 2: *SolutionRevise*

---

**Input :**  $S$  (one configuration to be revised)  
 $R$  (rule set)

**Output:**  $S_I$  (feature selection conforming to  $R$ )

$S_I \leftarrow \emptyset, S_E \leftarrow \emptyset, S_G \leftarrow \emptyset$

**foreach** rule  $r$  in  $S$   
 if the second marked as  $s$  of  $r$  is a group and the *minRange*  $> 0$ , **then**  
 $S_G \leftarrow r$

**end**  
 IncludeFeature(0);  
**foreach** feature  $f$  in  $S$   
 if  $f \notin S_I$  and  $f \notin S_E$ , **then** IncludeFeature( $f$ )  
**end**  
**foreach** rule  $r$  in  $S_G$   
 Note the number of feature members in the second of  $r$  which has been selected as  $N$   
**While**  $N < minRange$  **do**  
 select feature  $f'' \notin S_I$  and  $f'' \notin S_E$   
 IncludeFeature( $f''$ )  
 Recount  $N$   
**end**  
**end**

---

**Definition 3:** A feature  $f$  is selected, if any of the following conditions is true: 1)  $f$  is neither included nor excluded; 2)  $f$  as a single feature is the second of a rule and the corresponding first that is also a single feature is selected; 3)  $f$  as a single feature is the second of a rule and the corresponding first is a feature group

and the first is satisfied. 4)  $f$  existing in a feature group which is the second of a rule and the first is null or a single feature being selected, the number of group members that are not excluded is just the *minRange* of the feature group.

---

**Algorithm 3: IncludeFeature**


---

**Input :**  $f$  (one feature to be included)  
 $R$  (rule set)  
 $S_G$ (the set of rule whose second dimension is a feature group with the *minRange*  $>0$ )  
**Output:**  $S_I, S_E$   
 $S_I \leftarrow f$   
**foreach** rule  $r$  in  $S_G$   
    Denote the current group of  $r$  as  $S_{TG}$ ,  
    **if**  $f \in S_{TG}$  and the first of  $r$  is satisfied or the first is null **then**  
        Count the feature number of  $S_{TG}$  in  $S_I$  as  $Y$   
        **if** *maxRange* =  $Y$ , **then** ExcludeFeature(the other features of  $S_{TG}$  neither in  $S_I$  nor in  $S_E$ )  
    **end**  
**foreach** rule  $r$  in  $R$   
    **if** the first is a group set  $S_{TG}$  and  $f \in S_{TG}$  and the first is satisfied and the second is a single feature  $f'$ , **then** IncludeFeature( $f'$ )  
    **if** the first is  $f$  and the second is a single feature  $f'$  **then**  
        IncludeFeature( $f'$ )  
    **if** the first is  $f$  and the second is a group set  $S_{TG}$  **then**  
        set the feature number of  $S_{TG}$  in  $S_I$  as  $Y$ , the feature number in  $S_E$  is  $E$   
        **if** *maxRange* =  $Y$ , **then** ExcludeFeature(the other feature members of  $S_{TG}$  neither in  $S_I$  nor in  $S_E$ )  
        **if**  $\text{size}(S_{TG}) - E = \text{minRange}$ , **then**  
            IncludeFeature(other feature members of  $S_{TG}$  neither in  $S_I$  nor in  $S_E$ )  
**end**

---



---

**Algorithm 4: ExcludeFeature**


---

**Input :**  $f$  (one feature to be excluded)  
 $R$  (rule set)  
 $S_G$ (the set of rule whose second dimension is a feature group with the *minRange*  $>0$ )  
**Output:**  $S_I, S_E$   
 $S_E \leftarrow f$   
**foreach** rule  $r$  in  $S_G$   
    Denote the current group in  $r$  as  $S_{TG}$ ,  
    **if**  $f \in S_{TG}$  and the first of  $r$  is satisfied or the first is null **then**  
        Count the number of feature members of  $S_{TG}$  in  $S_E$  as  $E$   
        **if**  $\text{size}(S_{TG}) - E = \text{minRange}$  **then**  
            IncludeFeature(other features of  $S_{TG}$  neither in  $S_I$  nor  $S_E$  in  $S_{TG}$ )  
    **end**  
**foreach** rule  $r$  in  $R$   
    **if**  $f$  as a single feature is the second of  $r$ , and the first  $f'$  is a single feature, **then** ExcludeFeature( $f'$ )  
    **if**  $f$  as a single feature is the second of  $r$ , and the first is a group  $S_{TG}$  **then**  
        count the number of features of  $S_{TG}$  in  $S_E$  as  $E$   
        **if**  $\text{size}(\text{feature members of } S_{TG}) - E \geq \text{minRange}$  **then**  
            ExcludeFeature(the other feature members of  $S_{TG}$  neither in  $S_E$  nor  $S_I$ )  
    **if** the second is a group set  $S_{TG}$  and  $f \in S_{TG}$  and the first is a single feature  $f'$ , **then**  
        count the feature number of  $S_{TG}$  in  $S_I$  as  $Y$ , the number in  $S_E$  is  $E$   
        **if**  $\text{size}(\text{feature members of } S_{TG}) - E < \text{minRange}$  **then** ExcludeFeature( $f'$ )  
**end**

---

**Definition 4:** A feature  $f$  is excluded, if any of the following conditions is true: 1)  $f$  as a single feature is the first of a rule, the second which is also a single feature is excluded. 2)  $f$  as a single feature is the first of a rule, and the second is a feature group and the number of feature members which has not been excluded is less than *minRange*. 3)  $f$  existing in a feature group which is the first of a rule, the second which is a single feature is excluded, and the number of features that are not excluded is above the lower limit of the group. 4)  $f$  existing in a feature group which is

the second of a rule and the first is null or a single feature being selected, the number of features included is just the *maxRange* of the group.

*SolutionRevise* was designed based on these two principles. Algorithm 2 shows the detailed process of *SolutinRevise*. *SolutionRevise* contains two related operators named *IncludeFeature* and *ExcludeFeature*. These two algorithms, shown in Algorithm 3 and Algorithm 4, determine whether a feature is selected or excluded. *IncludeFeature* puts one specific feature  $f$  and the features that  $f$  requires in  $S_I$  and excludes the extra features when any *maxRange* is achieved after including  $f$ . Similarly, the operator of *ExcludeFeature* puts one specific feature  $f$  and the features that require  $f$  in  $S_E$  and puts the left features in  $S_I$  when the number of features in any group left by  $S_E$  is *minRange* after excluding  $f$ .

#### IV. EXPERIMENT SET UP

##### A. Setting up the feature model

Four feature models including two realistic models and two synthesis were selected from SPLOT<sup>[7]</sup>, a popular feature model repository. In this paper, we augmented the feature models with the same attributes and contributions of their values as that in [6]. And the five optimization objectives<sup>[6]</sup>: to minimized total cost, to maximize features that were used before, to minimize the total number of known defects, to maximize correctness and to maximize the number of offered features.

TABLE I. FEATURE MODELS USED IN THIS PAPER

Feature Model	Features	Total Rules	CTCs	CTCR	Limits
WebProtal	43	51	6	2.6%	COST $\leq$ 400
EShop	290	333	21	6.8%	COST $\leq$ 2900
FM500	500	572	50	7.8%	COST $\leq$ 4000
FM1000	1000	1146	100	8.0%	COST $\leq$ 9000

##### B. Setting up algorithms

In this paper, we designed MOOF<sub>HD</sub> and MOOF<sub>E+</sub> by introducing our *SolutionRevise* operator into IBEA suggested by [6]. Our two algorithms with NSGAI<sup>[8]</sup>, SPEA2<sup>[9]</sup>, IBEA<sub>HD</sub><sup>[10]</sup> and IBEA<sub>E+</sub><sup>[10]</sup> were experimented on the four models. All of these algorithms were implemented in jMetal<sup>[11]</sup>, a popular framework for multi-objective optimization. The parameters of these algorithms are default values in jMetal: population size=100, single-point crossover, crossover probability=0.90, bit-flip mutation, mutation probability=1/NumberOfVariables.

To diminish the impact of stochastic behavior of these randomized algorithms, we ran every algorithm independently run 30 times on a feature model. Initially, we performed all algorithms with 25K objective function evaluations. If no “correct” solutions were created, the evaluations time would turn to 500K. If not still, the evaluations would be 1M or 2M or 5M or 10M. But if no still even with 10M evaluations, the trial would be stop.

#### V. EMPIRICAL RESULTS

This section shows the comparing results of six algorithms on four feature models. We compare the run time to generate one correct solution.

- All other algorithms perform moderately with smaller WebProtal and EShop. However, to bigger model FM500, only IBEA<sub>HD</sub> can get correct solutions when evaluations is 500K. To FM1000, no correct solution is generated by all these algorithms till 10M objectives functional evaluations. However, the *SolutionRevise* operation makes MOOF<sub>HD</sub> and MOOF<sub>ε+</sub> always product correct solution.

TABLE II. RUN TIME COMPARISON

Feature Model	Algorithms	Eval.	RunTime	STR
WebProtal	NSGAII	25K	0.55sec	3.23
	SPEA2	25K	59.08sec	453.46
	IBEA <sub>HD</sub>	25K	0.11sec	-0.15
	IBEA <sub>ε+</sub>	25K	21.73sec	166.15
	MOOF <sub>HD</sub>	25K	0.80sec	5.15
	MOOF <sub>ε+</sub>	25K	0.13sec	/
EShop	NSGAII	500K	47.22sec	261.33
	SPEA2	500K	3.67sec	19.39
	IBEA <sub>HD</sub>	25K	3.68sec	19.44
	IBEA <sub>ε+</sub>	500K	135.97sec	754.39
	MOOF <sub>HD</sub>	25K	1.88sec	9.44
	MOOF <sub>ε+</sub>	25K	0.18sec	/
FM500	NSGAII	10M	--	--
	SPEA2	10M	--	--
	IBEA <sub>HD</sub>	500K	11.45sec	44.80
	IBEA <sub>ε+</sub>	10M	--	--
	MOOF <sub>HD</sub>	25K	2.67sec	9.68
	MOOF <sub>ε+</sub>	25K	0.25sec	/
FM1000	NSGAII	10M	--	--
	SPEA2	10M	--	--
	IBEA <sub>HD</sub>	10M	--	--
	IBEA <sub>ε+</sub>	10M	--	--
	MOOF <sub>HD</sub>	25K	4.74sec	9.68
	MOOF <sub>ε+</sub>	25K	0.27sec	/

"--" means no correct solution produced by the algorithm in the corresponding row and "/" means no calculation on STR. If the STR is negative, the time consumed by MOOF<sub>ε+</sub> is larger than the algorithm in the row.

- From the column of *RunTime*, we can observe that IBEA<sub>HD</sub> works best comparing with NSGAII, SPEA2 and IBEA<sub>ε+</sub> which is consistence with [6]. However, to larger model, its performance is turning worse than our algorithms.

Looking at the column of STR, except for the smallest WebProtal model, about 3 to 654 times more seconds are consumed by other algorithms to produce one correct solution than MOOF<sub>ε+</sub>.

## VI. RELATED WORK

Some remarking works transformed the feature selection into a SAT problem like Benavides et al.[2]. White et al.[12] and Guo et al. [13] selected features satisfying a series of resource constraints. Sayyad et al.[6] tried MEOAs to do feature selection towards multi-attribute optimization and proved the significant of IBEA. However, these approaches did not concern the explicit FRs and NFRs with constraints or not.

A solution repair operator was also proposed in [13]. It is for the basic feature model with binary cross-tree constraints. Mendonca et al.[14] observed that the cross-tree constraints in

the realistic models consisted of a mix of binary and ternary CNF clauses which are considered in our methodology.

## VII. CONCLUSIONS AND FUTURE WORKS

Aiming to automatically select features that conforms to the feature model and satisfies both the functional and nonfunctional requirements and the related constraints, we provided a multi-objective optimization algorithm template for feature selection which combined our configuration reviser with the existing MEOAs flexibly. During empirical experiments, we designed two algorithms based on the IBEA suggested by [6]. Through comparing with four widespread MEOAs, our algorithms work best when the timing-consuming and multi-attribute performance are considered comprehensively.

## REFERENCES

- [1] P.Clements and L.Northrop. *Software Product Lines: Practices and Patterns*. SEI Series in Software Engineering, Addison-Wesley (2001).
- [2] D. Benavides, A. Ruiz-Cortés and P. Trinidad, "Automated Reasoning on Feature Models," in *Proceeding of CAISE*, 2005, pp. 491-503.
- [3] N.Siegmund, M.Kuhlemann, M. Rosenmüller, C. Kästner and G. Saake. "Integrated product line model for semi-automated product derivation using non-functional properties," in *Workshop on Variability Modelling of Software-intensive Systems (VaMoS)* (pp. 25–31).2008. University of Duisburg-Essen
- [4] S. Soltani, M. Asadi, D. Gašević, M. Hatala, E. Bagheri, "Automated planning for feature model configuration based on functional and non-functional requirements," *ICPC 2012*
- [5] N. Siegmund, M. Rosenmüller, M. Kuhlemann, C. Kästner, S.Apel and G.Saake, "SPL Conqueror: Toward optimization of non-functional properties in software product lines. " *Software Quality Journal* 20(3-4): 487-517 (2012)
- [6] A.S.Sayyad, T.Menzies, H.Ammar. "On the Value of User Preferences in Search-Based Software Engineering: A case Study in Software Product Lines." In *Roc.ICSE (San Francisco, USA,2013)*,IEEE Press, 492-501.
- [7] M.Mendonca, M.Branco, D.Cowan. *S.P.L.O.T. - Software Product Line Online Tools*. In *Proc. OOPSLA*, Orlando, USA, 2009.
- [8] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II." *IEEE Transactions on Evolutionary Computation*, 6(2), 182-197.
- [9] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the strength pareto evolutionary algorithm," in *Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems*. Athens, Greece, 2001, pp. 95-100.
- [10] E. Zitzler, S. Kunzli. "Indicator-based selection in multiobjective search." In *Parallel problem Solving form Nature*. Berlin, Germany: Springer-Verlag, 2004, Vol.3242, 832-842.
- [11] J.J.Durillo, A.J.Nebro. "jMetal: A Framework for Multi-Objective Optimization." *Advances in Engineering Software*, 42(10),760-771,2011
- [12] J.White, B.Dougherty, D. C.Schmidt, "Selecting highly optimal architectural feature sets with Filtered Carte-sian Flattening," *Journal of Systems and Software* 82 (8), 1268-1284, 2009.
- [13] J. Guo, J. White, G. Wang, J. Li, and Y. Wang. "A Genetic Algorithm for Optimized Feature Selection with Resource Constraints in Software Product Lines." *Journal of Systems and Software*, 84(12), 2208-2221.
- [14] M.Mendonca, A.Wasowski, K.Czarnecki. "SAT-based Analysis of Feature Model is Easy," In *Proceedings of the 13th International Software Product line Conference*, (San Francisco, California, 2009), Carnegie Mellon University, 231-240.

# Flexible Modeling and Product Derivation in Software Product Lines

Jorge Barreiros<sup>1,2</sup>

<sup>1</sup>Instituto Superior de Engenharia  
Instituto Politécnico de Coimbra  
Coimbra, Portugal  
jmsousa@isec.pt

Ana Moreira<sup>2</sup>

<sup>2</sup>Faculdade de Ciências e Tecnologia  
Universidade Nova de Lisboa  
Caparica, Portugal  
amm@fct.unl.pt

**Abstract**—Software Product Line development entails planned reuse of development assets for creating applications in a specific domain. SPL development can benefit from incorporating soft constraints in both Domain and Application Engineering. Increased expressiveness is attained and important domain knowledge that would otherwise be lost can be included, allowing improved configuration support to be provided. The stakeholders' goals for a specific product can also be represented with soft constraints, allowing incomplete and inconsistent specifications to be inputted and then be automatically processed and analyzed. The approach is supported by a tool, which is capable of detecting inconsistencies, identifying the required trade-offs and explaining them to the stakeholder, who can then make an informed trade-off decision.

**Keywords** - Feature Models; Software Product Lines; Soft Constraints; Configuration Support; Feature Consistency; Feature Interactions; Domain Knowledge

## I. INTRODUCTION

In Software Product Line (SPL) development, feature models annotated with domain constraints are often used to represent domain variability and commonalities [1, 2]. Domain constraints specify stringent conditions that must be verified by all products within the scope of the SPL. Automated support for the derivation of products can use this information to assist the developer in the creation of viable configurations of products. Regrettably, there are situations where these constraints clash with a reality that is not all black and white and stubbornly, albeit reasonably and incontrovertibly, refuses to conform precisely to the exacting specifications of our models.

A direct consequence of this state of affairs is that developers must choose to either drop domain constraints representing useful domain knowledge for the sake of flexibility, or accept that the potential scope of the SPL must be crippled, lest under-specification of constraints allow unintended incursions into the realm of unfeasibility. The first solution strips relevant information from the model impairing automation and support for product derivation: the developer bears the burden of creating meaningful and feasible configurations with reduced assistance. The second solution may be too conservative: by playing it safe, security is gained but flexibility is lost. Rather than agonizing over the choice of the lesser of two evils, we propose that the concept of *soft constraints* (SC) is embedded into SPLs to

address these concerns. SCs represent useful information that is relevant to the configuration of products, but is not of forcing nature and can be ignored if convenient. They serve the primary and important purpose of allowing automated advice and suggestions to be provided during product derivation. Although the developer may choose to act against this advice, he is making an informed rather than blind choice (as would be the case had those constraints been completely ignored and no support provided because of it). Collateral but relevant benefits include the possibility of identifying required trade-offs of conflicting stakeholders goals.

Using these techniques improves domain modeling capabilities and product derivation support. In this paper we present a method and tools, based on the soft constraint model described in [3], for supporting the use of feature models annotated with SCs in SPL development. Goals of our approach include the following functionalities:

- Provide configuration suggestions based on constraints and feature tree structure, in addition to configuration propagation and completion services.
- Identify conflicted features, that is, features for which antagonistic configuration advice is offered.
- Provide an explanation for conflicted features so that the user (stakeholder) can better understand the required trade-offs.

Hence, the main contributions include a detailed description of an inconsistency analysis algorithm and the development of the corresponding tool support for SC-assisted SPL configuration.

## II. BACKGROUND

### A. Software Product Lines

SPL development is an approach for developing software products within a specific domain of application [1, 2] that promotes planned reuse of core assets (e.g., specifications, code, models). Two distinct roles can be identified in SPL development: the *domain engineer* and the *application engineer*. The domain engineer defines the scope of the product line by identifying the common and variable features of the products to be created. He also provides the composable assets that are used to create those products. The application engineer is responsible for the derivation of

specific products. These are created by configuring the variation points according to stakeholders' specifications and composing the corresponding assets. Feedback is provided to the domain engineer so that evolution of the SPL scope and core assets is possible.

### B. Feature Modeling

The domain engineer typically represents product commonality and variability using *feature models*. These identify valid product configurations, that is, configurations corresponding to a variant that can be created by the application engineer. Feature models identify valid configurations by using a feature tree annotated with additional domain constraints. These can be represented graphically (e.g., linking dependent features with a dependency arrow). Feature models can be represented formally using logic expressions according to well-known transformations described in [4, 5]. A feature model is said to be consistent if at least one valid configuration exists.

To allow better modularization and management of complex products, a product configuration is often created by the application engineer in an incremental process called *staged configuration*, where features are progressively included or dropped from the product [6]. This allows automated feedback to be provided by interactive configuration tools that assist the application engineer in obtaining a valid configuration. Interactive configuration support for staged configuration automatically propagates user choices, by selecting or deselecting open features as required, according to the partial configuration inputted by the user and the feature model and its restrictions. This is achieved by computing the configuration domain after each selection made by the user, by identifying features that become dead (must be always deselected) and features that are common (must be always selected) in the current partial configuration. Techniques and tools to assist feature model configuration have been described in [5, 7].

### C. Hard and Soft Constraints

Domain constraints can be categorized as hard or soft. While hard constraints are mandatory and must be satisfied, SCs are used to indicate optional but preferential or common configurations [3].

While all hard constraints must be satisfied in a consistent feature model, the same does not apply to soft constraints. This opens up the possibility of valid models containing conflicting soft constraints, that is, soft constraints that may be impossible to satisfy simultaneously. In such cases, a trade-off between conflicting constraints exists and must be resolved. It is worth pointing out that hard constraints cannot conflict by definition; otherwise the feature model would become inconsistent.

Soft constraints can also be prioritized. This allows a wider range of scenarios to be represented, by making it possible to establish relative degrees of importance. In this case, satisfaction of higher priority constraints takes precedence over those of lower priority.

## III. CONFLICT ANALYSIS AND CONFIGURATION SUGGESTIONS

In this Section we describe the overall strategy for generating configuration suggestions and provide a detailed description of the configuration suggestion algorithm. In the SPL lifecycle, SCs originate in domain modeling or can be used to model user configuration goals. The overall strategy to automate conflict analysis is as follows:

1. Identify the set of active<sup>1</sup> SCs with highest priority.
2. Temporarily consider the SCs identified in step 1 as hard constraints and verify consistency of the solution, by checking the satisfiability of the combined expression of feature model, partial configuration and the active constraint set.
  - a. If the result is consistent, it is possible to find a solution that satisfies all active SCs. Proceed to generate configuration suggestions, by performing standard common and dead feature identification (but do not change the configuration). These are outputted as selection/deselection suggestions, correspondingly.
  - b. If the result is inconsistent, then it is not possible to simultaneously satisfy all active SCs. The inconsistency must be analyzed to explicitly detect the conflicting constraints and how they conflict in terms of feature (de)selection. This algorithm is described in detail in the sequel.
3. The user selects or deselects a feature belonging to the domain of the active SCs.
4. Propagate the choice performed in the previous step by selecting all features that became common and deselecting all features that became dead.

The inconsistency analysis algorithm is detailed in Figure 1. The overall strategy is based on identifying a minimal set of conflicting SCs by local search branching out from a consistent base model conjoining the feature tree, hard constraints and partial configuration. Neighborhoods of this model, including all  $k$ -permutations of SCs, are progressively explored, in increasing  $k$  order, until an inconsistent model is found. Ultimately, the process is ensured to end when the entire set of active SCs is considered (as these have already been identified as being inconsistent). Conflicts are identified by finding common and dead features (i.e., suggestions for selection and deselection) for different  $k$ -permutations of SCs. If conflicting advice is found, a conflict is identified.

By considering the effect of  $k$ -permutations of SCs in increasing order of  $k$ , the algorithm will find the simplest explanations of the conflict (that is, explanations involving the smallest number of SCs). This is motivated for efficiency

---

<sup>1</sup> We designate soft constraints as *active* if they have not been falsified or satisfied in the current partial configuration of a staged configuration process.

sake, as the majority of inconsistencies can be assumed to originate in the interactions between a reduced number of SCs. In scenarios where the assumption does not hold, efficiency is degraded but correct operation is still preserved.

Algorithm 1: Analyze Inconsistency

```

1 Function analyzeInconsistency
  input : Partial configuration config, feature model fm and
         constraints
2 done ← false
3 k ← 1
4 while ¬ done do
5   foreach k-combination S of active soft constraints do
6     dead ← computeDeadFeatures (fm · S,config)
7     common ← computeCommonFeatures (fm · S,config)
      /* associate soft constraint satisfaction to
      required (de)selection of features. */
8     if not empty dead then
9       foreach feature f in dead do
10        deselect [f] ← true
11        Add S to deselectionExplanation [f];
12     if not empty common then
13       foreach feature f in common do
14        select [common] ← true
15        Add S to selectionExplanation [f];
16     if ¬done then
17       done ← there exists f such that deselect [f] · select [f]
18     k ← k + 1
19 return
    computeMetric(deselectionExplanation,selectionExplanation)

```

Figure 1. Inconsistency Analysis Algorithm.

Our algorithm associates a metric to each feature, in the range  $-1..1$ , which can be interpreted as a deselection/selection recommendation. This function is generically dependent on feature trade-off information as described above. Any number of reasonable possibilities can be considered. Currently, we map the percentage of recommendations for selection vs. total number of recommendations into the  $-1..1$  interval. Other methods could easily be adopted, based on factors such as an importance weight attributed to of each SC.

#### IV. ILLUSTRATIVE EXAMPLE AND TOOL DEMONSTRATION

We developed a tool that supports SPL configuration. Rather than devising a feature model specifically tailored for illustrating our approach and supporting tool, we decided to adapt and use models freely available by industrial and academic sources in the SPLOT feature model online repository [7]. For this, we selected the Web Portal SPL feature model, a medium-high complexity feature model, in terms of number of features and constraint density, originally described in [9]. This feature model describes a product line for creating web servers with differentiated capabilities such as support for alternative protocols, advertisement and search services, security features, etc.

#### A. Annotation of Feature Model with Soft Constraints

Usage of SCs in the Domain Engineering phase of SPL development entails annotating a feature model with SCs. This is achieved by transforming relevant domain knowledge into appropriate set of SCs. In our example, Figure 2 shows the SCs corresponding to domain knowledge such as:

- Although popups are possible, they are not well tolerated by users and are discouraged.
- If banners are used, a Flash based version is advisable.
- It is recommended that authentication should be conducted through secure *https* connections.

Other concerns are also represented as well.

#### B. Application Engineering

File *StakeholderSpecification.xml* in Figure 2 represents the stakeholder’s desired configuration represented with SCs, with the *ms*, *data\_storage*, *advanced*, *dynamic* and *https* features selected and the *nttp* feature deselected. These SCs are composed with the annotated feature model. As seen in Figure 3, the tool provides configuration suggestions (+100% is a strong selection suggestion, -100% is a strong deselection suggestion) with conflicts being indicated by intermediate scores. In this case, a conflicted feature is *https*. The conflict stems from the impossibility of simultaneously satisfying constraints *cfg00* and *cfg05*, since domain restrictions indicate that *https* support precludes *ms* level performance. In this case, the stakeholder must resolve a trade-off by deciding to drop either one of *cfg00* or *cfg05*. Once this decision is made, configuration can proceed by following the tools suggestions: since no other inconsistencies are found, the configuration is successfully completed.

This example highlights how soft constraints have been used throughout the development process to successfully create a product, from an initial specification that is neither complete nor consistent. Automated support was able to detect the inconsistency in the specification and provided a clear indication of what the required trade-off is. Even though the stakeholder’s specification is not complete, a complete configuration with desirable properties was achieved.

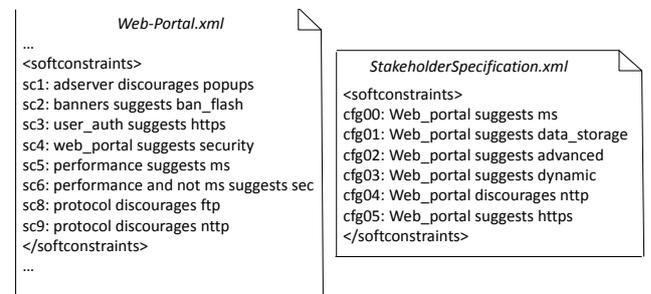


Figure 2. Capturing domain information and desired specification as SCs.

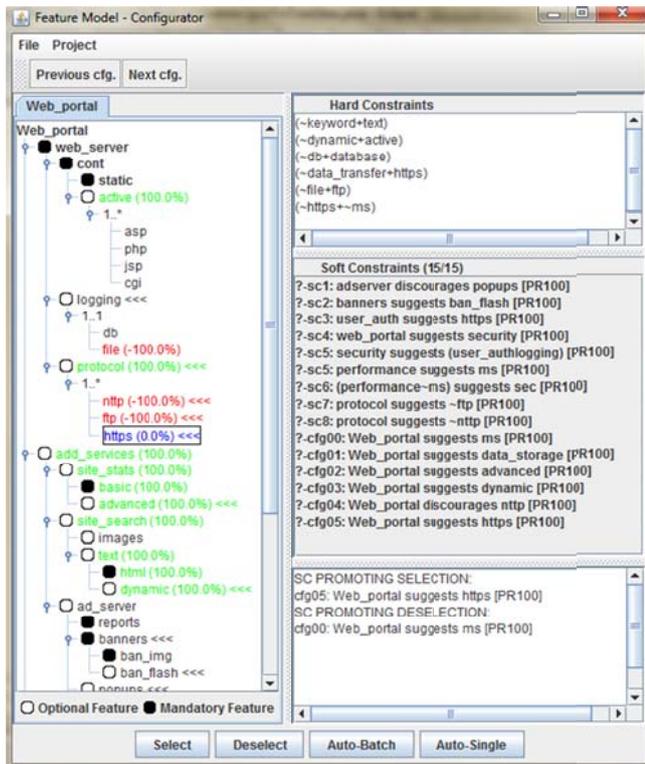


Figure 3. Tool support for SC-based configuration. On the feature tree view at the left side, suggestions for selection, deselection and conflicts are highlighted in green, red and blue, respectively. On the right side, from top to bottom, a description of the hard constraints, softconstraints, and conflict/trade-off information can be found.

## V. RELATED WORK

SCs have been largely ignored in SPL development until recent years. In [10], a fuzzy based model is described for including SCs in feature models. Configuration suggestions maximizing the coverage constraint satisfaction are made. However, no support is provided for facilitating alternative trade-off decisions or identifying alternative optimal configurations. In [3], the SC model considered in this work is described. Tool support is not presented and inclusion of the SCs in SPL development activities is not considered. Conflict detection is concerned with detecting universally unsatisfiable and incompatible SCs, while we consider incompatibilities that may arise during the configuration process. Work in [11] is mostly concerned with domain modeling with SCs, including typical patterns of application, and considers only the same types of constraints described in [3]. A preliminary variant of the inconsistency algorithm presented in this paper, based only on 1-permutations of constraints, was outlined by the authors in [12]. Our proposal differs by handling higher order conflicts and also addressing process, tooling and application. In [13], Czarnecki et al. describe probabilistic feature models. In these, features can be related by constraints that describe high probabilities of simultaneous selection of the corresponding features. This entails different semantics of

SCs and no specific support for conflict resolution is described.

## VI. CONCLUSIONS

This paper presented an approach for integrating the use of SCs in the SPL development process, with tool support being provided for relevant activities. This process allows richer representation of domain knowledge, allowing for improved configuration support in the form of automated configuration suggestions. Also, by representing stakeholder' goals for a product as SCs during the application engineering phase, not only automated identification of conflicting goals is possible, but additional information can also be provided to the stakeholders, aiding them in making the required trade-off decision. Our approach and corresponding tool support have been demonstrated by using published feature models available from a public repository. Future work may include the development of systematic domain analysis techniques with SC support. Another promising field of application of SCs that warrants further study is the representation of quality attributes and their interplay in feature models.

## REFERENCES

- [1] Clements, P.: *Software Product Lines: Practices and Patterns*. Addison-Wesley Professional (2001).
- [2] Czarnecki, K., Eisenecker, U.: *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley Professional (2000).
- [3] Barreiros, J., Moreira, A.: *Soft Constraints in Feature Models*. Proceedings of the Sixth International Conference on Software Engineering Advances, pp. 136–141. (2011).
- [4] Batory, D.S.: *Feature Models, Grammars, and Propositional Formulas*. Proceedings of the 9th International Conference on Software Product Lines, pp. 7–20. Springer, Rennes, France (2005).
- [5] Czarnecki, K., Wasowski, A.: *Feature Diagrams and Logics: There and Back Again*. Proceedings of the 11th International Conference on Software Product Lines, pp. 23–34. IEEE Computer Society (2007).
- [6] Czarnecki, K., Helsen, S., Eisenecker, U.W.: *Staged Configuration Using Feature Models*. 8th International Conference on Software Product Lines. pp. 266–283. Springer (2004).
- [7] Mendonca, M., Branco, M., Cowan, D.: *S.P.L.O.T.* Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications - OOPSLA '09, p. 761. ACM Press, New York, New York, USA (2009).
- [8] Janota, M.: *Do SAT Solvers Make Good Configurators?*, 12th Int. Conference on Software Product Lines, pp.191-195 (2008).
- [9] Mendonca, M., Bartolomei, T., Cowan, D.: *Decision-making coordination in collaborative product configuration*. 23rd Annual ACM Symposium on Applied Computing (2008).
- [10] Bagheri, E., Noia, T. Di, Ragone, A., Gasevic, D.: *Configuring Software Product Line Feature Models Based on Stakeholders' Soft and Hard Requirements*. 14th International Conference on Software Product Lines, pp. 16–31. Springer (2010).
- [11] Barreiros, J., Moreira, A.: *Soft Constraints in Feature Models : An Experimental Assessment*, International Journal On Advances in Software, vol.. 5, number 3 and 4, pp. 252–262 (2012).
- [12] Barreiros, J., Moreira, A.: *Configuration support for feature models with soft constraints*. 28th ACM Symposium on Applied Computing, pp. 1307–1308 (2013).
- [13] Czarnecki, K., She, S., Wasowski, A.: *Sample Spaces and Feature Models: There and Back Again*. Proceedings of the 12th International Conference on Software Product Lines, pp. 22–31. IEEE Computer Society (2008).

# A Critical Embedded System Product Line Model-based Approach

Paulo G. G. Queiroz<sup>\*†</sup>, Rosana T. V. Braga<sup>\*</sup>

<sup>\*</sup>Universidade de São Paulo – Instituto de Ciências Matemáticas e de Computação  
São Carlos-SP, Brasil

Email: pgabriel,rtvb@icmc.usp.br

<sup>†</sup>Universidade Federal Rural do Semi-Árido – Departamento de Ciências Exatas e Naturais  
Mossoró-RN, Brasil

Email: pgabriel@ufersa.edu.br

**Abstract**—The development of Critical Embedded Systems (CES) like Unmanned Aerial Vehicles (UAV) is complex because it needs to ensure a high degree of quality, with affordable cost and delivery time. It is also necessary to ensure security since failures in this type of system can lead to catastrophic results. In this sense, a Model-Driven Development (MDD) approach presents itself as a good alternative to the traditional development because coding complexity will be reduced by the use of high level models. In addition, it avoids the introduction of coding errors by human programmers, since the critical code will be built automatically through models transformation. From another perspective, Embedded Systems Development can benefit from Software Engineering techniques like Product Lines to reduce costs and time-to-market. While other works propose the use of Product Line techniques to improve Embedded Software development, we propose a Product Line approach to the whole Critical Embedded System development life cycle, including hardware variability management. Therefore, this paper proposes a Critical Embedded System Product Line Model Based approach, which aims to reduce the above mentioned challenges. The development approach proposes a Domain Engineering and Application Engineering focused on the system, with both software and hardware. To illustrate the proposed approach we include some artifacts from a case study in the UAV domain.

## I. INTRODUCTION

Embedded systems development approaches are constantly evolving to meet the growing need for new and more complex applications. In the embedded systems domain we can highlight CES, which are embedded systems whose failure could result in loss of lives or on significant environmental or property damage. CES are common in medical devices applications, aircraft flight control systems, weapons, and nuclear systems [1]. This kind of systems breaks with traditional systems engineering, as they bring together technical requirements such as: criticality, need for low power or fuel (autonomy), reactivity, robustness, scalability with guarantee of quality and functionality at affordable costs [2].

Since hardware evolves fast, the new possibilities created by their evolution increase users expectations for new functionalities to be realized and integrated into an embedded system. However, software development does not cope with the hardware evolution and consequently system development either. To build CES to meet the mentioned requirements in an acceptable time-to-market, we need new efficient and

flexible development methodologies that can reduce design complexity.

Among several possibilities, Software Product Line Engineering seems a promising one, as it brings great benefits such as: reducing system costs, because the systems are designed collectively and generated automatically or semi-automatically through the composition of reusable artifacts; time-to-market reduction, since this technique allows to design systems in such a way to predict the possible variabilities; and increasing system reliability, because the systems are assembled from reusable and extensively tested resources and many other advantages [3]. However, for the effective use of this technique in the CES domain we should adapt it for System Product Line Engineering (SPLE), i.e., to apply product line techniques to design the system and not only the software. In conjunction with SPLE we propose the use of model-driven development (MDD) techniques to increase the level of abstraction by means of models and to allow automatic generation of system product line artifacts through models transformations.

Therefore, this paper aims to define and present a complete approach for the design of CES using a combination of both System Product Line and MDD techniques. The differential of the proposed approach are: the use of MDD in both Domain and Application Engineering, the management of both software and hardware variabilities, and the control of software and hardware variability dependencies. For an effective use of MDD we propose the use of the MARTE profile [4] in conjunction with SysML [5] and a simple variability management stereotypes proposed here. Finally, to illustrate the various concepts presented in the proposed approach we include some artifacts from a case study in the UAV domain.

The rest of this paper is organized as follows: Section II presents a background summary of Product Line Engineering and Model-driven Development; Section III illustrates some related works; Section IV presents the proposed approach, followed by a description of the examples by means of a UAV product line case study; lastly, Section V presents the conclusions of this paper.

## II. BACKGROUND

According to Northrop et al. [3], a Software Product Line is a set of software systems that share common and managed features satisfying the specific needs of a particular market segment or mission and that are systematically developed from a common set of core assets. The products of a PL distinguish from each other in terms of features, which are user-visible aspects or characteristics of a software system or systems [6].

On the other hand, in Model-Driven approaches, the software complexity concentrate on high level models and not in the code, which can be automatically generated from the models. Furthermore, one can improve the system quality with the use of V&V (Verification and Validation) methods [7]. According to model-based approaches, models become part of the final product and most of the development complexity shall belong to the transformations that should be used to automatically or semi-automatically produce code (or pieces of code) by transformation tools.

These techniques offers an interesting solution to the challenges in the CES domain because they have great synergy, since they propose automation, raising the level of abstraction and reuse. Additionally, they are being widely used in various academic research and industrial projects as discussed in the next Section.

## III. RELATED WORKS

While there is a large number of researches that make use of either PL and embedded systems or MDD and embedded systems, due to space limitations, it is not possible here to give an exhaustive description, thus we only provide a brief summary of some works that make use of PL and MDD for the CES domain, similarly to the approach proposed in this work.

Eriksson et al. [8] present a work that focuses on the system (hardware and software) and introduce a feature-based and use case approach for the process of system requirements. While the focus of these authors is only the step of requirements, the work presented in this paper focuses on the entire System Product Line development life cycle.

In the thesis described in Habli [9], the author defines and evaluates a model-based approach to ensure systems and process in a CES Product Line. The approach is based on a safety metamodel that captures the necessary dependencies between the safety plan, the safety assessment and the developed artifacts in the PL. The focus of that work is to ensure systems and processes.

The work presented by Polzer et al. [10] is concerned with the variability in control systems software. In their work, they present a model-based PL engineering process using Rapid Control Prototyping system combined with MDD techniques. The authors modularize the components parameterization in a separate setup, which is isolated from the model that defines the behavior of the controller. The authors use Simulink for the control modeling and automatic code generation and Pure::variants [11] to obtain the products, which are proprietary tools.

Regarding the development of critical embedded applications product lines, there are approaches such as ProLiCES [12] and SyMPLES [13]. ProLiCES creates a parallel path in the process to handle the PL domain engineering and also proposes the use of Matlab/Simulink as MDD technique, which limits requirements analysis and concentrates the MDD only in one step of the process. SyMPLES is an approach for PL application in embedded systems through the extension of SysML language to include variability together with a development process, but in this study the authors do not distinguish between the characteristics of hardware and software and focus on the use of SysML for the architecture description. They both show the application of the approach in the development of a UAV PL.

In short, from the observation and analysis of existing approaches, we note that: some of them require a lot of previous knowledge like the use of non standard profiles, which hinders its adoption; other approaches do not define a specific treatment for hardware variability and its impact in software variability; others are based on the use of proprietary software, such as Matlab/Simulink; and, yet other approaches use MDD in limited parts of the development, specially from project to code or just in application engineering phase.

## IV. PROPOSED APPROACH

The approach we propose in this paper is different from the above mentioned related works, as it focuses on: a simplified variability management, which addresses both hardware variability with its underlying software requirements dependences and vice versa; the use of MDD techniques, like automatic generation of hardware descriptions and embedded software from high level models, for rapid design and specification of CES; models based on UML and the use of SysML and MARTE profiles, because it is the most widely used modeling language, it is easier to understand and is considered as a standard by various free existing tools; the use of MDD in both Domain Engineering and Application Engineering phases, with focus on model-to-model transformations in requirements, analysis and design stages and not only model-to-text transformations in later development stages or just during Application Engineering, by enabling the automatic generation of SPL members.

A brief overview of the proposed approach, is illustrated in Fig 1. As well as the classical PL approaches, the proposed approach has two phases, Domain Engineering and Application Engineering. This approach first differs from the others in the abstraction levels proposed for the Domain Engineering: a system abstraction level, a hardware abstraction level and a software abstraction level, which are detailed later on this section.

Observe that hardware variability could impact directly on software requirements and vice versa. For example, consider the following system requirement in a UAV product line: *the system should allow the user to choose between broadcast the images to the ground control station at real time or to record a video (in flash memory, for example)*. In that case, the

UAV hardware must include a camera. Moreover, for each new sensor added, their corresponding software drivers must also be added. Another highlight is the continuous feedback in the artifacts repository, in which we can store any kind of artifact from both hardware or software types. This feedback can come from updates in Domain Engineering or from new different requirements elucidated from new members modeled in Application Engineering. Also noteworthy is the feedback that may exist inside the system, hardware or software abstraction levels, and among these levels. The feedback is represented by both dashed arrows and double-headed arrows.

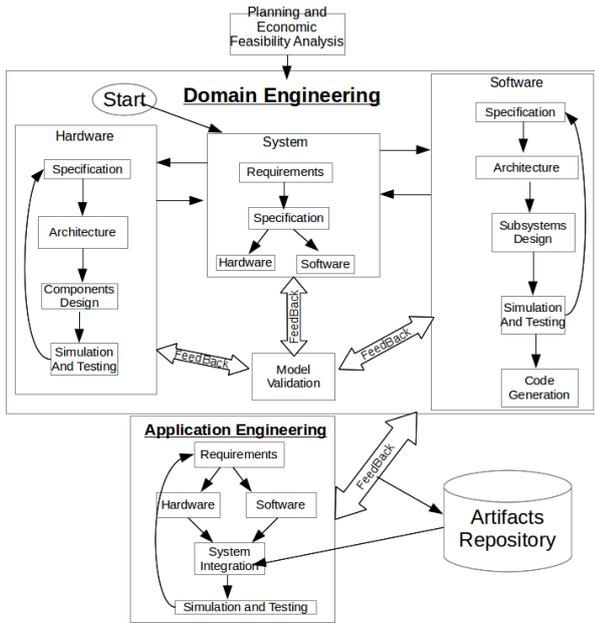


Fig. 1. Overview of the proposed approach.

As previously mentioned, the proposed approach is based on MDD and proposes the use of UML as a modeling language. Therefore, for implementing model transformations, it is necessary to use: UML MARTE and SysML profiles in a controlled manner to avoid conflicts as explained below. For efficient use of these modeling languages we need to use one or more tools with MARTE and SysML metamodel support.

### A. Domain Engineering

The main purpose of Domain Engineering is to produce reusable artifacts that are stored in the SPL artifacts repository and are used to derive SPL members during Application Engineering. The artifacts repository includes artifacts that should be present in all SPL members, called core artifacts; artifacts that are present only in some SPL members, known as optional artifacts; and even artifacts that have different versions required by different SPL members, known as alternative artifacts. The distinction between these types of artifacts must be detailed in the models listed below through the use of mandatory, optional and alternative stereotypes respectively, like proposed in [14].

Before the Domain Engineering takes place, it is performed a development activities planning, as well as an economic feasibility analysis of the SPL, which will indicate whether or not it is worth to be developed. If the SPL is feasible, then we start the Domain Engineering by modeling requirements in the system abstraction level, as detailed below. It is out of the scope of this work to propose domain analysis techniques, as they can be easily found in the literature. So, existing techniques such as those mentioned in the survey by Prieto-Diaz and Arango [15] can be used.

The Domain Engineering is performed in three abstraction levels and starts mainly from the observation and analysis of existing system on the domain of interest: the system level, where we define the system as a whole by means of requirements definition and specification; the hardware level, in which the hardware specification and hardware variability management take place in conjunction with architecture definition, design of the components and simulation; and, the software level, where we perform the software variability specification and management, architecture definition, subsystems design, simulation, testing, and code generation. Since the proposed approach focuses on CES, various activities of validation and verification permeate the activities of each level. It is important to notice that at the hardware and software abstraction levels a more detailed specification takes place by the eventual allocation with schedulability and underlying model transformations (model-to-model and model-to-text transformations), which are used to bridge the gap between these abstract design models and subsequent design phases, such as verification, hardware descriptions of modeled targeted architecture and generation of platform specific embedded software from independent architecturally software specifications.

1) *System abstraction level:* The system abstraction level starts with the SPL requirements definition by means of a domain analysis, in which the user initially specifies the requirements related to the system product line. For this purpose a SysML requirements diagram with the distinction between functional and non-functional requirements including a simple tagged stereotypes mentioned above to define mandatory, optional and alternative requirements is recommended. Refinements of this diagram can be done as long as necessary, specially after hardware and software specification, to produce a validated artifact that can be included in the SPL repository. In the Application Engineering, when concrete products are instantiated, this document will be matched against the specific product requirements. As illustrated in Figure 1, the artifact repository is update during SPL life cycle for both Domain Engineering and Application Engineering. So, if concrete products have new requirements not covered by the SPL, the requirements diagram can be further reviewed to include them.

After the requirements identification, we propose the definition of SPL features based on the requirements model, by means of feature model [16]. This model also defines the SPL mandatory, optional and alternative features, as well as the constraints that must be validated during the composition of the products. The restrictions are mainly represented by the

dependencies between features.

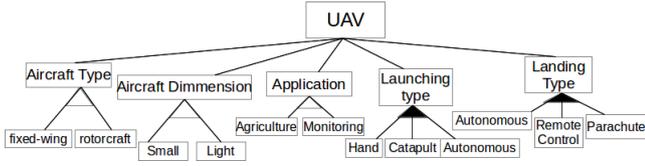


Fig. 2. A portion of the feature model for the UAV Product Line – system abstract level.

In Figure 2, we illustrate the system abstraction level UAV feature model. It is important to notice that this is a initial high level feature model and included both hardware and software features that should be refined in both hardware and software abstraction levels. The decision regarding the implementation of some of these features in hardware or software is done at a later point in the project, which can be even possible at the Application Engineering phase, as it depends on the results of simulations in conjunction with specific restrictions of each product, like for example, execution time, cost and other factors. Some of the features should be implemented and stored in a software/hardware artifact repository. Regarding the repository to store hardware artifacts, we refer in a logical level, to a hardware models repository.

In the next step the system requirements are converted into use case scenarios and into a system domain model. The use case is described using traditional UML Use Case Diagrams plus the stereotypes to represent mandatory, optional and alternative use cases. The system domain model can be modeled using a class diagram, in which the concepts are represented by pseudo-classes, where we also propose the use of the above mentioned stereotypes. These two modeling concepts are strongly related to the functional high level specification described subsequently. While the use case is needed to obtain a SysML block diagram, the domain model is used to communicate with domain experts for a better understanding about the domain, for validating the specification and for a future definition of a domain specific language by means of UML profile.

To finalize the system abstraction level, each use case is converted into a SysML block (or internal block), for example by applying the MADES methodology [2], with the difference that a mandatory use case is converted to a mandatory block, an optional use case is converted to an optional block, and an alternative use case is converted to an alternative block. After including all the developed artifacts in the repository, we can continue the SPL development by going to hardware or software abstract levels or even to both in parallel. Even though some authors consider MARTE and SysML profiles incompatible, by using the MADES methodology we avoided conflicts related to the two profiles. SysML is used for initial requirements and functional description, while MARTE is utilized for the enriched modeling of the global functionality and execution platform/software modeling [2].

2) *Hardware and software abstraction levels:* In the next step of domain engineering we lower the abstraction level, with

the possibility of starting with hardware or software abstraction level, or both in parallel. In both hardware and software abstraction levels, the requirements specification focuses on variability management through feature diagram refinement. It is not necessary to write another requirements model from scratch, but it might be necessary to update the previous one to maintain traceability between artifacts.

It is important to keep in mind that hardware variability management should concern the impact evaluation of hardware variabilities on software requirements, and also software variability management should concern the impact evaluation of software variabilities in hardware requirements. To manage this impact we propose the use of a dependence matrix, which relates every possible impacting hardware variabilities to the corresponding software change variabilities and, on the other hand, relate every possible impacting software variabilities to the corresponding hardware change variabilities, as illustrated in Figure 3. This matrix can be configured as OCL constraints [17] to validate product instantiation in application engineering. Observe that for example **automatic landing** software feature requires **Landing gear** hardware feature.

Hardware Variabilities	Dependence	Software Variability
Landing gear	Require	Automatic landing
Batery sensor	Require	Automatic abort the mission
R/C receiver	Require	Manual Control

Fig. 3. A portion of the dependence matrix for the UAV Product Line

Once the refined feature management is completed, the designer can move to the partitioning of the system in question: depending upon the requirements and resources in hand, he or she can determine which part of the system needs to be implemented in hardware or software. It is possible, although it substantively increases SPL cost, to increase the number of possible SPL members derivation by implementing system features in a redundant way, ie, whenever possible, maintain in the repository features implementations in both hardware and software. Thus, it becomes part of the Application Engineering, whenever possible, to decide if the features implementation component should be integrated in the product by software or hardware.

For a description of the different steps related to each design level by means of MARTE concepts, see the work of Quadri et al. [2], which can be adapted to this approach by adding mandatory, optional and alternative stereotypes to the target models for the hardware and software specification, architectural definition, components and subsystems design and simulation. It is also important to perform validation activities in every model to improve safety. At the end of this phase our repository contains all the artifacts and the domain engineering is concluded.

## B. Application Engineering

Application Engineering corresponds to a product line target system configuration by assembling reusable artifacts from the repository. This step is the responsibility of the application

engineer, who is responsible for eliciting system requirements, verifying that the system being developed belongs to the SPL and, if so, selecting the desired features for the target system. The feature selection defines which system artifacts should make part of the new system. From this configuration, the target system is assembled from reusable components developed in the Domain Engineering. To end this step, it is necessary to conduct simulation and testing also on the target system to validate it. The software system instantiation can be automatic by configuring the SPL in an application generator like Pure::variants [11] or other available tools.

## V. CONCLUSIONS

This paper aimed to present an approach for the development of critical embedded systems product lines with the use of MDD in both Domain and Application Engineering, as well as software and hardware variability management. To fulfill this objective we have used a subset of UML profiles like SysML, MARTE and a simple variability management profile combined with the feature modeling and a dependency matrix to simplify the challenges of critical embedded system product line development. The use of the proposed approach in the CES domain can bring the benefits of PL and MDD techniques like reducing system costs and time-to-market and increasing system reliability. The steps, models and concepts from the proposed approach have been applied by a UAV product line case study that was not shown due to lack of space. For future work we propose to evaluate the use of other MDD techniques during domain engineering to increase the advantages of model transformations, to facilitate the adoption of the proposed approach, and for automatic generation of target system models.

## REFERENCES

- [1] J. C. Knight, "Safety critical systems: challenges and directions," in *Proceedings of the 24th International Conference on Software Engineering*, ser. ICSE '02. New York, NY, USA: ACM, 2002, pp. 547–550. [Online]. Available: <http://doi.acm.org/10.1145/581339.581406>
- [2] I. R. Quadri, A. Sadovykh, and L. S. Indrusiak, "Mades: A sysml/marte high level methodology for real-time and embedded systems," in *ERTS2 2012: Embedded Real Time Software and Systems*, 2012.
- [3] L. M. Northrop, P. C. Clements, F. Bachmann, J. Bergey, G. Chastek, S. Cohen, P. Donohoe, L. Jones, R. Krut, R. Little, J. McGregor, and L. O'Brien, "A Framework for Software Product Line Practice, Version 5.0," 2009. [Online]. Available: <http://www.sei.cmu.edu/productlines/framework.html>
- [4] OMG, "Uml profile for marte: Modeling and analysis of real-time embedded systems," 2009.
- [5] O. M. Group, *OMG Systems Modeling Language (OMG SysML), V1.3, OMG document number formal/2012-06-01*, 2012.
- [6] K. C. Kang, J. Lee, and P. Donohoe, "Feature-Oriented Product Line Engineering," *IEEE Software*, vol. 19, no. 4, pp. 58–65, 2002.
- [7] L. Belategi, G. Sagardui, and L. Etxeberria, "Model based analysis process for embedded software product lines," in *Proceedings of the 2011 International Conference on Software and Systems Process*, ser. ICSSP '11. New York, NY, USA: ACM, 2011, pp. 53–62. [Online]. Available: <http://doi.acm.org/10.1145/1987875.1987886>
- [8] M. Eriksson, J. Börstler, and K. Borg, "Marrying features and use cases for product line requirements modeling of embedded systems," in *Proceedings of the Fourth Conference on Software Engineering Research and Practice in Sweden*, 2004, pp. 73–83.
- [9] I. M. Habli, "Model-based assurance of safety-critical product lines," Ph.D. dissertation, York, UK, 2009.
- [10] A. Polzer, S. Kowalewski, and G. Botterweck, "Applying software product line techniques in model-based embedded systems engineering," in *Model-based Methodologies for Pervasive and Embedded Software (MOMPES 2009), Workshop at the 31st International Conference on Software Engineering (ICSE 2009)*, vol. 0. IEEE Computer Society, May 2009, pp. 2–10.
- [11] P. Systems, "pure::variants," 2012.
- [12] R. T. V. Braga, K. R. L. J. C. Branco, O. T. Júnior, P. C. Masiero, L. de Oliveira Neris, and M. B. 0002, "The prolices approach to develop product lines for safety-critical embedded systems and its application to the unmanned aerial vehicles domain." *CLEI Electron. J.*, vol. 15, no. 2, 2012. [Online]. Available: <http://dblp.uni-trier.de/db/journals/cleiej/cleiej15.html#BragaBJMNB12>
- [13] R. F. Silva, V. H. Fragal, E. A. de Oliveira Junior, I. M. de Souza Gimenes, and F. Oquendo, "Symples - a sysml-based approach for developing embedded systems software product lines." in *ICEIS (2)*, S. Hammoudi, L. A. Maciaszek, J. Cordeiro, and J. L. G. Dietz, Eds. SciTePress, 2013, pp. 257–264. [Online]. Available: <http://dblp.uni-trier.de/db/conf/iceis/iceis2013-2.html#SilvaFJGO13>
- [14] H. Gomaa, *Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 2004.
- [15] R. Prieto-Diaz and G. A. (eds.), *Domain Analysis and Software Systems Modeling*. IEEE Press, 1991.
- [16] K. C. Kang, S. Kim, J. Lee, K. Kim, G. J. Kim, and E. Shin, "Form: A feature-oriented reuse method with domain-specific reference architectures," *Annals of Software Engineering*, vol. 5, pp. 143–168, 1998.
- [17] OMG, "OMG Object Constraint Language (OCL), Version 2.3.1," Object Management Group, January 2012. [Online]. Available: <http://www.omg.org/spec/OCL/2.3.1/>

# Creating Proprietary Terms Using Lightweight Ontology: A Case Study on Acquisition Phase in a Cyber Forensic Process

Tamer Fares Gayed, Hakim Lounis

Dépt. d'Informatique  
Université du Québec à Montréal  
Succursale Centre-ville, Montréal QC H3C 3P8, Montréal,  
Canada  
gayed.tamer@courrier.uqam.ca lounis.hakim@uqam.ca

Moncef Bari

Dépt. de Didactique  
Université du Québec à Montréal  
Succursale Centre-ville, Montréal QC H3C 3P8, Montréal,  
Canada  
bari.moncef@uqam.ca

**Abstract**—Terms and their meaning connections provided by the Resource Description Framework (RDF) present nowadays the standard mechanism for Linking Data (LD) on the web. All the existing terms, whether they are built-in terms (imported from well-known vocabularies on the semantic web) or proprietary terms (custom terms created by data publisher) can be used to describe and link different things in the world through RDF statements, and by applying the general architecture of the World Wide Web known as Linked Data Principles (LDP). Sometimes, these existing terms are not enough and adequate to describe a particular data set; more proprietary terms need to be created and developed in a dedicated vocabulary using lightweight ontology of LD. The latter uses the constructors of Resource Description Framework Schema (RDFS) and little features from Web Ontology Language (OWL) to create new proprietary terms describing such data set. This idea is depicted in this paper through a phase retrieved from a Cyber Forensic (CF) process, called acquisition phase, where different forensic tasks need to be described using new proprietary terms. This paper explains how these new proprietary terms can be created and published using the constructors of the lightweight ontology to describe this forensic phase.

**Keywords**—*Linked Data; Linked Data Principles; Resource Description Framework Schemas; Web Ontology Language; Proprietary Terms; Cyber Forensic; State Preservation.*

## I. INTRODUCTION

Today, the WWW has radically altered the way to share information [1][3]. The interrelation is not just between documents, but it has evolved to link the data within these documents (i.e., Linked Data-LD) using the same aspects of web architecture (URI [4] and Hyper Text Transfer Protocol-HTTP [5]). This data is described and represented using different terms imported from well-known vocabularies on the semantic web, or from proprietary terms retrieved from custom vocabularies created by the data publisher (i.e., such terms representing real objects and abstract concepts in the world). HTTP URIs are not only used to identify and interlink web documents but also interlink such terms, the fact that allows the latter to be dereferenceable/resolvable (i.e., it means that HTTP clients can look up the URI using the HTTP and retrieve a description of term/resource that is identified by this URI).

Sometimes, the existing terms are not adequate, or there are no existing ontologies (vocabularies) containing terms describing a particular data set. Thus, new proprietary terms

need to be developed in a dedicated vocabulary, applying the features of RDFS [9] and OWL [10] to describe this particular data set. However, before creating a new custom term, some aspects (criteria) should be taken into consideration<sup>1</sup>:

1. Search for terms from widely used vocabularies that could be reused to describe the domain in interest. If the widely deployed vocabularies do not provide the required terms to describe such domain, so new terms should be defined as proprietary terms.
2. When you define a new term, you need to have a namespace that you own and control (i.e., unique namespace), in order to mint your new terms to this domain/namespace.
3. When you create new terms, you have to map these terms to those in existing vocabularies.
4. Apply the LDP to your new terms by using the web technology stack (HTTP, URL, and RDF [8]) and this task takes place along the publication process, starting from the identification of terms until their publication.
5. Label and comment each term you create.
6. If your term is a property (predicate), you have to define its *domain* and *range* using the constructors of RDFS and do not overload your new term with ontological axioms.
7. If at later time you discover that another term was enough, an RDF link should be set between the new created term and the existing one.

This paper resumes the work published in [7] (see Figure 1). In this work, a Cyber Forensics-Chain of Custody framework (*CF-CoC*) was proposed to transform the *CoC* from tangible documents into electronic data (*e-CoC*). This framework provided several layers to perform such task. The first two layers were about the creation of custom terms (i.e., forensic/victim terms) using lightweight ontology. This paper discusses how such terms are identified and defined using lightweight ontology in order to describe forensic data in order to be published later on the web of data.

The current paper is organized as follows: next section discusses the state of the art and the map between lightweight ontology and *CF*. Section 3 presents the different constructors of lightweight ontologies; constructors of RDFS and OWL. Section 4 explains how the first two layers of the

---

<sup>1</sup> <http://linkeddatabook.com/editions/1.0/>

framework are designed and implemented, and how some forensic terms of the acquisition phase are defined. Finally, the last section concludes and summarizes this work.

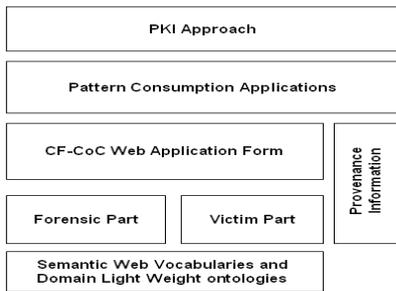


Figure 1 CF-CoC Framework [7]

## II. STATE OF THE ART: ONTOLOGICAL CONCEPTS AND FORENSIC PROCESS

Publishing data on the web passes by three phases. It starts with identifying terms in the domain of interest. These terms are the things whose properties and relationships will be used later in the publication of data. Second phase, the identified terms are defined using different constructors of RDFS [9] and OWL [10], and uniquely named by HTTP URIs (i.e., as being explained in last section, a new term should be minted to a domain controlled and owned by the creator of the term). Finally, once terms are identified, defined, and named using HTTP URIs, they are then published on standardized contents formats. This format is the RDF [8] that provides a generic data model composed of a triple containing three slots: subject, predicate, and object, where the defined terms occupy one or more slot(s) in this triple. Because the RDF model does not provide any domain-specific terms for describing classes of things in the world and their interrelation, it allows the combination of schema languages such as RDFS/OWL [9][10], and different mixture of distinct RDF vocabularies. Publication of these created terms takes place through a publication form (i.e., the third layer of the framework).

Although, the process of selecting terms is subjective and depends on the term creator (i.e., we may have two creators selecting and identifying two different terms describing the same concept in the real world), this does not affect the quality of terms being published, because the LDP on the web of data make them self-descriptiveness. The latter advantage is due to two reasons:

- LDP with naming using HTTP/URIs, offer a dereferenceable nature to the term, so that any LD consumption applications can look up the RDFS/OWL definitions and retrieve more information about such term [11].
- LDP with some schema constructors (i.e., OWL) can map a new term to existing terms from well-defined vocabularies in the form of RDF links [12].

In a domain like *CF*, it is scarce to find forensic terms or well known vocabularies describing it, because it is still in its infancy and development. The most related work to define

an ontology in *CF*, was published in [13], where an ontological model (i.e., with small ‘o’) was created for outlining *CF* tracks in the education process. Its aim was only to construct a hierarchical structure for classification of certification domains (i.e., the best convenient vocabulary to be used by the web of data to construct such type of ontology is the Simple Knowledge Organization System - SKOS<sup>2</sup>). Thus, *CF* is a domain that requires the definition of new proprietary terms. In the *CF-CoC* framework, the need of creating new terms emerged from the objective to describe and represent tangible documents of *CoC* into electronic data.



Figure 2 Abstract model of a Forensic Process

The role player is the one who is responsible to prepare the tangible *CoC* for each forensic phase to answer the 5Ws (What, When, Where, Why, Who) and 1H (How), and to describe all the accomplished tasks performed during his forensic investigation (Figure 2). He will use the *CF-CoC* framework to represent *CoC* by defining new proprietary terms and publish such information on the web of data in RDF format. Any forensic process contains a set of phases, where each phase is assigned to a role player. Each forensic phase contains a set of forensic tasks; each task can be described using a set of terms representing the forensic information. However, before discussing how the role player can use the *CF-CoC* to generate the *e-CoC*, it is necessary to explain how the forensic information can be ontologically mapped (Figure 3).

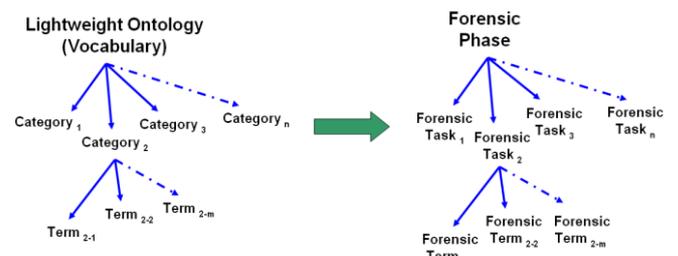


Figure 3 Mapping between Ontological Concepts and a Cyber Forensic Phase.

As shown in Figure 3, each forensic phase will have a corresponding lightweight ontology. Each lightweight ontology has a set of  $n$  categories, which will be equivalent to  $n$  forensic tasks. A category in the vocabulary should be described using a set of  $m$  terms. These terms are the proprietary terms describing a forensic task.

This work considered the acquisition phase as an example to elaborate the idea of creating lightweight ontology with new proprietary forensic terms. [14][15][16][17][18].

<sup>2</sup> <http://www.w3.org/2004/02/skos/>

**Acquisition Phase:** this phase is about acquiring digital evidences from digital suspected devices (e.g., small-scale devices, large-scale devices, etc.). It contains three forensics tasks: state preservation, recovering, and copying. The role player of this phase is the first responder [17][14].

- **State preservation:** the first task is saving the state of the digital device under question [19], by seizing the machine containing the suspected device.
- **Recovery:** after seizing the suspected device, the role player tries to recover all deleted files on the device, especially the system files that records valuable details about this suspected device.
- **Copy:** after recovering the deleted files, the first responder takes copy from the suspected device to avoid tampering and alteration.

From the above forensics tasks, the role player can start to determine different terms to describe his CoC. In this paper, the authors are those who identified some terms of acquisition phase (see Table 1).

TABLE-1 PROPRIETARY TERMS OF PRESERVATION TASK

	Term name	Type
<b>T- Box</b>	First_responder	Class
	Role_player	Class
	Acquisition	Ontology
	Digital_media	Class
	preserve	Property
	preservedby	Property
<b>A- Box</b>	SN	Property
	Jean-Pierre	Subject/Object
	PDA-device	Subject/Object
	OG-4023-32-362	Object

The T-Box, A-Box, and the column type will be discussed in next section.

### III. LIGHTWEIGHT ONTOLOGY: RDFS AND OWL CONSTRUCTORS

RDF is the standard format to create LD and it is sufficient to use the constructors of RDFS and a little feature of OWL. Combination of constructors from both vocabularies represents the lightweight ontology of RDF, known as RDFS++. Next subsections highlight all the RDFS constructors and some OWL primitive constructors to construct the first two layers of CF-CoC framework.

The RDFS and OWL constructors are classified according the term type (*rdfs:class*, or to be a property *owl:objectProperty*). This definition takes place before the term will be used (i.e., before its publication- T-Box). Later, the defined terms are used to describe and publish different data (A-Box, Assertion Box) [23]. The type of the term also determines its slot position during publication.

#### A. RDFS Constructors

The RDFS constructors are used to define terms and their relationships. Consider the term in question is named *X*.

TABLE-2 RDFS CONSTRUCTORS FOR PROPERTY AND CLASS TERMS

If X is a term of type ( rdf : type ) Property ( rdfs : Property / owl : ObjectProperty )	
<i>rdfs : subPropertyOf</i>	When the term <i>X</i> is of type <i>property</i> it can be also a sub property of another <i>property</i> term. The <i>subPropertyOf</i> of a property term is a term of type <i>Property</i>
<i>rdfs : range</i>	The <i>range</i> of a property term is always a <i>Class</i> . A <i>range</i> of a property term <i>X</i> states that the <b>object</b> slot of the <i>X</i> (i.e., where <i>X</i> is a predicate, because <i>X</i> is a <i>property</i> ), interpreted by a reasoners as an instance of said <i>range</i> of <i>X</i>
<i>rdfs : domain</i>	The <i>domain</i> of a property term is always a <i>Class</i> . A <i>domain</i> of a property term <i>X</i> states that the <b>subject</b> slot of the <i>X</i> (i.e., where <i>X</i> is a predicate, because <i>X</i> is a <i>property</i> ), interpreted by a reasoners as an instance of said <i>domain</i> of <i>X</i>
If X is a term of type ( rdf : type ) Class ( rdfs : Class )	
<i>rdfs : subclassOf</i>	When the term <i>X</i> is of type <i>Class</i> , it can be also a sub class of another <i>Class</i> term. The <i>subclassOf</i> of a property term is a term of type <i>Class</i>
Common Constructors between Property and Class terms	
<i>rdfs : comment</i>	Any term should have a <i>comment</i> . A <i>comment</i> is used to provide a human-readable description of a resource. <i>Comment</i> is an instance of <i>rdf : Property</i>
<i>rdfs : label</i>	Any term should have a <i>label</i> . A <i>label</i> is used to provide a human-readable name for a resource. <i>Label</i> is an instance of <i>rdf : Property</i>

#### B. OWL Constructors

The primitive selected from the OWL are mainly used to map between class and property terms.

TABLE-3 OWL CONSTRUCTORS FOR PROPERTY AND CLASS TERMS

If X is a term of type ( rdf : type ) Property ( rdfs : Property / owl : ObjectProperty )	
<i>owl : equivalentProperty</i>	This constructor is used to map between two terms of type <i>Property</i>
<i>owl : inverseProperty</i>	This constructor is used to state that one property is the inverse of another. It is use to describe inverse relation between properties (i.e., exactly like the passive voice in the grammar)
<i>owl : inverseFunctionalProperty</i>	When the type ( <i>rdf:type</i> ) of a <i>property</i> term <i>X</i> is defined to be of <i>inverseFunctionalProperty</i> , Whenever <i>X</i> <i>property</i> is used as a predicate in a triple, its <b>object</b> will have <b>one</b> and <b>only one</b> <b>subject</b> . Thus, each object should be able to uniquely identify a subject. This constructor is a sub class of <i>owl : objectProperty</i>
<i>owl : FunctionalProperty</i>	Same idea as the last constructor, but here, when <i>X</i> is defined to be of type <i>FunctionalProperty</i> , each <b>subject</b> , where <i>X</i> is a predicate, can have at most one <b>object</b> . This constructor is a subclass of <i>rdf : property</i>
If X is a term of type ( rdf : type ) Class ( rdfs : Class )	
<i>owl : equivalentClass</i>	This constructor is used to map between two terms of type <i>Class</i>
Common Constructors between Property and Class terms	
<i>owl : sameas</i>	Two URI terms can be mapped together using the <i>sameas</i> constructor. This constructor indicates that these two terms actually refer to the same thing. It can be used as well to map between two ontologies.

### IV. DESIGN AND IMPLEMENTATION: CASE STUDY ON ACQUISITION PHASE

#### A. Work Environment

The CF-CoC framework is implemented using Php and easyRDF, Graphviz tool, and its graph objects are used within the easyRDF to produce and draw different RDF models<sup>3</sup>.

#### B. CF-CoC Terms Definitions

As shown in Figure 3, the first step is to create the ontology corresponding to a forensic phase. This ontology will contain all the forensic terms describing the different tasks of acquisition phase. Next figure shows the CF-CoC web application.

<sup>3</sup> <http://www.easyrdf.org/>

<http://www.graphviz.org/Documentation.php>

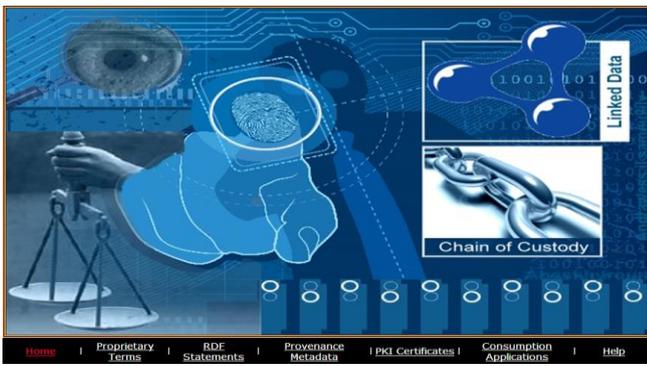


Figure 4 CF-CoC Web Application

As shown in Figure 4, the CF-CoC web application contains several modules. The first module, “Proprietary Terms”, contains two main tasks: the creation of ontology and the creation of terms (Figure 5).

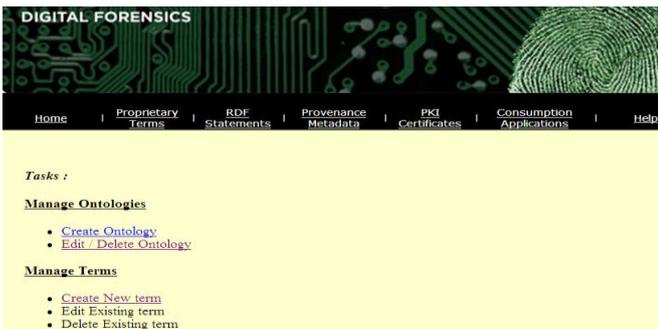


Figure 5 Tasks of Proprietary Terms Module

### 1) Creation of Ontology (Vocabulary):

This task is about to create the ontology object of the acquisition phase (see Figure 3). The domain name field is required to mint the ontology to a unique domain name owned by the publisher (aspect 2). The second field is about the selection of role player certificate [20]. In addition, the value type of the role player can be a resource or a literal. Next fields are the ontology name and its label description. Last field is the publication date of the acquisition ontology (see Figure 6).

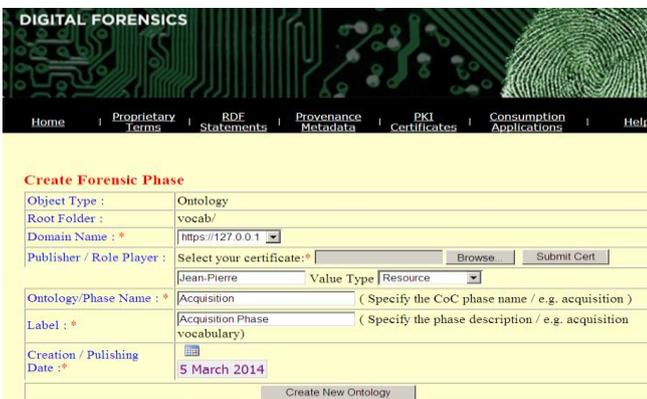


Figure 6 Creation of Acquisition Ontology

After completing this form, the acquisition ontology is generated by using the *Graphviz* module [24] (see Figure 7).

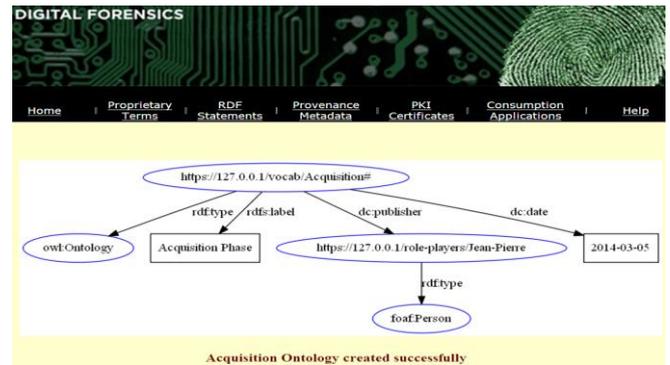


Figure 7 Creation of Acquisition Ontology

After creating the acquisition ontology, the role player proceeds with the next module to create terms and append them to this new ontology object.

### 2) Creation of terms (Acquisition Terms):

This task relates to four essential fields. The first field is the term name. The second field is selecting ontology to append the new proprietary term. The third field specifies the category/forensic task (see Figure 3). In our case, the category could be one of the three tasks provided in section 2 (preservation, recovery, or copy). In this field, the user may select ‘New’ to create a new category or select ‘Existing’ to import an existing category, defined in another vocabulary (ontology) created by another role player (see Figure 8) (i.e., two different forensic phase may have a common category/task [17]). Last field is the selection of term type (i.e., a term can be a property or a class).

Let now consider the following tangible CoC:

*“The name of the first responder in the acquisition phase is Jean-Pierre. He is the role player of this phase, and he preserved the state of the digital media, PDA device, which has the SN: 0G-4023-32-362. The date he did this task is 5 March 2014”*

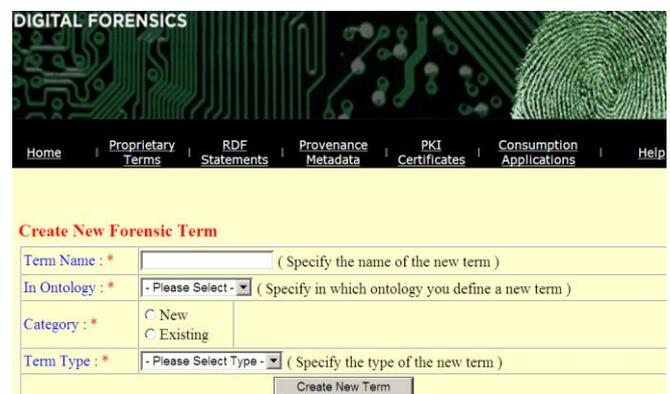


Figure 8 Creation of a New Term

The first step to create an *e-CoC* from this tangible *CoC* is to identify the terms (see Table 1). This case study contains T-Box and A-Box information. Terms of T-Box are of type class and property. The *Role\_player* term is a class that can be defined as a subclass from the class *Person* in the *FOAF* (friend of a friend) ontology<sup>4</sup> (see Figure 9). This term will belong to a forensic task called *Preservation*.

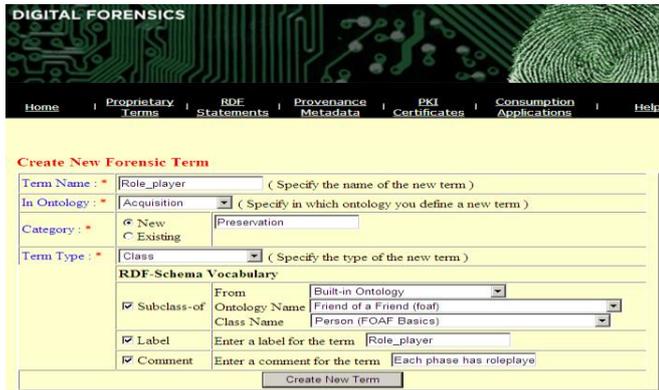


Figure 9 Creation of the Role\_player Class

The *First\_responder* term is a class that can be an instance of the *Role\_player* class. Now, the *Preservation* category will be found under the 'Existing' category. Finally, the *Digital\_media* is a subclass of *owl:Thing* (see Figure 10).

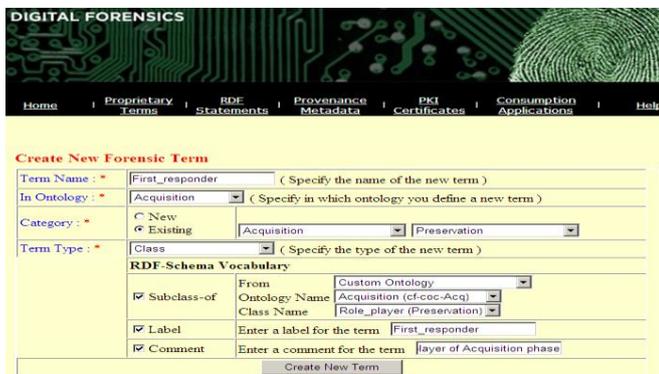


Figure 10 Creation of the First\_responder Class

Now, the property terms (*owl:objectProperty*) will be defined. The *domain* and *range* of the term *preservedby* are defined to be *Digital\_media* and *First\_responder* class, respectively. This property term is defined to be a subproperty from *foaf:made* property (see Figure 11).

The *preserve* property is the inverse of *preservedby* property. Thus, the *domain* and *range* of the former will be also the inverse, *First\_responder* and *Digital\_media* respectively. Simply, if a digital media is preserved by a first responder, then this means that the first responder preserved the digital media (see Figure 12). The last property is *SN*: the serial number of a device is an inverse functional property,

<sup>4</sup> <http://www.w3.org/TR/owl-ref>  
<http://xmlns.com/foaf/spec/>

because each serial number identifies one and only one subject (see Table 3).

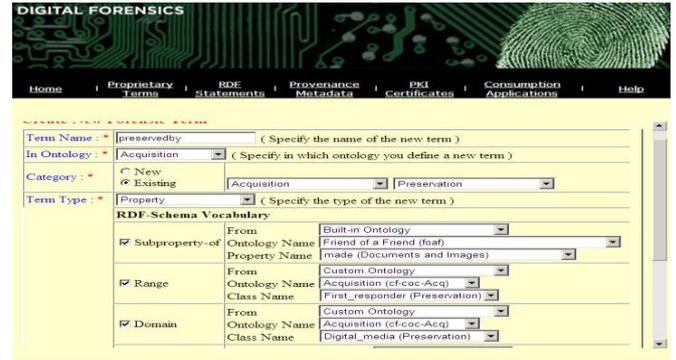


Figure 11 Creation of the preservedby Property

After creating all terms, the role player can generate the acquisition ontology with all the property and class terms of the preservation forensic task (Figure 12).

The created T-Box terms will be used to publish data. Therefore, they will describe the A-Box data. The latter is the *e-CoC* that will be consumed later by jury in court of law using different consumption application (browsing [22] [21], searching [6] or querying [2]). Now, the data can be described and published using the terms defined in the T-Box and by using the third layer of the framework, the user can publish different triples (i.e., using different vocabularies of the semantic web) and by the support of proprietary vocabularies defined by the role player.

Figure 13 shows the *e-CoC* (A-Box) of the forensic preservation task. This generated ontology does not answer all the question of *CoC*. It answers only the Who: Jean-Pierre, What: PDA device, and When: publication date of ontology. In order to have the answers to other questions, more terms need to be determined and defined. In this figure, the *cf-coc-Acq* is the prefix namespace of the acquisition ontology: *Jean-Pierre* is an instance from the *First\_responder* class (i.e., which is an instance of the *Role\_player* class), *PDA device* is an instance of *Digital\_media* (i.e., which is instance from *Things* class), and *presevedby* is the inverse property of *preserve* property. *SN* is a functional property where its domain is the *PDA device* and its range is the *OG4023-32-362* (i.e., which is an instance of the *Literal* class).

## V. CONCLUSION AND FUTURE WORK

This paper discusses the idea of creating proprietary terms using lightweight ontology in order to publish and describe forensic information on the web of data. The work illustrates this idea through the preservation task of the acquisition forensic phase. It depicts how the first two layers of the *CF-CoC* framework are designed and implemented. Future work will discuss the remaining layers of this framework and how an *e-CoC* will be consumed by juries in a court of law.

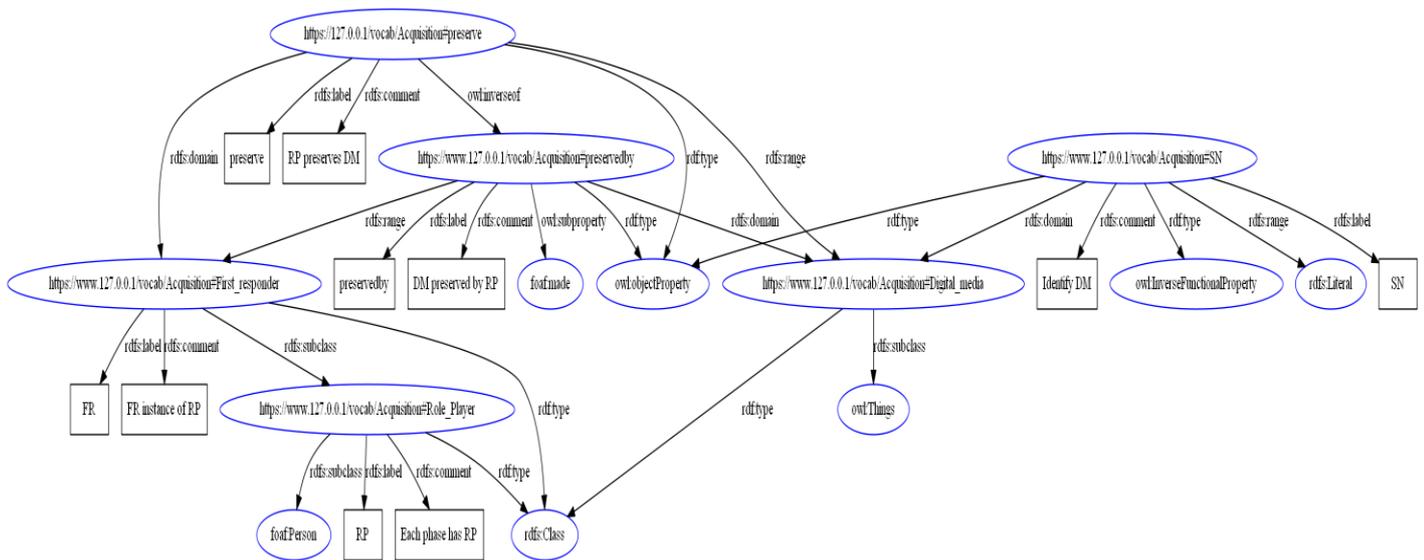


Figure 12 T-Box Ontology of Forensic Preservation Task <sup>5</sup>

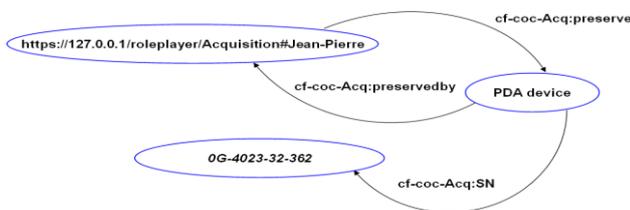


Figure 13 A-Box Ontology of Forensic Preservation Task (e-CoC)

## REFERENCES

- [1] T. Berners-Lee et al., "The World-Wide Web," Communications of the ACM, 2009, 37(8):76-82.
- [2] O. Hartig, C. Bizer, and J. Christoph Freytag, "Executing sparql queries over the web of linked data," In Proceedings of the International Semantic Web Conference, pp. 293-309, 2009
- [3] I. Jacobs and N. Walsh, "Architecture of the World Wide Web," W3C Recommendation., 2004, Volume One, <http://www.w3.org/TR/webarch/> [Retrieved June 14, 2009].
- [4] T. Berners-Lee, et al. (2005), "Uniform Resource Identifier (URI): Generic Syntax. Request for Comments: 3986," <http://tools.ietf.org/html/rfc3986> [Retrieved June 14, 2009].
- [5] R. Fielding, "Hypertext Transfer Protocol"-HTTP/1.1.Request for Comments:<http://www.w3.org/Protocols/rfc2616/rfc2616.html> [Retrieved June 14, 2009].
- [6] G. Cheng and Y. Qu, "Searching linked objects with falcons: Approach, implementation and evaluation," International Journal on Semantic Web and Information Systems (IJSWIS), pp. 49-70, 2009
- [7] T. F. Gayed, H. Lounis, and M. Bari, "Representing Chains of Custody along a Forensic Process: A Case Study on Kruse Model," The 25<sup>th</sup> International Conference on Software Engineering and Knowledge Engineering (SEKE 2013), Boston, USA.
- [8] G. Klyne and J. J. Carroll, "Resource Description Framework (RDF): Concepts and Abstract Syntax," - W3C Recommendation, <http://www.w3.org/TR/rdf-concepts/> [Retrieved 2004].
- [9] D. Brickley and R. V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema-W3C Recommendation. <http://www.w3.org/TR/rdf-schema/>, [Retrieved 2004].
- [10] D. L. McGuinness and F. V. Harmelen, "OWL Web Ontology Language Overview," - W3C Recommendation. <http://www.w3.org/TR/2004/REC-owl-features-20040210/>, [Retrieved 2004].
- [11] D. Berrueta and J. Phipps. Best practice recipes for publishing rdf vocabularies - w3c note. <http://www.w3.org/TR/swbp-vocab-pub/>, 2008.
- [12] N. Mendelsohn, The self-describing web-tag finding. <http://www.w3.org/2001/tag/doc/selfDescribingDocuments.html>, 2009.
- [13] A. Brinson, A. Robinson, and M. Rogers, "A Cyber Forensics Ontology: Creating a New Approach to Studying Cyber Forensics," "The International Journal of Digital Forensics & Incident Response. Vol 3, pp. 37-43, Sep 2006.
- [14] Technical Working Group for Electronic Crime Scene Investigation, "A Guide for first responders, United States Department of Justice," 2001.
- [15] E. Casey, "Digital Evidence and Computer Crime - Forensic Science," Computers and the Internet, 3rd Edition. Academic Press 2011, pp. 1-807, ISBN: 978-0-12-374268-1,
- [16] S.O.Ciardhuain,"An extended model of CC investigations," International Journal of digital Evidence, vol. 3, 2004.
- [17] Y. Yusoff, R. Ismail, and Z. Hassan, "Common Phases of Computer Forensics Investigation Models," IJCSIT, vol. 3, No 3, June 2011, pp. 17- 31.
- [18] M. Köhn, J. Eloff, and M. Olivier, "UML DFPMS," in Proceedings of Innovative Minds Conference, Johannesburg, South Africa, July 2008.
- [19] Technical Working Group for Electronic Crime Scene Investigation, Electronic Crime Scene Investigation: A Guide for first responders, United States Department of Justice, 2001.
- [20] T. F. Gayed, H. Lounis, and M. Bari, "Linked Closed Data Using PKI: A Case Study on Publishing and Consuming data in a Forensic Process," 6<sup>th</sup> International Conference on Advanced Cognitive Technologies and Applications, IARIA 2014, Venice, Italy [Status : Accepted].
- [21] T. Berners-Lee et al., "Tabulator: Exploring and analyzing linked data on the semantic web," In Proceedings of the 3rd International Semantic Web User Interaction Workshop, 2006.
- [22] Heath, T. How will we interact with the web of data? IEEE Internet Computing, pp. 88-91, 2008.
- [23] G. Antoniou and F. V. Harmelen, "Web Ontology Language: OWL," 2005, pp. 1-21.

<sup>5</sup> Labels and comments are reduced to increase the image clearness; RP: Role\_player, FR: First\_responder, and DM: Digital\_media

# DKDs: An Ontology-based System for Distributed Teams

Rodrigo G. C. Rocha, Ryan  
Azevedo, Dimas Cassimiro, Ana  
Raquel Morais  
Federal Rural University of  
Pernambuco - UFRPE  
Garanhuns, Brazil  
rodrigo, ryan, dimas@uag.ufrpe.br,  
anaraqueldemorais@gmail.com

Marcos P. Duarte  
Information Systems Course  
Paraíso College of Ceará  
Juazeiro do Norte, CE, Brazil  
marcos.duarte@fapce.edu.br

Silvio Meira  
Center of Informatics,  
Federal University of  
Pernambuco (CIn-UFPE)  
Recife, PE, Brazil  
srlm@cin.ufpe.br

**Abstract**— Global Software Development has become an option for software companies to expand their horizons and work with geographically dispersed teams, exploiting the advantages brought by this approach. However, this way of developing software enables new challenges to arise, such as the inexistence of a formal, normalized model of a project's data and artifacts accessible to all the individuals involved, which makes it harder for them to communicate, understand each other and what is specified on the project's artifacts. Then, this paper proposes a knowledge management tool that utilizes a domain-specific ontology for distributed development environments, aiming to help distributed teams overcome the challenges brought by this modality of software development proposing techniques and best practices. Thus, the main output of this work is **Ontology-based System to Support the software development process with distributed teams**.

**Keywords**-Global Software Development; Ontologies; Knowledge.

## I. INTRODUCTION

Motivated by opportunities like the availability of experts worldwide, cost reduction, local government incentives and employee turnover reduction, several software development companies have been starting to work with geographically distributed development teams, adopting the Distributed Software Development approach.

The aforementioned distribution of teams brings along with it new challenges to the software development scenario. Carmel [1] and Komi-Sirvo and Tihinen [2] reiterate the existence of these challenges by presenting some factors that are likely to lead distributed software development projects into failure: inefficient communication between distributed team members, diverging cultures and high complexity or lack of project management.

In this context, the nonexistence of a formal, normalized project data model accessible by the entirety of the team makes the communication between them and the understanding of the project artifacts harder, which can be aggravated when each member's culture and customs is barely or even not known by the rest of the team.

In order to mitigate these problems, the utilization of ontologies can be useful because they enable the creation of a common vocabulary. Wongthongtham et al. [3] mention that the use of ontologies represent a paradigm shift in Software Engineering and can be used especially to provide semantics for support tools, strong, knowledge-based communication, centralization and information availability.

This paper proposes DKDOnto, a domain-specific ontology for distributed software development projects, whose purpose is to aid those projects by defining a common vocabulary for distributed teams. Besides, this work proposes a tool that enables both handling and searching the information in the knowledge base, in order to get more useful information as to mitigate and avoid future problems inside the project.

The main goal of this work is the proposal of both the ontology and the tool, which together will compose a mechanism to ease the distributed software development process, from sharing of common knowledge between distributed team members or smart agents to the decision-making process effectuated by the project managers.

This paper is organized as follows: Ontology concepts are presented in Section II; Section III contains the knowledge-based system proposal; Related works are presented in Section IV, where a succinct analysis and comparison of related work and this paper is made; and, finally, Section V brings the final considerations.

## II. ONTOLOGIES

Various definitions are given as to determine a meaning to ontologies in the Computer Science context, the most popular and best-known definition being “a formal, explicit specification of a shared conceptualization”, given by Gruber [4]. By ‘formal’, he means that it is declaratively defined so that it can be comprehended by smart agents; by ‘explicit’, he means that the elements and their restrictions are clearly defined; by ‘conceptualization’, he means an abstract model of a field of knowledge or a limited universe of discourse; by ‘shared’, he indicates it is consensual knowledge, a common terminology of the modeled field. Thus, ontologies set an unambiguous, common higher abstraction level for several knowledge domains.

Ontologies, according to Guizzardi [5], are composed by concept, relations, function, axioms and instances. In short, concept can be ‘anything’ about ‘something’ that is going to be explained. The interaction between a domain’s concepts and attributes is called relation, whose type is called function. Axioms model sentences that are always true and instances represent elements from the domain associated with specific concepts.

The use of ontologies has been made popular by many other Computer Science subfields, such as: Software Engineering, Artificial Intelligence, Database Design, and Information Systems. One of the principal persons responsible behind this phenomenon is Web Semantics’ creator [6], Sir. Tim Berners Lee. Many reasons instigate the development of ontologies, according to [7] [8]. Some of these reasons are:

- Sharing common understanding of how information is structured between humans and smart agents;
- Reusing knowledge of a domain. In case there is an ontology that adequately models certain knowledge of a domain, it can be shared and used by engineers and ontology developers, as well as teams that develop semantic and cognitive applications;
- Making explicit assumptions of a domain. Ontologies provide vocabulary to represent knowledge and its use prevents misinterpretations;
- Possibility of translation from and to various languages and knowledge representation formalisms. The translation concretizes an ideal pursued for generations by researchers in Artificial Intelligence. It makes it easier to reuse knowledge, and may allow for communication between agents in different formalisms, since this service is available in an increasing number of knowledge representation formalisms.

Furthermore, ontologies help solve some of DSD project problems; for example, how to establish better communication, allow a homogenous comprehension of project information, make the project management a less laborious task, prevent task interpretation errors and synchronize the enrolled, distributed team’s efforts and facilitate the knowledge sharing and standardization.

### III. KNOWLEDGE-BASED SYSTEM PROPOSAL

In this work, is presented the DKDOnto, a domain-ontology according to classification adopted by [9], which classifies the types of ontologies in: i) generic, ii) domain, iii) task and iv) application.

The ontology proposed intends to be the basis for possible solutions of knowledge-based systems in the context of global software development, in order to assist all the professionals (client too) involved in the software development process with distributed teams. The DKDOnto emerges, thus, as a common knowledge base for this context, leveraging the challenges deals, best practices and possible solutions, as well a road map with all the actors and their assignments.

This proposal takes a step beyond, discussing also an inference engine called DKDs, extremely flexible, customizable for each environment and giving support for the professional in real time. The general flow, operating means and features of the proposed system and the DKDOnto.

#### A. DKDOnto: Proposal Ontology

The DKDOnto ontology was developed using Ontology Engineering, Methontology [14] and IEEE Standard [15] for developing knowledge-based information systems methodologies; also, Method 101, proposed by N. F. Noy and D. L. Mcguinness’s [7] was used a complement to Methontology.

Thus, the language used to build the ontology was OWL, which eases the publication and sharing of ontologies [16] and it has also been proposed as a standard for the World Wide Web Consortium (W3C), incorporating and taking advantage of the strength of earlier languages. OWL is an ontology language (Semantic Web [17]) with high-level expressivity and great potential for knowledge inference. In order to edit the ontology, the use of Protégé [18] was employed. It is a free, extensible, Java-based, open-source ontology editor and knowledge-based framework.

The DKDOnto has about 50 classes, but this paper describes the following core classes.

- **Project:** the main class of this knowledge base. It is responsible to store all the information about the settings of projects, from allocated team members to phases to activities to artifacts used.

- **Member:** it is a subclass of Resource. Member is an individual who has access to the environment and are allocated to Projects. A member has skills and works in a place and participates directly in the project, reporting best practices and challenges, using and creating artifacts.

- **Best Practices:** all the solutions and best practices used to face any problem should be stored in this entity. This class is responsible for helping avoid challenges and problems found and reported by a member during the execution of their activities. It also to solve these challenges and problems.

- **Challenges:** all the challenges and problems found by members should be stored in this class. A challenge can use best practices to solve itself. This entity is fundamental because the challenges has some solution or best practice associated with some practice can be used and available to another members with same problems.

- **Skills:** all members’ knowledge are stored in this entity. The Member’s skill enables to avoid challenges and solve it too. This class allows too that activities be distributed for the members according their skills.

- **Place:** it is a fundamental class to define exactly where the envolved member are in Project. This entity estores all information about member’s localization, defining what is dispersion level and temporal distance.

- **Artifact:** class that is used by almost all other main classes. It supports members and their activities. Tools can use artifacts in specific activities, too.

This ontology uses two fundamental classes for the suces of this proposal. These classes are responsables for storage all information about the problems and solutions during the project. These classes are called of Challenges and BestPractices. Thus, user’s queries allows to view responses of the challenges, the knowledge base returns the best practices found for a certain team setting and can be applied to support

challenges, which can be useful for other teams involved with the same project or other teams from different projects.

Another important concept about ontologies is the Axiom, Freitas [8] affirms that Axioms are an important component of ontologies to describe the relationships among the concepts. The DKDOnto used a set of axioms to ensure the rules were met by knowledge domain. For a better contextualization, some axiom are presents below, in Figure 1.

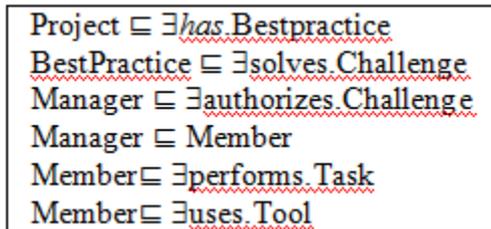


Fig1. Some Axioms from DKDOnto

All the main concepts used in Axioms are referents the fundamental classes from DKDOnto. The first means that a Project has a Bestpractice, so, the second means that Bestpractice can solve a Challenge, then the manager can authorize Challenges related by another members, Manager is a Member, Member executa the activities or tasks, and finally, Member user Tools. It was verified the creation of an existential quantifier in generated axioms indicating that Description Logic was used.

*B. DKDs: Proposal Tool*

DKDs was developed to aid in the transmission, generation and distribution of knowledge. It is a support tool for decision-making in DSD, which, based in resources and information from the context of a project, the system suggests possible solutions for the problems found to its users. In this sense, the system accesses the knowledge base having distributed projects experiences, their configurations, challenges faced and solutions used to overcome those challenges.

This tool’s main goal is to support the complete DSD process, offering recommendations considering the project setting and organization, technical and nontechnical experiences.

In order to develop DKDs, the general platform adopted was J2EE [19]; the web application frameworks utilized were Grails [20] (High-productivity web framework based on the Groovy language [21]) and Google Web Toolkit (GWT) [22]; Hibernate (Java persistence framework project) [23] was used for persistence; and to manipulate the ontology, the Jena framework was employed, which is also responsible for construction and manipulation of Resource Description Framework (RDF) [24] graphics.

With the DKDs a member from a project can know who are the another members involved and have some instructions to talk each other depending their cultural characteristics. So, it helps to avoid any problems the communication (email, talk, phone). Furthermore, any doubt about some artifact or activity can be solved with the correct member, that is indicated by the tool.

Among DKDs’ main features, the most important ones are: DKDs uses the inference engine Pellet for inferring facts based

on the information that has been previously stored in the knowledge base, thus, some outcomes that the system can generate:

- Starting the project, request a guideline with suggested best practices for similar contexts
- Starting the project, request a guideline with main challenges for similar contexts
- Determines who are the most qualified members to solve technical problems;
- Suggests possible practices, tools or techniques that can be employed to avoid challenges
- Find possible solutions used previously to problems encountered
- Evaluating the solutions proposed by the tool
- Suggest adaptations to the proposed solutions

The application is basically composed by four modules, the Inference Module: allows for a precise deduction of information about DKDOnto in RDF and OWL code, using inference engine Pellet. The Query Module, this is where all the queries made by users occur. As it was mentioned earlier, queries are made in SPARQL language and are transparent to the users. The Views Module: gives access to all the reports made according to the users’ needs. And finally, the Management Module: responsible for enabling access to the ontology with insertion, removal and editing of the data in the ontology permissions.

For example, an user can access the application and insert, delete, edit and view all the data (instances contained in DKDOnto) by the Management Module. The same user can use View Module for the ask the system to inform what is necessary, so, this module activates the Query Module that use the Inference Module to bring appropriate responses for the user.

The users have an access interface to execute the abovementioned functions on one side, whereas on the other side, there is the SPARQL (Query language for Resource Description Framework) [25] inference engine to consult DKDOnto, and the interface component (OWL API [26]) in the middle, which interacts with both sides. Integrating all the demands from user using the inference module.

Figure 2 shows the tool’s general functioning as described above.

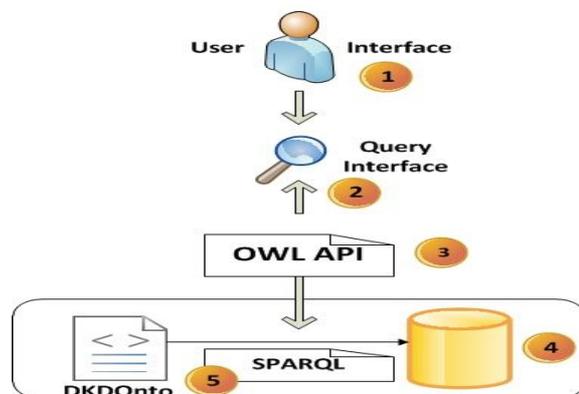


Fig. 2. Tools General Functioning

#### IV. CONCLUSION

As globalization took place, the distribution of software development processes have become an increasingly common fact. The DSD work environments are very complex and there are no mature practices for this context since it is relatively new. In this sense, ontologies can bring benefits such as a shared understanding of information, ease of communication among distributed teams and effectiveness in information management.

DKDOnto and DKDs fulfill what has been proposed, consisting of a computing tool that can be used for treatment, analysis and utilization of information on distributed software projects. In this sense, the ontology and the tool allow that actors in this scenario obtain and access correct information and artifacts, providing a high-level knowledge model for the team members. This work was realized after a systematic mapping that aimed to identify ontologies formalized in DSD context, provided that advance the state of art, highlighting the need to use ontology in this field. Is possible to view all the Systematic Mapping Results in Borge's work [27].

The results obtained to this date are expressive, in which, for example, the project manager has actual consistent knowledge of which cultures are involved in the distributed teams and which are the implication of this, which enables them to handle each case effectively. Similarly, a technical leader has access to the project participants' technical knowledge, making them able to require or assign specific activities accordingly to the expertise of each team member.

Another important point is that the ontology, as presented in Section 3, has two fundamental classes, namely Challenges and Solutions that are directly utilized by the query tool. That way, the knowledge base will return the challenges found for a certain team setting and also which solutions can be applied to such challenges, which can be useful for other teams involved with the same project or other teams from different projects.

The next step in this segment is to concretize the acquisition of knowledge in a systematic way in order to fill the ontology. In this case, it will be possible to make tests and simulations with higher precision since all the inserted data will be from real projects. Furthermore, other techniques can be used for improves the support of Challenges, for example, the use of natural procesing language for retrieve better solutions or best practices based in challenges cases.

#### REFERENCES

- [1] E. Carmel. "Global Software Teams: Collaboration Across Borders and Time Zones". Prentice-Hall, EUA. 1999.
- [2] S. Komi-Sirvo and M. Tihinen. "Lessons Learned by Participants of Distributed Software Development". *Journal Knowledge and Process Management*, vol. 12 no 2, 2005, pp. 108–122.
- [3] P. Wongthongtham, E. Chang, T. Dillon, and I. Sommerville. "Ontology-based Multi-site Software Development Methodology and Tools". *J. of Systems Architecture*. ACM, New York. 2006. 640–653.
- [4] T. Gruber. "Toward Principles for the Design of Ontologies used for Knowledge Sharing". In *formal Ontology in Conceptual Analysis and Knowledge Representation*. Kluwer Academic Publishers. 1995.
- [5] G. Guizzardi. "A methodological approach to development and reuse, based on formal domain ontologies" Master Degree. Federal University of Espirito Santo. 2000.
- [6] T. Berners-Lee, O. Lassila, and J. Hendler. "The semantic web." *Scientific American*, 2001, pp. 5:34–5:43.

- [7] N. Noy and D. McGuinness. "Ontology development 101: A guide to creating your first ontology." [Online]. Available: <http://www.ksl.stanford.edu/people/dlm/papers/ontology101/ontology101-noy-mcguinness.html>. [retrieved: 06, 2013]. 2001.
- [8] F. Freitas. "Ontologies and the semantic web". *Proceedings of XXIII Computer Science Brazilian Society Symposium*. Campinas: SBC. v. 8, 2003, pp. 1-52.
- [9] N. Guarino, "Formal ontology and information systems," in *Proceedings of FOIS98*. Trento, Italia: IOS Press, pp. 3–15. 1998.
- [10] I. Mirbel. "OFLOSSC, "An Ontology for Supporting Open Source Development Communities". In *Proceedings of the International Conference on Enterprise Information Systems (ICEIS)*. 2009.
- [11] W. Maalej and H. Happel. "A Lightweight Approach for Knowledge Sharing in Distributed Software Teams". In *Proceedings of the Practical Aspects of Knowledge Management (PAKM)*. 2008.
- [12] T. Dillon and G. Simmons. "Semantic Web support for Open-source Software Development". In *Proceedings of the International Conference on Signal Image Technology and Internet Based Systems (SITIS)*. 2008.
- [13] A. Chaves, I. Steinmacher, C. Lapasini, E. Huzita, and A. Biasão. "OntoDISEN: an Ontology to Support Global Software Development". *CLEI Electronic Journal*. 2011. v. 14, pp. 1-12.
- [14] M. Fernandez, A. Gomez-Perez, and N. Juristo, "Methontology: from ontological art towards ontological engineering," in *Proceedings of the AAAI97 Spring Symposium Series on Ontological Engineering*, Stanford, USA, 1997, pp.33–40.
- [15] IEEE, "Standard for developing software life cycle processes". p. 96, may 1997, eEE Computing Society. Available: <http://standards.ieee.org/catalog/olis/archse.html>. [retrieved:06, 2013]. 1997.
- [16] OWL. Web ontology language overview. Available: <http://www.w3.org/TR/owl-features>. [retrieved: 06, 2013]. 2009.
- [17] J. Berners-Lee and O. Lassila, "The semantic web," *Scientific American Magazine*. [retrieved: 06, 2013]. 2001.
- [18] Protégé. Protégé ontology editor. Online. [Online]. Available: <http://protege.stanford.edu/doc/users.html>. [retrieved: 06, 2013]. 2009.
- [19] J2EE. JAVA Enterprise Edition. Available: <http://oracle.com/technetwork/java/javaee/overview/index.html>, [retrieved: 06, 2013]. 2013.
- [20] Grails. Available: <http://grails.org>, [retrieved: 06, 2013]. 2013.
- [21] Groovy. Available: <http://groovy.codehaus.org>, [retrieved: 06, 2013]. 2013.
- [22] Google Web Toolkit. Available: <http://gwtproject.org>, [retrieved: 06, 2013]. 2013.
- [23] Hibernate. Available: <http://hibernate.org>, [retrieved: 06, 2013]. 2013.
- [24] J. Carroll, D. Reynolds, I. Dickinson, A. Seaborne, C. Dollin, and K. Wilkinson, "Jena: Implementing the semantic web recommendations". pp. 74–83. 2004.
- [25] SPARQL Query Language for RDF. Available: <http://w3.org/TR/rdf-sparql-query>, [retrieved: 06, 2013]. 2013.
- [26] OWL API. Available: <http://owlapi.sourceforge.net>, [retrieved: 06, 2013]. 2013.
- [27] A. Borges, R. Rocha, C. Costa, H. Tomaz, S. Soares, and S. Meira. "Ontologies Supporting the Distributed Software Development: a Systematic Mapping Study". In *Proceedings of the International Conference on Evaluation & Assessment in Software Engineering (EASE)*. Porto de Galinhas, PE, Brasil. 2013.

# Industrial Analytics to Discover Knowledge from Instrumented Networked Machines

Aldo Dagnino

Software Research Area  
ABB Corporate research  
Raleigh, NC. USA  
aldo.dagnino@us.abb.com

David Cox

Software Research Area  
ABB Corporate research  
Raleigh, NC. USA  
David.cox@us.abb.com

**Abstract**— Computer technologies, sensor technologies, fiber optics, high-performance network technologies, mobile devices, wireless sensor networks, and powerful data repositories have permeated all layers of industry and opened the door for unprecedented data analytic capabilities used to improve equipment operation, equipment design, process quality, and product quality. Powerful and inexpensive sensors can continuously collect data about industrial equipment operations and industrial processes. Human operators and maintenance workers also utilize mobile devices to carry out their work collecting and transmitting enormous amounts of data related to the industrial activity. This paper presents the concept of Industrial Analytics which provides a framework for analyzing industrial data to discover valuable knowledge of the industrial activity that can help make decisions related to equipment design, predictive maintenance, process optimization, remote services, and product quality. The paper also presents an Industrial Analytics example applied to instrumented and networked electro-mechanical devices conducting their work in industrial environments.

*Keywords*— industrial analytics; sensors; machine learning; data repositories; networked electro-mechanical devices; knowledge discovery.

## I. INTRODUCTION

We are in a new era of industrial growth that combines computers, sensors, data repositories, high bandwidth networks, mobile devices, autonomous machines, and data analytics that drive industrial innovation and growth. [6]. Exemplified variously by the “Industrial Internet,” “Internet of Everything,” “Sensor Analytic Ecosystems,” and the “Internet of Things,” this era merges fields such as machine learning, machine-to-machine and human-to-machine communications, and sensor and database technologies. [4].

Examples include Brizzi et al. [1] where real-time data collection and analysis on manufacturing processes allow companies to optimize energy and water consumption at each stage of the production cycle. Liu et al. [5] also discuss how the IoT can be applied to the automotive industry for monitoring, managing, and processing automobile manufacturing and the post-sale social management process. Authors like daCosta [3] discuss how the IoT will enable machine-to-machine communication.

We find in the literature very little discussion about the automated analysis of industrial data. We refer to this capability as Industrial Analytics. The hallmarks of Industrial Analytics is the automated collection of large volumes of data generated by machines, computers, and people and the application of statistical and machine learning techniques to draw conclusions and knowledge about industrial processes.

In this paper we present one example of Industrial Analytics where data generated by electromechanical equipment is analyzed for the purpose of industrial system design and predictive maintenance. Section II of this paper discusses the importance of instrumented and networked machines in Industrial Analytics. Section III presents a proposed framework for Industrial Analytics. Section IV presents results and analyses conducted in a real-world case at ABB. Finally, Section V presents conclusions and future work in the field of Industrial Analytics.

## II. INSTRUMENTED AND NETWORKED MACHINES

Instrumented machines have integrated sensors that collect data related to machine operations, interactions with other machines, machine-process data, and product quality data [1]. Examples of these machines include: (a) power transformers that operate in power transmission and distribution grids and have sensors that measure attributes such as concentration of gasses and moisture levels in the coolant oil, vibration, oil temperature, power loading, and ambient temperature and moisture; (b) robots that operate in manufacturing, packaging, and assembly, with sensors that measure controller temperatures, torques, arm loads, electrical values, CPU utilization, ambient temperatures, ambient acidity, and ambient moisture; (c) motors and drives; (d) instrumented mining equipment such as gearless mill drives; (e) cargo ships; and (f) instrumented machine tools, among others. Instrumented machines connect to high-performance networks (HPNs) to transmit high data volumes in real-time. HPNs quickly transmit large volumes of data, allow the sharing of expensive computational resources, and allow distributed analytics with computing and data resources located at widely separated locations. In theory, this allows users to bring more computing power and better data to bear on their problems, but in practice the difficulties

imposed by distributed computing have often outweighed the benefits, resulting in limited acceptance of distributed computing by the user community. Instrumented machines are becoming more of a norm than exception in industry and as the time goes by, this trend will continue to grow.

### III. INDUSTRIAL ANALYTICS FRAMEWORK

Figure 1 shows a high level diagram of the context employed at ABB for Industrial Analytics. Instrumented networked machines produce work for industrial processes. These machines and processes are instrumented with sensors that collect data store it in a variety of repositories. A data analytics platform pre-processes and cleanses the data. The platform also is utilized to analyze the performance of both machines and process. These activities result in the generation of knowledge that can be used for predictive maintenance, diagnostics or for improving product design.

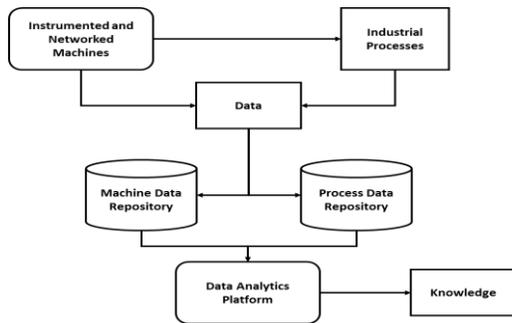


Figure 1. Industrial Analytics Framework

### IV. GENERATING DESIGN KNOWLEDGE FROM INSTRUMENTED ELECTROMECHANICAL MACHINES

Industrial machines perform heavy, repetitive, potentially dangerous, high volume work. These machines include motors and drives, mining equipment, power equipment, and flexible automation equipment, among others. Fig. 2 illustrates a simplified installation diagram used in the remainder of this paper to discuss the concept of Industrial Analytics. This installation consists of: (a) instrumented equipment cooling fans; (b) instrumented electro-mechanical machines that perform production work; (c) instrumented controller(s); (d) high-bandwidth fiber optics network; (e) fiber optics transmitters; and (f) array of data repositories that persist time series data gathered by the sensors. The sections below describe examples of how data generated by real-world implementations of Fig. 2 was analyzed to derive useful knowledge for product re-design, diagnostics and predictive maintenance.

### V. DATA DESCRIPTION AND ANALYSIS

Production data for installed machines at many customer sites are continuously collected and stored in a relational database. The database schema consists of more than 70 tables and at the time of the study contained less than 0.5 terabyte of data. The machines are of the same equipment family and the data available consist of event log entries.

The date, time, type, and reason behind each event are recorded. Data include measurements on battery status, fan speeds, communications status, voltage levels, I/O Bus information, memory, CPU utilization, temperatures, positional and movement information, and controller-related data. Also recorded are alarms associated with the recorded events. Measurements and event log entries are associated with alarms. Alarm information includes status code, comments, acknowledgement date, and notifications times.

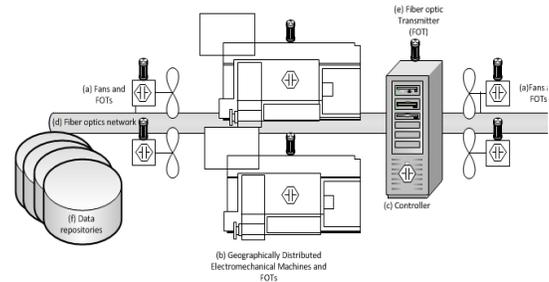


Figure 1. Single-site instrumented networked machines

Critical elements for which data analyses are conducted in the system presented in Fig. 2 utilizing sensor data collected include the cooling fans, CPU temperature, CPU utilization, and motion analyses. Historical data based on these sensor data was utilized to develop machine learning models that are utilized to analyze incoming new data from these components. These analyses are summarized below.

#### 1) Cooling Fans Data Analysis

Several fans are used in installations similar to the one that in Fig. 2. These include CPU, cooling unit, the controller, and power fans. Speeds (RPMs) are collected, transmitted and stored for all fans. Fig. 4 shows the distribution of the speeds of a typical fan. The letters A through F indicate the presence of multiple peaks. Peaks A and D are very prominent suggesting two distinct major and four minor populations. Fan speeds for this installation vary considerably between 2,700 and 14,000 RPM. Analysis of the data generates knowledge about velocities related to fan failures, associations between failures and events, associations between velocities and temperatures, and associations between velocities and system alarms.

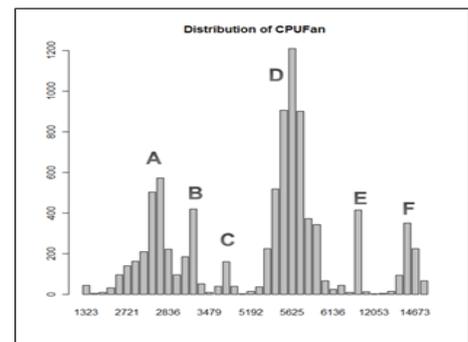


Figure 2. Distribution of fan speeds.

Fig. 4 illustrates a downward trend in fan speed over several weeks that precede a fan failure. In this case it is possible to observe that the speeds return to normal after the fan is replaced. Using time series prediction techniques it is possible to anticipate fan failures from this data often days or weeks in advance of the actual failure.

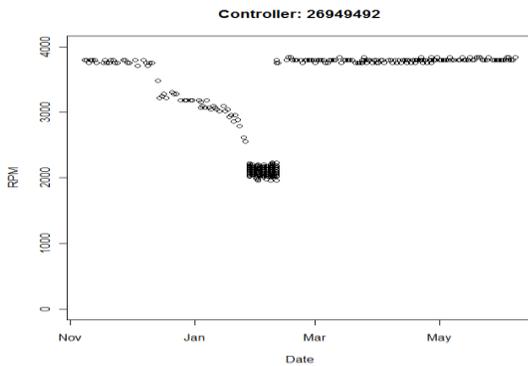


Figure 3. Fan speed degradation over time.

### 2) Fan Speed and CPU Temperatures

Referring to Fig. 2 fan speeds form several clusters. By assigning each speed to the nearest peak, we perform an analysis of variance to see if fan speeds are related to CPU temperature. Fig. 5 shows a resulting box plot and the P-value ( $<2.23 \cdot 10^{-16}$  in Fig. 6) indicates a very strong association.

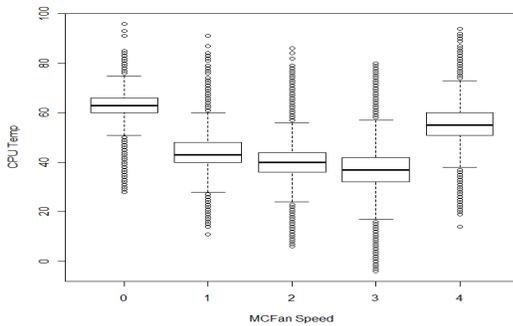


Figure 4. Fan speed boxplot.

Analysis of Variance					
Response: y					
	DF	Sum Sq	Mean Sq	F value	Pr(>F)
group	4	35858939	8964735	145351	$<2.2e-16$
Residuals 513437 31667076 62					

Figure 5. Fan speed analysis of variance.

The box plot shows temperatures decreasing as speeds increase. One might expect that fan speeds are a response to temperature and that higher temperatures associate with higher speeds. What we observe is that higher speeds are associated with the degree of cooling and, thus, a decrease in

temperatures. However, above a certain threshold temperatures increase indicating that the fan cannot provide additional cooling.

### 3) Fan Speeds and Events

A Decision Tree analysis models correlations between fan speeds and events as illustrated in Fig. 7. For example, the model identifies alarm 20310 (fan failure) with an accuracy of 92.99%, a precision of 63.53%, a recall of 99.53%, and an f-measure of 77.56%. The model also identifies alarm 37054 (faulty fan) with an accuracy of 92.99%, a precision of 95.14%, a recall of 91.51%, with a calculated f-measure of 93.29%. Lastly, the Decision Tree model identifies alarm 50204 (motion supervision module issue) with an accuracy of 92.99%, a precision of 98.75%, a recall of 92.54%, and an f-measure of 95.54%.

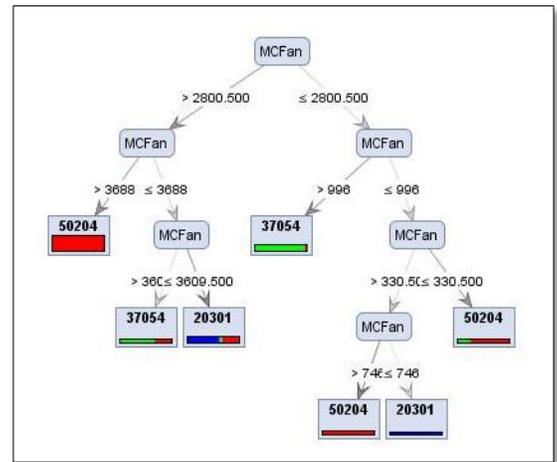


Figure 6. Decision tree correlating events to fan speeds.

### 4) Memory Usage and Temperature Analysis

The amount of RAM utilized by the main controller computer over time was also collected. Decision Tree analyses on alarms were conducted to identify trends between alarms and RAM usage (Fig. 8). The model was able to predict alarm 37050 (over temperature in main computer) with a precision of 94.44%, a recall of 61.08%, an f-measure of 74.1% and an accuracy of 95.61%. The model also predicts alarm 37054 (Faulty Computer Unit fan) with a precision of 95.69%, a recall of 99.59%, and a calculated f-measure of 97.60% and an accuracy of 95.61%.

### 5) Motion Analysis

Electromechanical devices move and to be useful they must move in predictable ways. However, mechanical devices drift over time causing errors in motion. We examined position, speed and torque and employed decision trees to predict the warnings and alarms from various combinations of values. In Figure 9 we examined position and speed to construct a decision tree to predict collisions with items in the environment.



# Testing Network Protocols: Formally, at Runtime and Online

Xiaoping Che, Stephane Maag, Jorge Lopez and Ana Cavalli

CNRS UMR 5157, Institut Mines-Telecom/Telecom SudParis, Evry, France

Xiaoping.Che,Stephane.Maag,Jorge.eleazar.Lopez\_coronado,Ana.Cavalli@telecom-sudparis.eu

**Abstract**—Testing a protocol at runtime in an online way is a complex and challenging work. It requires the ability to handle numerous messages in a short time, and also requires the same offline testing preciseness. Meanwhile, since online testing is a long term continuously process, the tester has to undergo severe conditions when dealing with large amount of nonstop traces. In this paper, we present a novel logic-based online passive testing approach to test, at runtime, the protocol conformance and performance through formally specified properties with new definitions of verdicts. In order to evaluate and assess our methodology, we experimented our approach with several Session Initiation Protocol properties in a real IP Multimedia Subsystem environment. Relevant verdicts and discussions are provided.

## I. INTRODUCTION

Testing is a crucial activity in the evaluation process of a system or an implementation under test (IUT). Among the commonly applied approaches, the *passive* testing techniques (also called *monitoring*) are today gaining efficiency and reliability [9]. These techniques are divided in two main groups: *online* and *offline* testing approaches. Offline testing computes test scenarios before their execution on the IUT and gives verdicts afterwards, while online testing provides continuously testing during the operation phase of the IUT.

With online testing approaches, the collection of traces is avoided and the traces are eventually not finite. Indeed, testing a protocol at runtime may be performed during a normal use of the system without disturbing the process. Several online testing techniques have been studied by the community in order to test systems or protocol implementations [14], [10], [2]. These methods provide interesting studies and have their own advantages, but they also have several drawbacks such as the presence of false negatives, space and time consumption, often related to a needed complete formal model, etc. Although they bring solutions, new results and perspectives to the protocol and system testers, they also raise new challenges and issues. The main ones are the non-collection of traces and their on-the-fly analysis. The traces are observed (through an interface and an eventual sniffer) and analyzed on-the-fly to provide test verdicts and no trace sets should be studied a posteriori to the testing process. In our work, we present a novel formal online passive testing approach applied at runtime to test the conformance and performance of the IUT.

We herein extend our previous proposed methodology [3], [4] that presented a passive testing approach for checking the requirements of communicating protocols. In [3] and [4], a formalism was applied to test in an offline way the conformance and performance of an IUT. In this new paper, we develop our approach to test these two aspects in an online way in considering the above mentioned inherent constraints and challenges. Furthermore, our framework is designed to test them at runtime, with new required verdicts definitions of ‘*Time-Fail*’, ‘*Data-Inc*’ and ‘*Inconclusive*’ representing unobserved

message within timeout, untested data portion and uncertain status respectively. Finally, to demonstrate the efficiency of our online approach, we apply it on a real communicating environment for assessing its preciseness and efficiency.

Our paper’s primary contributions are:

- We provide a formal online passive testing approach to avoid stopping the execution of the testing process when monitoring a tested protocol. The analyzed traces are never cut which improves the accuracy of the test verdicts.
- Our approach allows the testing process to be executed in a transparent way without overloading the CPU and memory of the used equipment on which the tester will be run.
- Data portion of the messages is taken into account in our online testing approach, and new definitions of online testing verdicts are introduced.

## II. RELATED WORKS

When studying the literature, we note that there are very few papers tackling online passive testing. We can however cite the following ones.

In [16], the authors proposed two online algorithms to detect 802.11 traffic from packet-header data collected passively at a monitoring point. They built a system for online wireless traffic detection using these algorithms. Besides, some researchers presented a tool for exploring online communication and analyzing clarification of requirements over the time in [8]. It supports managers and developers to identify risky requirements. We should also cite the works [11], [15] from which an industrial testing tool has been developed. These works are based on formal timed extended invariant to analyze runtime traces with deep packet inspection techniques. However, while most of the functional properties can be easily designed, complex ones with data causality can not. Though their approach is efficient with an important data flow, the process is still offline. To be complete, we have to mention that studies have also been performed to generate invariant from model-checkers. However, it requires a formal model and it still raises unresolved issues [6].

Inspired from these above cited works, we propose an online formal passive testing approach by defining functional properties of IUT, without modeling the complete system, and by considering eventual false negatives. For this latter, we introduce a new verdict ‘*Time-Fail*’ for distinguishing the real faults and the faults caused by timeouts. In addition, since online protocol testing is a long-term continuously testing process, we provide a temporary storage for remaining the integrity of incoming traces. Further, our approach provides the ability to test both the data portion and control portion, accompanying with another new verdict ‘*Data-Inc*’.

### III. ONLINE TESTING APPROACH

In this section, we describe the architecture and testing process of our online testing approach. We also provide the new definitions of online testing verdicts.

#### A. Architecture of the approach

In our approach, the Horn logic [7] is used for formally expressing properties as formulas. This logic has the benefit of allowing the re-usability of clauses. And it provides better expressibility and flexibility when analyzing protocols. A syntax tree generated from the formulas will be used for filtering incoming traces and optimizing evaluation processes. For the evaluation part, we use the SLD-resolution algorithm for evaluating formulas. The architecture of our online testing approach is illustrated in Figure 1.

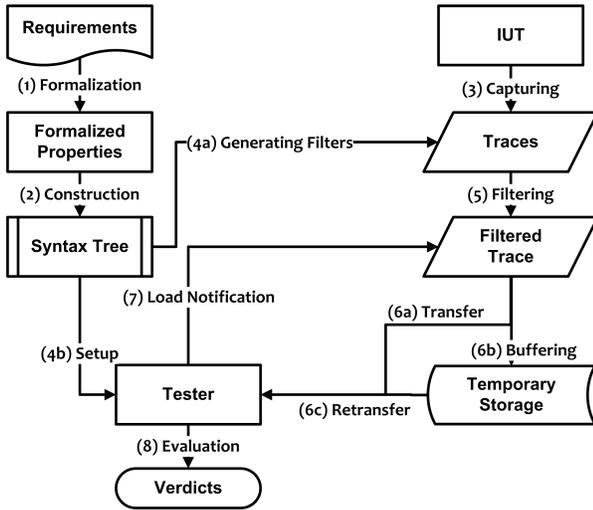


Fig. 1. Architecture of our online testing approach

#### B. Testing Process

The testing process consists of 8 parts (Figure 1): Formalization, Construction, Capturing, Generating Filters/Setup, Filtering, Transfer/Buffering, Load Notification and Evaluation.

**Formalization:** Initially, informal protocol requirements are formalized using Horn-logic based syntax. Due to the space limitation, we will not go into details. The interested readers may have a look at the works [3] and [9]. Then the verdicts  $\{‘Pass’, ‘Fail’, ‘Time-Fail’, ‘Inconclusive’, ‘Data-Inc’\}$  are provided to the interpretation of obtained formulas on real protocol execution traces.

**Construction:** From formalized formulas, a syntax tree is constructed for further testing processes. In this process, each formula representing a requirement will be transformed to an Abstract Syntax Tree (AST) using the TREEGEN algorithm [12]. The standard BNF representation of each formula is the input to construct an AST. All the generated ASTs are finally combined to a syntax tree using a fast merging algorithm [1]. The syntax tree will be transferred to the tester as requirements and will be used to filter the captured traces.

**Capturing:** The monitor consecutively captures traces of the protocol to be tested from points of observations (P.Os) of

the IUT, until the testing process finishes. When messages are captured, they are tagged with a time-stamp  $t_m$  in order to test the properties with time constraints and to provide verdicts on the performance requirements of the IUT.

**Generating Filters and Setup:** Once the syntax tree is constructed, it will be applied to captured traces for playing the role of a filter. Meanwhile, the tree will also be sent to the tester with the definition of verdicts. According to different conditions, verdicts are defined as below:

- a) **PASS:** The message or trace satisfies the requirements.
- b) **FAIL:** The message or trace does not satisfy the requirements.

c) **TIME-FAIL:** The target message or trace cannot be observed within the maximum time limitation. Since we are working on online testing, a timeout is used to stop searching target message in order to provide the real-time status. The timeout value should be the maximum response time written in the protocol standard. If we cannot observe the target message within the timeout time, then a *Time-Fail* verdict will be assigned to this property. It has to be noticed that this verdict is only provided when no time constraint is required in the requirement. If any time constraint is required, the violation of this requirement will be concluded as *Fail*.

d) **INCONCLUSIVE:** Uncertain status of the properties. Different from offline testing, this verdict will not appear in the final results. It only exists at the beginning of the test or when the test is paused, in order to describe the indeterminate state of the properties (e.g. a property that requires a special occurrence on the potocol that did not occur yet).

e) **DATA-INC (Data Inconclusive):** In the testing process, some properties may be evaluated through traces containing only control portion (there is no data portion or the latter case mentioned in Step ‘Transferring’). If any property requires for testing the data portion, *Data-Inc* verdicts will be assigned to the property, due to the fact that no data portion can be tested. However, these *Data-Inc* verdicts will be eventually updated to *Pass* or *Fail* based on the data (coming from complete traces) analyzed on the tested properties. Currently we are using worst-case solution (all concluded as *Fail* verdicts). It won’t affect the overall results, since *Data-Inc* verdicts only represent a tiny proportion (less than 0.1%) of the whole traces in our experiments. However, expecting eventual contingencies, we plan to apply a support vector machine (SVM) approach [5] in the future.

**Filtering:** The incoming captured traces will go through the filtering module, and messages in the traces are filtered into different sets. The unnecessary messages irrelevant to any of the requirements are filtered into the “Unknown” set, and they will not go through the testing process. Finally, traces will be filtered to multiple optimized streams. This step will obviously reduce the processing time, since futile comparisons with irrelevant messages are omitted.

**Transferring:** The filtered traces are transferred (6a) to the tester when the tester is capable for testing. If the tester priority has to be decreased (e.g. the CPU and RAM must be used for another task on the user computer), a “load notification” (7) is provided to the monitor to transfer/store incoming traces.

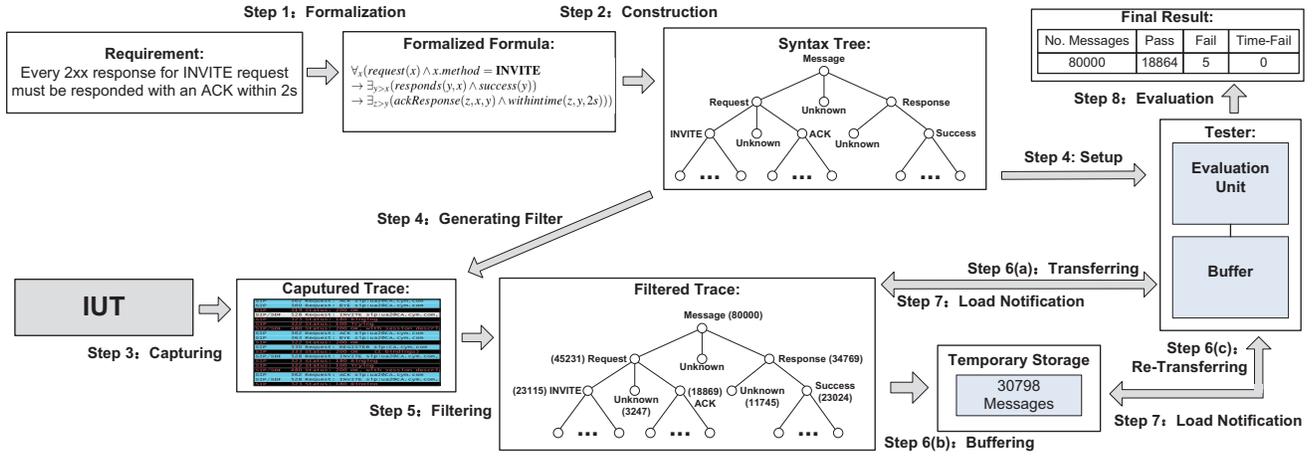


Fig. 2. Use Case for Testing Process

Based on the message format of the protocols to be tested, different buffering methods will be applied.

- If in the message format, the size of its header is larger than its body. Then the whole message will be buffered in the temporary storage.

- On the contrary, if the size of its header is equal or less than its body, then only the control portion of the packets are buffered (6b) in the temporary storage. Since not all the protocol requirements have specific needs on the data portion, only buffering the control portion will save a lot of memory space when buffering millions of messages.

When the tester is available (notification obtained), the stored traces are retransferred (6c) to the tester. In the latter case mentioned above, only the control portion of packets are provided. In both cases, the continuity of traces is ensured, since no packet will be dropped in any condition. If the protocol requirement has specific needs on the data portion, then the new verdict *Data-Inc* can be given and will be eventually updated to final verdicts by future analysis with the entire traces (the tester is indeed available again).

**Load Notification:** When the tester reaches its limit regarding the amount of data processable or is given a lower priority (e.g. to discharge the CPU / RAM), it sends a "Load Notification *Y*" to pause incoming filtered traces and store them in the temporary storage. When the tester is available back, a "Load Notification *N*" to release stored traces and to pursue incoming packets is sent. When captured traces from the IUT are transferred to the tester buffer, a checking overflow function will be called. If the buffer already reached to its maximum capacity, it will notify the IUT to redirect incoming traces to temporary storage in order to avoid the overflow. On the contrary, if the buffer is in a stable condition, it will send the available notification *N* to the temporary storage for releasing stored messages and to the IUT for returning back to normal transport process.

**Evaluation:** The tester checks whether the incoming traces satisfy the formalized requirements, and provides the final verdicts *Pass*, *Fail* or *Time-Fail* and temporary verdicts *Inconclusive* or *Data-Inc*.

## IV. EXPERIMENTS

### A. Environment

The IP Multimedia Subsystem (IMS) is a standardized framework for delivering IP multimedia services to users in mobility. It aims at facilitating the access to voice or multimedia services in an access independent way, in order to develop the fixed-mobile convergence. Most communication with its core network and between the services is done using the Session Initiation Protocol (SIP) [13]. For our experiments, communication traces were obtained through four ZOIPER<sup>1</sup> clients which are VoIP soft clients, meant to work with any IP-based communication systems and infrastructure. On the other side, the server is provided by Fonality<sup>2</sup>. The tests are performed in the virtual machines by opening a live capture on the client local interface.

### B. Test Results

For better understanding how our approach works, we illustrate a simple use case tested on one of the clients. As shown in Figure 2, we have a SIP requirement to be tested: "Every 2xx response for **INVITE** request must be responded with an ACK within 2s", which can be formalized to a formula as Step 1 shows.

This formula will be transformed to a syntax tree. When the syntax tree is generated and transferred to the IUT monitor, it will start to capture the trace and apply the syntax tree as a filter (Step 3 and 4) for captured messages. Meanwhile, the syntax tree will be applied in the tester as requirement. Once the captured trace is filtered into different sets (Step 5), it will check the Load Notification value first. Currently, the Load Notification value equals to *N*, which makes the tester available to test incoming traces. Then all incoming traces will be sent to the tester directly (Step 6a). As soon as the tester receives the trace, it tests the trace through the formalized property. When the tester is almost reaching to its maximum capacity, it will send a load notification value *Y* back to the monitor (Step 7 and 8). In this case, all incoming traces will be stored in the temporary storage (Step 6b) until the tester recovers to

<sup>1</sup><http://www.zoiper.com/softphone/>

<sup>2</sup><http://www.fonality.com>

Properties	Total Messages	Filtered out Messages	Filtered out Rate	Pass	Fail	Time-Fail	Incon	Data-Inc
Prop.1	2324506	1631797	70.19%	631271	0	61432	52	2164
Prop.2	2324506	1631797	70.19%	498124	194579	0	52	2164
Prop.3	2324506	1979904	85.17%	314923	0	29673	14	1086
Prop.4	2324506	1979904	85.17%	247257	97339	0	14	1086
Prop.5	2324506	2259032	97.18%	61550	0	3924	6	371

TABLE I. ONLINE TESTING RESULT FOR PROPERTIES

an available state (Step 6c). Finally, after our 2 hours testing process, we got 18864 ‘Pass’ verdicts, 5 ‘Fail’ verdicts caused by violation of the time constraint and no *Time-Fail* verdicts.

Secondly, we test our approach in a more complex environment. It has been performed to concurrently test five properties on a huge set of messages: “Prop.1: Every request must be responded”, “Prop.2: Every request must be responded within 8s”, “Prop.3: Every **INVITE** request must be responded”, “Prop.4: Every **INVITE** request must be responded within 4s” and “Prop.5: Every **REGISTER** request must be responded”.

The table I shows a snapshot of temporary testing verdicts after 3 hours online continuously testing. Benefited from the filtering function, more than 70% irrelevant messages are filtered out before testing process, which apparently reduce the cost of computing resources. Besides, numbers of *Fail* and *Time-Fail* verdicts can be observed. *Time-Fail* verdicts in Prop.1, Prop.3 and Prop.5 indicate that there are 61432, 29673 and 3924 messages respectively that cannot be observed within the timeout, in other words, they are lost during the communication between the client and the server. Besides, the ‘0’ *Fail* verdict indicates there is no error observed in the data portion for these three properties currently. On the other side, *Fail* verdicts reported in Prop.2 and Prop.4 indicate that there are 194579 and 97339 messages that cannot satisfy the time requirement. These *Fail* verdicts include the *Time-Fail* verdicts reported in Prop.1 and Prop.3, since lost messages also violate the time requirement.

Moreover, several ‘*Inconclusive*’ verdicts indicating the numbers of pending procedures for each property can be observed. We also used the control-portion-only buffering mechanism to test the usage of ‘*Data-Inc*’. All the buffered messages without data portion are successfully reported as ‘*Data-Inc*’ shown in Table I. Since they take a tiny proportion of whole traces (between 0.015% and 0.09%), we conclude them as *Fail* in the worst-case. During the whole testing process, our approach successfully handled this huge set of messages and did not suspend.

## V. CONCLUSION

This paper introduces a novel online passive testing approach to test conformance and performance of network protocol implementation. It allows to formally define relations between messages and message data, and then to use such relations in order to define the conformance and performance properties. The evaluation of the property returns a *Pass*, *Fail*, *Time-Fail*, *Inconclusive* or *Data-Inc* result, derived from run-time traces. To verify and test the approach, several SIP properties are designed to be evaluated. Our methodology has been implemented into a real-time IMS communications environment, and results from testing several properties online have been obtained successfully. Consequently, applying our

approach under billions of messages and extending more testers in a distributed environment is part of our future works. In that case, the efficiency and processing capacity of the approach will be scalably tested.

## REFERENCES

- [1] Mark R. Brown and Robert E. Tarjan. A fast merging algorithm. *Journal of the ACM*, 26(2):211–226, 1979.
- [2] Tien-Dung Cao, Patrick Félix, Richard Castanet, and Ismail Berrada. Online testing framework for web services. In *Third International Conference on Software Testing, Verification and Validation*, pages 363–372, 2010.
- [3] Xiaoping Che, Felipe Lalanne, and Stephane Maag. A logic-based passive testing approach for the validation of communicating protocols. In *Proceedings of the 7th International Conference on Evaluation of Novel Approaches to Software Engineering*, pages 53–64, 2012.
- [4] Xiaoping Che and Stephane Maag. A formal passive performance testing approach for distributed communication systems. In *Proceedings of the 8th International Conference on Evaluation of Novel Approaches to Software Engineering*, pages 74–84, 2013.
- [5] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.
- [6] Gordon Fraser, Franz Wotawa, and Paul Ammann. Testing with model checkers: a survey. *Software Testing and Verification Reliability*, 19(3):215–261, 2009.
- [7] Alfred Horn. On sentences which are true of direct unions of algebras. *Journal of Symbolic Logic*, 16(1):14–21, 1951.
- [8] E. Knauss and D. Damian. V:issue:lizer: Exploring requirements clarification in online communication over time. In *35th International Conference on Software Engineering (ICSE)*, pages 1327–1330, 2013.
- [9] Felipe Lalanne and Stephane Maag. A formal data-centric approach for passive testing of communication protocols. In *IEEE/ACM Transactions on Networking*, volume 21, pages 788–801, 2013.
- [10] D. Lee and R.E. Miller. Network protocol system monitoring—a formal approach with passive testing. *IEEE/ACM Transactions on Networking*, pages 14(2):424–437, 2006.
- [11] Gerardo Morales, Stéphane Maag, Ana R. Cavalli, Wissam Mallouli, Edgardo Montes de Oca, and Bachar Wehbi. Timed extended invariants for the passive testing of web services. In *IEEE International Conference on Web Services (ICWS)*, pages 592–599, 2010.
- [12] Robert E. Noonan. An algorithm for generating abstract syntax trees. *Computer Languages*, 10(3-4):225–236, 1985.
- [13] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, and J. Peterson. Sip: Session initiation protocol. 2002.
- [14] Margus Veanes, Colin Campbell, Wolfram Schulte, and Nikolai Tillmann. Online testing with model programs. In *Proceedings of the 10th European Software Engineering Conference*, pages 273–282, 2005.
- [15] Bachar Wehbi, Edgardo Montes de Oca, and Michel Bourdellès. Events-based security monitoring using mmt tool. In *IEEE Fifth International Conference on Software Testing, Verification and Validation (ICST)*, pages 860–863, 2012.
- [16] Wei Wei, Kyoungwon Suh, Bing Wang, Yu Gu, James F. Kurose, Donald F. Towsley, and Sharad Jaiswal. Passive online detection of 802.11 traffic using sequential hypothesis testing with tcp ack-pairs. *IEEE Transactions on Mobile Computing*, 8(3):398–412, 2009.

# Testing Model Transformation Programs using Metamorphic Testing \*

Mingyue Jiang<sup>1,3</sup>, Tsong Yueh Chen<sup>1</sup>, Fei-Ching Kuo<sup>1</sup>, Zhi Quan Zhou<sup>2</sup>, Zuohua Ding<sup>3</sup>

<sup>1</sup>Department of Computer Science and Software Engineering  
Swinburne University of Technology, Hawthorn, VIC 3122, Australia

<sup>2</sup>School of Computer Science and Software Engineering  
University of Wollongong, Wollongong, NSW 2522, Australia

<sup>3</sup>Laboratory of Scientific Computing and Software Engineering  
Zhejiang Sci-Tech University, Hangzhou, Zhejiang, 310018, China

## Abstract

*Model transformations are crucial for the success of Model Driven Engineering. Testing is a prevailing technique of verifying the correctness of model transformation programs. A major challenge in model transformation testing is the oracle problem, which refers to the difficulty or high cost in determining the correctness of the output models. Metamorphic Testing alleviates the oracle problem by making use of the relationships among the inputs and outputs of multiple executions of the target function. This paper investigates the effectiveness and feasibility of metamorphic testing in testing model transformation programs. Empirical results show that metamorphic testing is an effective testing method for model transformation programs.*

**Keywords:** Metamorphic Testing, Model Transformation, Software Quality, Software Testing, Test Oracle

## 1. Introduction

Model transformation, which refers to the automatic process of transforming one model into another, is a vital element of Model Driven Engineering (MDE). In MDE, model transformations are usually used to transform models between different languages or different abstraction levels. In this way, models are automatically transformed and refined until code of final software is produced. The success of MDE critically depends on the correctness of model transformation programs as an incorrect transformation will result in incorrect models and the final software.

Testing is a prevailing technique of verifying the correctness of model transformation programs. A major challenge

in the testing process is the *oracle problem*: In general, it is difficult to obtain test oracles for model transformation programs [8]. We propose the technique of *Metamorphic Testing* (MT) to alleviate the oracle problem in testing model transformation programs. MT has been successfully applied to detect real-world faults [3, 5]. In MT, programs are tested against their expected necessary properties. A major difference between MT and all the other testing methods for model transformation is that the properties used by MT are relationships among the inputs and outputs of *multiple* executions of the target program (known as *metamorphic relations*), whereas the properties used by the other methods focus on the input and output of a *single* execution. Another difference is that when testing model transformations, metamorphic relations (MRs) can be extracted from informal specifications, whereas most of the other methods rely on formal specifications.

## 2. Model Transformation

Model transformation is a critical activity in MDE, which is about the generation of target models from source models. A framework of model transformation is given in Fig. 1. The *source metamodel* (MMa) and the *target metamodel* (MMb) describe the static information of models, which are manipulated by the model transformation. The *source* (Ma) and *target* (Mb) models conform to their respective metamodels. The *transformation model* (Mt) refers to an implementation (program) of the model transformation, and MMt is the metamodel of Mt. The model transformation program (Mt) takes a source model as input and produces a target model as output.

There are different transformation languages, of which a popular one is the ATLAS Transformation Language (ATL) [6]. We conducted a case study using a popular model transformation program written in ATL, namely,

\*This project was supported in part by Australian Research Council (Project ID: LP100200208), and National Natural Science Foundation of China (Grant Nos. 61170015 and 61210004).

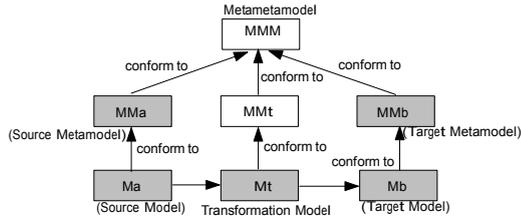


Figure 1. A framework of model transformation

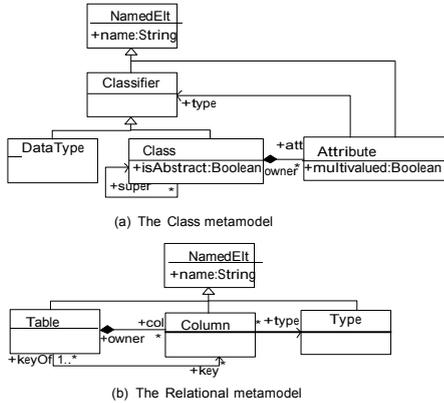


Figure 2. The metamodels of Class2Relational

Class2Relational, which is an “advanced example” open-sourced in the ATL Transformations Zoo<sup>1</sup> and is often used as a subject program by various experimentations [4].

In Class2Relational, Class model is the source model and Relational model is the target model. The Class and Relational models conform to the Class and Relational metamodels, respectively (see Fig. 2). In a Class model, each DataType represents a primitive data type, and each Class has a name and a set of Attributes, each of which can be single-valued or multi-valued and has either DataType or Class as its type. In a Relational model, each Table contains a name, a reference to its key Columns and a set of Columns, each of which is described by its name and type. The following describes the requirements of how Class2Relational should transform a Class model into a Relational model:

- (1) For each DataType, a Type is created.
- (2) For each Class, a Table (Type1) is created. Their names are identical. The Table contains a key Column, whose name is “objectId” and type is a specific type (In this example, it refers to Integer). Each Attribute of the Class is also manipulated, which is described in the following.
- (3) For each single-valued Attribute of type DataType, a Column is created, and their names and types are identical.
- (4) For each multi-valued Attribute of type DataType, a Table (Type2) is created. Two Columns of the Table are also created. One is the identifier Column (with a specific type)

and the other contains name and type of the Attribute.

(5) For each single-valued Attribute of type Class, a Column is created. The name of the Column is the Attribute’s name + “id”, and the type of the Column is a specific type.

(6) For each multi-valued Attribute of type Class, a new Table (Type3) is created, which has two Columns with specific types (one is the identifier Column, and the other is named attribute.name + “id”).

(7) The name of the Table (Type2, Type3) is set to str1+ “\_”+ str2, str1 represents the name of the Class which contains the Attribute, and str2 represents the name of the Attribute. The Table’s identifier Column is named str1+ “id”.

The model transformation program class2relational.atl was written according to the above requirements. An example Class model is given in Table 1 (left column), written in the XML Metadata Interchange (XMI) format. After executing class2relational.atl with this Class model as input, the output model, that is, the corresponding Relational model, is shown in Table 1 (right column). Obviously, it is not difficult to manually verify the correctness of the transformation. It should be noted that real-world models are much larger and much more complex than the above example. Checking the correctness of the transformations of real-world models is therefore a very difficult task.

### 3. Metamorphic Testing

Metamorphic Testing (MT) [3] is a methodology designed to alleviate the oracle problem. Different from conventional testing strategies, MT uses some specific properties known as Metamorphic Relations (MRs) involving multiple test cases and their outputs.

Let  $p$  be a program implementing function  $f$ . To test  $p$ , suppose a set of test cases  $T = \{t_1, t_2, \dots, t_n\}$  ( $n > 0$ ) have been generated using some test case selection strategies (such as black-box, white-box or random testing). Test cases in  $T$  are referred to as original test cases. Based on the knowledge of  $f$ , some MRs can be identified. For each MR, a set of follow-up test cases can be generated for  $T$ . Suppose  $t'_i$  is a follow-up test case for the original test case  $t_i$ , then  $(t_i, t'_i)$  is called a metamorphic test group [12]. MT runs the original and follow-up test cases and checks whether the outputs satisfy the MRs, regardless of the availability of an oracle for each individual test case.

### 4. Application of Metamorphic Testing to Model Transformation

The procedure is outlined as follows: First, identify MRs and construct a set of original test models. For each MR, generate follow-up test models based on the original test models. Then execute the model transformation program using both the original and follow-up test models, and

<sup>1</sup><http://www.eclipse.org/atl/atlTransformations>

**Table 1. A Class model (left column) and the corresponding Relational model (right column)**

<pre> &lt;?xml version="1.0" encoding="ASCII"?&gt; &lt;xmi:XMI xmi:version="2.0" xmlns:xmi= "http://www.omg.org/XMI" xmlns="Class"&gt; &lt;DataType name="Integer"/&gt; &lt;DataType name="String"/&gt; &lt;Class name="C1"&gt;   &lt;attr name="A1" multiValued="false" type="1"/&gt;   &lt;attr name="A2" multiValued="true" type="1"/&gt; &lt;/Class&gt; &lt;Class name="C2"&gt;   &lt;attr name="A3" multiValued="false" type="2"/&gt;   &lt;attr name="A4" multiValued="true" type="2"/&gt; &lt;/Class&gt;&lt;/xmi:XMI&gt; </pre>	<pre> &lt;?xml version="1.0" encoding="ASCII"?&gt; &lt;xmi:XMI xmi:version="2.0" xmlns:xmi= "http://www.omg.org/XMI" xmlns="Relational"&gt; &lt;Table name="C1" key="/0/@col.0"&gt;   &lt;col name="objectId" keyOf="/0" type="2"/&gt;   &lt;col name="A1" type="3"/&gt; &lt;/Table&gt; &lt;Table name="C2" key="/1/@col.0"&gt;   &lt;col name="objectId" keyOf="/1" type="2"/&gt;   &lt;col name="A3Id" type="2"/&gt; &lt;/Table&gt; &lt;Type name="Integer"/&gt; &lt;Type name="String"/&gt; &lt;Table name="C1_A2"&gt;   &lt;col name="C1Id" type="2"/&gt;   &lt;col name="A2" type="3"/&gt; &lt;/Table&gt; &lt;Table name="C2_A4"&gt;   &lt;col name="C2Id" type="2"/&gt;   &lt;col name="A4Id" type="2"/&gt; &lt;/Table&gt;&lt;/xmi:XMI&gt; </pre>
--	---

collect the output models. Finally, check the relationship among the original and follow-up test models and their respective output models against the MR. Any violation of MR implies that the program under test is faulty.

A key activity in MT is the identification of MRs, which requires knowledge of the model transformation requirements. Once MRs are identified, they can be used for testing irrespective of the programming language of the model transformation software. We are now going to present some MRs for the subject program *Class2Relational*.

We will use *Type1\_Table*, *Type2\_Table* and *Type3\_Table* to represent the aforementioned three kinds of *Tables* of the *Relational* model and use *Specific\_Columns* to represent *Columns* whose *type* refers to the specific type. Let  $C_1$  denote the original test model and  $C_2$  denote the follow-up test model with  $T_1$  and  $T_2$  being their output models, respectively. We use  $X.Y$  to indicate the element  $Y$  of model  $X$ , and  $X.#Y$  to denote the number of  $Y$  of model  $X$ .

Based on the requirements of *Class2Relational*, the following categories of MRs can be identified:

**1. Reset of values of some attributes of the test model.**

**MR1.1:** If we modify the values of some attributes of  $C_1$  to obtain  $C_2$  (the modified values are legal), then

$$T_2.\#Type1\_Tables = T_1.\#Type1\_Tables,$$

$$\text{and } T_2.\#Types = T_1.\#Types.$$

**MR1.2:** Suppose *Attr* is an *attribute* of *Class Cla* in  $C_1$ , and  $C_2$  is constructed by reversing the value of *Attr.multivalued*.

- If *Attr.multivalued* is *true* (It is *false* in  $C_2$ ), then we have:

$(T_2.Tab.Columns \setminus T_1.Tab.Columns) = \{Col / Col.name \text{ contains } Attr.name\}$ , where *Tab* is the *Table* whose name equals *Cla.name* and  $\setminus$  is the set difference operator, which will be used hereafter in this paper,

$T_2.\#T_{s1} = (T_1.\#T_{s1} - 1)$ , where  $T_{s1}$  is a set composed of *Tables* whose name contains *Attr.name*,

and  $T_2.\#T_{s2} = (T_1.\#T_{s2} - 1)$ , where  $T_{s2}$  is a set composed of *Tables* whose name contains *Cla.name*.

- If *Attr.multivalued* is *false*, then  $(T_1.Tab.Columns \setminus T_2.Tab.Columns) = \{Col / Col.name \text{ contains } Attr.name\}$ ,

$$T_2.\#T_{s1} = (T_1.\#T_{s1} + 1), \text{ and } T_2.\#T_{s2} = (T_1.\#T_{s2} + 1).$$

**MR1.3:** Suppose *Attr* is an *attribute* of *Class Cla* in  $C_1$ , and  $C_2$  is constructed by changing *Attr*'s *type*.

- If *Attr.type* refers to a *DataType* (*Attr.type* will refer to a *Class* in  $C_2$ ), then

$$T_2.\#Specific\_Columns = (T_1.\#Specific\_Columns + 1).$$

- If *Attr.type* refers to a *Class*, then we have:

$$T_2.\#Specific\_Columns = (T_1.\#Specific\_Columns - 1).$$

**2. Insertion of an element into the test model.**

**MR2.1:** Construct  $C_2$  by adding a *DataType* into  $C_1$ , then

$$T_2.\#Columns = T_1.\#Columns,$$

$$T_2.\#Types = (T_1.\#Types + 1), \text{ and } T_2.\#Tables = T_1.\#Tables,$$

**MR2.2:** Construct  $C_2$  by adding a *Class* into  $C_1$ , then  $T_2$

$$\#Type1\_Tables = (T_1.\#Type1\_Tables + 1), T_2$$

$$\#Columns > T_1.\#Columns, T_2.\#Types = T_1.\#Types,$$

$$\text{and } T_2.\#specific\_Columns > T_1.\#specific\_Columns.$$

**MR2.3:** Construct  $C_2$  by adding an *Attribute* to  $C_1$ .

**MR2.3.1:** The added *Attribute* is a single-valued *Attribute* of *DataType*, then  $T_2.\#Columns = (T_1.\#Columns + 1)$ ,

$$T_2.\#Tables = T_1.\#Tables, T_2.\#Types = T_1.\#Types,$$

$$\text{and } T_2.\#specific\_Columns = T_1.\#specific\_Columns.$$

**MR2.3.2:** The added *Attribute* is a multi-valued *Attribute* of *DataType*, then  $T_2.\#Columns = (T_1.\#Columns + 2)$ ,

$$T_2.\#Tables = (T_1.\#Tables + 1), T_2.\#Types = T_1.\#Types,$$

$$T_2.\#Type1\_Tables = T_1.\#Type1\_Tables,$$

$$T_2.\#Type2\_Tables = (T_1.\#Type2\_Tables + 1),$$

$$\text{and } T_2.\#specific\_Columns = (T_1.\#specific\_Columns + 1).$$

**MR2.3.3:** The added *Attribute* is a single-valued *Attribute* of *Class*, then  $T_2.\#Columns = (T_1.\#Columns + 1)$ ,

$$T_2.\#Tables = T_1.\#Tables, T_2.\#Types = T_1.\#Types$$

$$\text{and } T_2.\#specific\_Columns = (T_1.\#specific\_Columns + 1).$$

**MR2.3.4:** The added *Attribute* is a multi-valued *Attribute* of *Class*, then  $T_2.\#Columns = (T_1.\#Columns + 2)$ ,

$$T_2.\#Tables = (T_1.\#Tables + 1), T_2.\#Types = T_1.\#Types$$

$$T_2.\#Type1\_Tables = T_1.\#Type1\_Tables,$$

$$T_2.\#Type3\_Tables = (T_1.\#Type3\_Tables + 1),$$

$$\text{and } T_2.\#specific\_Columns = (T_1.\#specific\_Columns + 2).$$

### 3. Deletion of data from the test model according to the output model

**MR3.1:** Suppose *Col* is a *Column* of *Table Tab* (*Tab* is a *Table* of Type1) in the output mode of  $C_1$ . Construct  $C_2$  by deleting information related to *Col* of  $C_1$ .

- If *Col* is related to a single-valued *Attribute*, then  $(T_1.Tab.Columns \setminus T_2.Tab.Columns) = \{Col\}$ .
- If *Col* is related to a multi-valued *Attribute*, then  $(T_1.Tables \setminus T_2.Tables) = \{T / T.name = Tab.name + \_ + Col.name\}$ , and  $(T_1.Columns \setminus T_2.Columns) = \{Col / Col.name \text{ either contains } Tab.name \text{ or } Col.name\}$ .

**MR3.2:** Suppose *Tab* is a *Table* of Type1 in the output model of  $C_1$ , and  $C_2$  is constructed by deleting information related to *Tab* of  $C_1$ . Then we have:  $(T_1.Tables \setminus T_2.Tables) = \{T / T.name = Tab.name + str, \text{ where } str \text{ can be empty}\}$ .

### 4. Interchange of data in the test model

**MR4** Suppose  $Cl_{a_1}$  and  $Cl_{a_2}$  are two *Classes* of  $C_1$ , and  $Attr_1$  and  $Attr_2$  are *Attributes* of  $Cl_{a_1}$  and  $Cl_{a_2}$ , respectively.  $C_2$  is constructed by interchanging the data of  $Attr_1$  and  $Attr_2$  (that is, in  $C_2$ ,  $Attr_1$  becomes an *Attribute* of  $Cl_{a_2}$  and  $Attr_2$  becomes an *Attribute* of  $Cl_{a_1}$ ).

- If  $Attr_1$  and  $Attr_2$  are both single-valued *Attributes*, then  $T_2.Columns = T_1.Columns, T_2.\#Tables = T_1.\#Tables,$   
 $DiffTable = (T_2.Tables \setminus (T_2.Tables \cap T_1.Tables)) = \{Tab / Tab.name = Cl_{a_1}.name \text{ or } Cl_{a_2}.name\}$ , where  $\cap$  is the set intersection operator, which will be used throughout this paper, and  $DiffTables.size = 2$ , where *size* is the number of elements in the set.

- If  $Attr_1$  and  $Attr_2$  are both multi-valued *Attributes*, then  $T_2.Columns = T_1.Columns, T_2.\#Tables = T_1.\#Tables,$   
 $DiffTable = (T_2.Tables \setminus (T_2.Tables \cap T_1.Tables)) = \{Tab / Tab.name \text{ contains } Attr_2.name \text{ and } Cl_{a_1}.name \text{ or contains } Attr_1.name \text{ and } Cl_{a_2}.name\}$  and  $DiffTables.size = 2$ .

- If one of these two *attributes* (namely,  $Attr_1$ ) is single-valued and the other (namely,  $Attr_2$ ) is multi-valued, then  $T_2.\#Columns = T_1.\#Columns, T_2.\#Tables = T_1.\#Tables,$   
 $DiffColumns = (T_2.Columns \setminus (T_2.Columns \cap T_1.Columns)) = \{Col / Col.name \text{ contains } Cl_{a_1}.name\}$  and  $DiffColumns.size = 1$ ,

$DiffTable1 = (T_2.Type1.Tables \setminus (T_2.Type1.Tables \cap T_1.Type1.Tables)) = \{Tab / Tab.name = Cl_{a_1}.name \text{ or } Tab.name = Cl_{a_2}.name\}$  and  $DiffTable1.size = 2$ ,

$DiffTable2 = (T_2.Type2,3.Tables \setminus (T_2.Type2,3.Tables \cap T_1.Type2,3.Tables)) = \{Tab / Tab.name \text{ contains } Attr_2.name \text{ and } Cl_{a_1}.name\}$  and  $DiffTable2.size = 1$ .

## 5. Empirical Evaluation

### 5.1. Experimental procedure

We conducted empirical evaluation of MT using the model transformation program *class2relational.atl*, which has 107 lines of code and contains 6 ATL rules and 1 ATL

helper. Using the MRs described in Section 4, the testing procedure consists of the following three steps:

- (1) Generation of original test models. The set of original test models were generated randomly in such a way that (i) they all conform to the source metamodel, (ii) all elements of the source metamodel are covered, and (iii) different original test models have different values in the same attributes in order to maximize diversity.
- (2) Construction of follow-up test models. Different MRs will result in different follow-up test models. These models were generated automatically.
- (3) Verification of test results. This step was also performed automatically by our test script against the MRs.

A total of 100 *Class* models were generated as the original test models for testing the subject program *class2relational.atl*. No violation of MRs was detected. This is expected as *class2relational.atl* is a popular and open-source program. In order to evaluate the fault-detection effectiveness of MT, we then applied *mutation analysis* [7] to generate 20 non-equivalent mutants from *class2relational.atl*. Details of the mutants are shown in Table 2, where  $M_i$  denotes the  $i^{th}$  mutant.

### 5.2. Results of experiments

We applied MT to test every mutant using the 100 original test models. Results of experiments are summarized in Table 3 in terms of the violation ratio which is defined as the ratio of violated metamorphic test groups among all used metamorphic test groups. The last row shows the average violation ratio for each individual MR, and the last column shows the average violation ratio for each individual mutant. It is observed that every mutant has some violated metamorphic test groups. In other words, all seeded faults are detected.

Table 3 shows that the average violation ratios of MRs range from 0.00 to 0.54. This result is consistent with many other MT studies, which reported that different MRs can have very different fault-detection effectiveness. Table 3 also shows that the fault-detection effectiveness of an MR is mutant dependent. Consider MR2.3.1, for instance, it has varied violation ratios for M3, M9, M19, M20, which are 1.00, 0.08, 0.00 and 1.00, respectively.

The effectiveness of MT can be further analyzed using metamorphic test groups. For each mutant,  $100 \times 12 = 1,200$  metamorphic test groups have been executed. Therefore, there is a total of  $1,200 \times 20 = 24,000$  metamorphic test groups. The total number of violated metamorphic test groups is 5,240, which gives the overall effectiveness of MT (in terms of violated metamorphic test groups) to be  $5,240 / 24,000 = 22\%$ . This result shows that MT is quite effective because a failure will be revealed after running about 5 metamorphic test groups on average.

**Table 2. Details of mutants of *class2relational.atl***

Mutant	Line number	Original code	New code	mutation operator
M1	42	type<-a.type	type<-a.owner	ROCC
M2	77	thisModule.objectIdType	a.type	RSCC
M3	37	a.type.ocIsKindOf(CLA!DataType)	a.ocIsKindOf(CLA!DataType)	RSMD
M4	56	name<-a.owner.name+'_'+a.name	name<-a.name+'_'+a.name	RSMD
M5	88	name<-a.owner.name+'_'+a.name	name<-a.name+'_'+a.name+	RSMD
M6	60	name<-a.owner.name+'Id'	name<-a.name+'Id'	RSMD
M7	56	name<-a.owner.name+'_'+a.name	name<-a.owner.name+'_'+a.owner.name	RSMA
M8	7	select(e / e.name = 'Integer')	select(e / true)	CFCD
M9	11	c:CLA!Class	c:CLA!Class(c.attr.size())>0)	CFCA
M10	29	REL!Type	REL!Table	CCCR
M11	29	REL!Type	REL!Column	CCCR
M12	21	type<-thisModule.objectIdType		CACD
M13	57	col<-Sequence/id,value/		CACD
M14	57	col<-sequence/id,value/	col<-sequence/value/	CACD
M15	61	type<-thisModule.objectIdType		CACD
M16	89	col<-Sequence(id,foreignKey)		CACD
M17	89	col<-sequence/id,foreignKey/	col<-sequence/id/	CACD
M18	7	select(e/e.name='Integer')	select(e/not(e.name='Integer'))	CFCP
M19	52, 84	a.type.ocIsKindOf(CLA!DataType) a.type.ocIsKindOf(CLA!Class)	not a.type.ocIsKindOf(CLA!DataType) not a.type.ocIsKindOf(CLA!Class)	CFCP
M20	37, 72	a.type.ocIsKindOf(CLA!DataType) a.type.ocIsKindOf(CLA!Class)	not a.type.ocIsKindOf(CLA!DataType) not a.type.ocIsKindOf(CLA!Class)	CFCP

**Table 3. Results of experiments: violation ratios**

	MR 1.1	MR 1.2	MR 1.3	MR 2.1	MR 2.2	MR 2.3.1	MR 2.3.2	MR 2.3.3	MR 2.3.4	MR 3.1	MR 3.2	MR 4	Average violation ratio for each mutant
M1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.73	0.06
M2	0.00	0.00	0.55	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.87	0.49	0.24
M3	0.00	0.47	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.45	0.16
M4	0.00	0.59	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.41	0.77	0.73	0.29
M5	0.00	0.34	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.25	0.00	0.63	0.19
M6	0.00	0.00	0.00	0.00	0.00	0.00	0.94	0.00	0.00	0.00	0.00	0.49	0.12
M7	0.00	0.58	0.00	0.00	0.00	0.00	0.94	0.00	0.00	0.05	0.00	0.81	0.20
M8	0.00	0.00	0.50	0.00	0.50	0.00	0.50	0.50	0.50	0.00	0.00	0.45	0.25
M9	0.00	0.00	0.00	0.00	0.54	0.08	0.08	0.08	0.05	0.03	0.00	0.55	0.12
M10	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.47	0.12
M11	0.00	0.00	1.00	1.00	1.00	0.00	1.00	1.00	1.00	0.00	0.77	0.53	0.61
M12	0.00	0.00	0.00	0.00	0.49	0.00	0.00	0.00	0.00	0.00	0.00	0.04	0.07
M13	0.00	0.00	0.41	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.54	0.16
M14	0.00	0.00	0.54	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.53	0.17
M15	0.00	0.00	0.54	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.56	0.18
M16	0.00	0.00	0.60	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.50	0.18
M17	0.00	0.00	0.49	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.46	0.16
M18	0.00	0.00	1.00	0.00	1.00	0.00	1.00	1.00	1.00	0.00	0.00	0.49	0.46
M19	0.00	0.00	0.49	0.00	0.00	0.00	1.00	0.00	1.00	0.00	0.89	0.46	0.32
M20	0.00	0.00	0.48	0.00	0.00	1.00	0.00	1.00	0.00	0.00	0.87	0.44	0.32
<b>Average violation ratio for each MR</b>	0.00	0.10	0.33	0.10	0.18	0.10	0.47	0.23	0.33	0.04	0.20	0.54	

### 5.3. A further analysis of the effectiveness of MRs

Table 3 shows that the fault-detection effectiveness of different MRs can be very different: MR4 was violated by every mutant, but MR1.1 was never violated. The most effective MR is MR4. Its average violation ratio is 0.54. That is, on average, more than half of its metamorphic test groups can reveal a failure. MR4 is highly effective because it makes use of more information of the transformation requirements than the remaining MRs. MR4 checks almost all data items of the *Relational* model, taking their concrete values into consideration, instead of just comparing the numbers of some elements. The other MRs (ex-

cept the worst one, MR1.1) generate follow-up test models by adding or deleting some elements, or resetting the attributes' values of some elements of the original test model. Their average violation ratios range from 0.10 to 0.47. They are less effective than MR4 because they check only certain parts of the *Relational* model. We analyzed each MR together with all the mutants that violated it. For any given pair of mutant and MR, a high violation ratio will be intuitively expected if the fault in the mutant is relevant to the MR.

MR1.1, which constructs follow-up test models by changing the values of some arbitrary attributes, was the least effective MR: It did not detect any violation. The rea-

son for this is twofold. First, in each and every metamorphic test group generated by MR1.1, the original and follow-up test case executions are almost identical in the sense that the same statements of the subject program are exercised (and in the same sequence). The original and follow-up output models generated in this way are therefore very similar. As a result, MR1.1 is very likely to be satisfied. This observation confirms the findings of Chen et al. [2] and Cao et al. [1]: an effective MR should make the original and follow-up test case executions as different as possible. Secondly, MR1.1 checks the test results at a quite high abstraction level by ignoring many details of the output models. Consequently, even if an output model is incorrect, the incorrect data item buried in the output model is not checked by MR1.1 and hence a violation cannot be detected. This finding shows that an effective MR should look at the details of the output as much as possible.

We have obtained two useful guidelines. First, MRs whose original and follow-up test case executions are very different, are likely to have a higher chance of detecting a failure than those whose original and follow-up test case executions are similar. Secondly, an effective MR should involve detailed information from the requirements specification as much as possible and as complete as possible.

## 6. Discussions and Conclusion

We propose to apply Metamorphic Testing (MT) to alleviate the oracle problem in testing model transformation programs. To evaluate the effectiveness of the proposed approach, a case study has been conducted using *Class2Relational* and mutation analysis. The empirical results show that MT can effectively detect model transformation faults. We used Metamorphic Relations (MRs) involving four kinds of operations, namely, addition of elements, deletion of elements, alteration of attribute's values, and interchange of elements. We have obtained two guidelines for applying MT to model transformation programs. The first guideline is to select MRs whose original and follow-up test case executions are significantly different. The second guideline is to select MR that involves as many details of the model transformation as possible from the transformation requirements. These two guidelines are appropriate for the selection of MRs for any model transformation programs.

Many applications have a model transformation component or have been developed using model transformations. Examples of the former include software development tools that use model transformations to generate the application code [10]. Examples of the latter include context-aware pervasive systems [9] and secure XML data warehouses [11] which are developed using Model Driven Development (MDD) method. Obviously, the correctness

of the model transformations will affect the quality of the final systems. MT can be used to test the model transformations in such applications.

## References

- [1] Cao Y., Zhou Z.Q., Chen T.Y. On the correlation between the effectiveness of metamorphic relations and dissimilarities of test case executions. In *Proceedings of the 13th International Conference on Quality Software (QSIC'13)*, pages 153–162, 2013.
- [2] Chen T.Y., Huang D.H., Tse T.H., Zhou Z.Q. Case studies on the selection of useful relations in metamorphic testing. In *Proceedings of the 4th Ibero-American Symposium on Software Engineering and Knowledge Engineering (JI-SIC'04)*, pages 569–583, 2004.
- [3] Chen T.Y., Kuo F.-C., Towey D., Zhou Z.Q. Metamorphic testing: Applications and integration with other methods. In *Proceedings of the 12th International Conference on Quality Software (QSIC'12)*, pages 285–288, 2012.
- [4] Guerra E., Lara J.D., Wimmer M., Kappel G., Kusel A., Retschitzegger W., Schönböck J., Schwinger W. Automated verification of model transformations based on visual contracts. *Automated Software Engineering*, 20:5–46, 2013.
- [5] Harman M., McMin P., Shahbaz M., Yoo S. A comprehensive survey of trends in oracles for software testing. Technical report, Technical Report (CS-13-01), Department of Computer Science, University of Sheffield, 2013.
- [6] Jouault F., Allilaire F., Bézivin J., Kurtev I. ATL: A model transformation tool. *science of computer programming. IEEE Transactions on Software Engineering*, 72(1-2):21–39, 2008.
- [7] Mottu J.M., Baudry B., Traon Y.L. Mutation analysis testing for model transformation. In *Proceedings of the Second European Conference on Model Driven Architecture: Foundations and Applications (ECMDA-FA'06)*, pages 376–390, 2006.
- [8] Mottu J.M., Baudry B., Traon Y.L. Model transformation testing: Oracle issue. In *Proceedings of International Conference on Software Testing Verification and Validation*, pages 105–112, 2008.
- [9] Serral E., Valderas P., Pelechano V. Towards the model driven development of context-aware pervasive systems. *Pervasive and Mobile Computing*, 6(2):254–280, 2010.
- [10] Thang N.X., Zapf M., Geihs K. Model driven development for data-centric sensor network applications. In *Proceedings of the 9th International Conference on Advances in Mobile Computing and Multimedia*, pages 194–197, 2011.
- [11] Vela B., Blanco C., Fernández-Medina E., Marcos E. A practical application of our mdd approach for modeling secure xml data warehouses. *Decision Support Systems*, 52(4):899–925, 2012.
- [12] Xie X.Y., Wong W.E., Chen T.Y., Xu B.W. Metamorphic slice: An application in spectrum-based fault localization. *Information and Software Technology*, 55(5):866–879, 2013.

# Reactive Variability Realization with Test-Driven Development and Refactoring

Glauco Silva Neves

Informatics and Statistics Department - INE  
Federal University of Santa Catarina - UFSC  
Florianópolis, Brazil  
glauco.neves@posgrad.ufsc.br

Patrícia Vilain

Informatics and Statistics Department - INE  
Federal University of Santa Catarina - UFSC  
Florianópolis, Brazil  
vilain@inf.ufsc.br

**Abstract**— Software product line is a practice that has proven its advantages since it can offer to a company the reduction of time to market, the decrease of development costs, the increase of productivity and the improvement of the final product quality. However, this practice requires a high initial investment and offers long-term risks to dynamic markets where changes are difficult to predict. One of these markets is the mobile application development, which presents a growing demand, with smartphone and tablets having already surpassed sales of PCs and notebooks. Currently, proposals bring the advantages of software product line for dynamic markets through the use of agile software development practices, which is called Agile Product Line Engineering (APLE). This paper investigates the use of test-driven development (TDD) and refactoring techniques for performing reactive variability in APLE. The variability mechanism chosen is the configuration file that allows achieving more than one platform, an important problem in mobile application development. In that manner, new products can be built as needed, without the high upfront investment, but with a code easier to maintain.

**Keywords**- *Software product line, test-driven development, refactoring, variability realization*

## I. INTRODUCTION

A Software Product Line (SPL) is a set of applications that share common artifacts addressing the need of a particular market segment. This practice has already proven the advantages of reducing the time to market, decreasing development costs, increasing productivity and improving the quality of applications [1]. There are two essential activities in the SPL development: domain engineering and application engineering [1].

In domain engineering an initial planning to identify the commonalities and variability points of the applications is done and the system architecture is defined. Moreover, the resources that will be reused in applications are produced and become part of the core assets repository. Application engineering, in its turn, focuses on understanding the needs of each specific application and then reusing the resources (e.g. architecture, code, tests) in the core assets repository through the activation of variability points and selection of components necessary to satisfy the requirements.

Despite its advantages, a SPL requires a high upfront and long-term investment to design and to develop the core assets repository, hindering the SPL use in dynamic markets due to the risk of unforeseen changes and the cost of developing artifacts that may no longer be reused. To overcome this difficulty, there is a proposal of combining agile software development practices with SPL, resulting in the Agile Product Line Engineering (APLE) [2].

One example of a dynamic market that has grown rapidly is the one of applications for mobile devices. Smartphones and tablets have surpassed PC and laptop sales [3]. New and different devices are constantly released with different connection capacities, pixels densities, resolutions, screen sizes, sensors (accelerometer, barometer, gyroscope), GPS and storage [4] besides using different operating systems such as iOS, Android, Windows Phone, and others.

Considering this scenario where changes are constant, APLE is an interesting proposal for mobile application companies that want to make use of SPL but do not want to suffer the disadvantages the traditional SPL development.

This article uses the APLE approach to define a process for developing reactive SPLs using agile practices. The SPL variability is carried out on demand using Test-Driven Development (TDD) and refactoring practices. The variability mechanism used is the configuration file.

This paper is organized as follows. Section 2 describes SPL combined with agile software development. In section 3 the proposed process is explained. Section 4 shows an example of a mobile application that was developed by applying the proposed process. Related work is discussed in section 5. Finally, section 6 presents some conclusions.

## II. AGILE PRODUCT LINE ENGINEERING

There are several reasons for combining SPL practices with agile software development practices [2]. This combination can be applied when:

1. There is not much knowledge about the domain to perform domain engineering;
2. It is not possible to predict changes in product requirements;

3. There is a need to decrease the risk of developing artifacts that may not be reused, due to market modifications.

According to Silva [5], Scrum and Extreme Programming (XP) are the most used agile methods in SPL development. Some studies combine Scrum and XP due to the fact that the nature of these methods has different focus. While Scrum focuses on project management, XP focuses on development practices. One of the most famous practices of XP is the Test-Driven Development (TDD).

TDD is a way of programming where the coding tasks are performed in small cycles according to the mantra Red/Green/Refactor [6]. In the red phase, a failed unit test is written, which may not even compile. In the green phase, code is modified in the simplest way just to pass the test. And in the last phase, refactor, the code is modified in order to improve and maintain the behavior. It is in the refactor phase that design decisions are made, one at a time.

TDD also assists in preventing bugs and may serve as documentation of the system. According to Beck [6], TDD can also aid the framework development because it focuses on what is needed instead of attempting to accommodate different features at once. As new features arise, the code is tested and refactored, eliminating duplicated code. The common code is placed in separate from the specific code, facilitating the reuse of common code later.

There are two main strategies for building a SPL: proactive and reactive [1]. The more traditional approach is proactive, where scope, architecture, components, and other resources, are defined in early stages, anticipating commonalities and specificities. In the reactive approach one or more applications are developed at first, then the core assets are extracted from them. It is also possible an incremental approach, combining the ideas of both strategies at different times.

In this work, we choose the reactive approach to be used in conjunction with agile practices because among its advantages is the lower cost for development, since the core assets repository is not constructed upfront [1]. In this way, the risk of developing useless components is lower since there is no need to provide the variations of products in advance and the knowledge about the domain can grow as applications are developed.

### III. A PROCESS FOR REACTIVE SPL DEVELOPMENT

The goal of this work is to define a process for developing a reactive SPL. Such process applies TDD and refactoring agile practices to product conception and variability realization.

During this process, the first application is developed focusing only on satisfying its existing requirements. This development is test-driven and its results include the application and a unit test suite that ensures its behavior.

During the development of the second application, its requirements are either mapped to new features or the existing features can be modified. In the case of new features, new unit tests and production code is developed through the red/green/refactor form and then added to the core assets repository.

When new requirements are variations of features previously added to the SPL, unit tests and code changes are applied to allow activating this new variability. Testing helps with making sure that the first application continues to behave properly. The new unit tests will drive the changes in the source code for the integration of variability points, which are then configured in the second application.

The main mechanism to activate the variability is the configuration file. In addition to centralize the settings for building applications, the configuration file also documents the variations of each application. The file format chosen was the JavaScript Object Notation (JSON) [7], because it is a lightweight file format, easy to read and is commonly used to data-interchange.

The configuration file is created when features are added as variabilities to the features that already exist in the core assets repository. This file has a key and a value to configure each feature. Along with this file a test class is created for testing the possible combinations of keys and values. It is also necessary to create a class in the code to read the configuration file and to instantiate a configurator object, which will be used to inform the application of what variability will be activated. The configuration file is modified as more variabilities are added to the SPL. Each application has its own configuration file, activating and configuring variabilities as needed.

The proposed process can be described as follows:

1. The first application is developed using TDD as if it were the only one, without attempting to predict future features and variations.
2. Tests related to the first application become part of the core asset artifacts.
3. The features of the second application are gathered and described by user stories, which is a usual technique to define requirements in agile methods.
4. The features of the second application that are not related to existing features are developed with TDD.
5. The features of the second application that are variations of existing features follow a slightly different path:
  - 5.1. Identify where the existing feature is tested.
  - 5.2. Add new(s) test(s) to include the new expected behavior and refactor the existing code to pass these tests.
  - 5.3. Refactor the existing code to use the configurator object that is modified.
  - 5.4. Add new tests to verify the behavior of the new entries in the configurator object.

### IV. EXAMPLE

To illustrate the use of our proposed process, the application developed in the book *Test-Driven iOS Development* [8] was chosen as the first application of a reactive SPL. The choice of an existing application, which was test-driven developed, has been made to reduce the bias of

writing the first application. Moreover, this application was not developed following the SPL approach. This first application serves as a basis for the development of the second application, where new features and variabilities are inserted. The tests of the first application are refactored so that the core assets repository continues to support both applications.

In the following, we initially describe the first application. Then we describe the second application and the modifications that were made to the core assets repository.

### 1.1 First Application

The first application is an application for the iOS operating system that provides to the user the latest questions of Stack Overflow<sup>1</sup> forum related to mobile development for iOS. The questions are organized by topics, and users can also view the answers that were given to these questions. The name of this application is BrowseOverflow.

Describing such application in more detail, we obtain the following user stories:

- **List of topics.** The application starts showing a list of topics related to iOS. Each topic is associated with a tag from Stack Overflow.
- **Access the latest questions.** When a topic is selected, another list appears showing the 20 latest questions that have a tag associated with the selected topic. This list is sorted in chronological order, from most recent to least recent. Each item of the list shows the title of the question, the user who made the question (name and image), and the score of the question (number of people who upvoted or downvoted the question).
- **Connection availability.** To load the list of recent questions, an internet connection is necessary, but it is possible that the connection to the stackoverflow.com site fails. In the latter case, the application must then inform the user that the information cannot be loaded, either for lacking an internet connection or communication failure with the site.
- **Get answers to one question.** When selecting a question, the application opens a screen showing the question description and a list of answers. If an answer was chosen as correct, it appears first. Otherwise, if no answer is chosen, the answers are listed according to the score they have. For each answer the name and avatar of the user who answered it are also shown.

The user stories were transcribed to a feature model that shows which features were developed. The resulting feature model is shown in Figure 1. As the features come from the first application, all of them are mandatory.

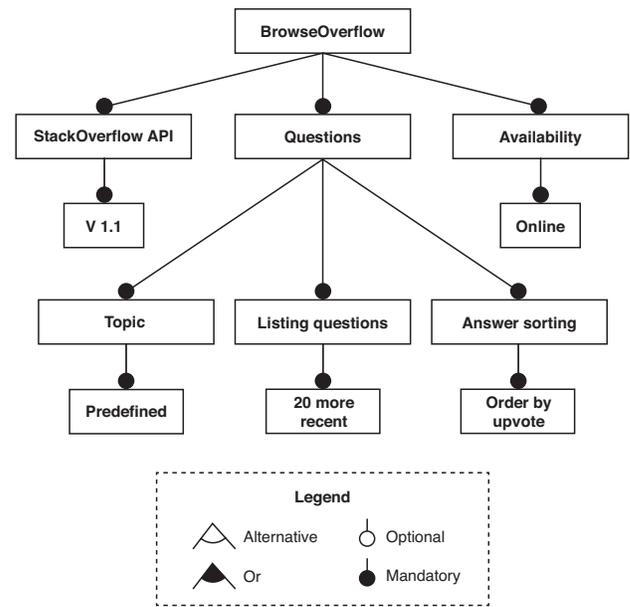


Figure 1. Feature model of the first application

Besides the development of the first application has been test-driven, it was done incrementally following a layered architecture. The model layer was developed first, followed by the controller layer. Finally the integration between both layers was done. The source of the first application is open<sup>2</sup> and has 24 classes and 182 unit tests.

### 1.2 Second Application

In the second application some new user stories were added, some user stories from the first application were modified and some user stories from the first application were maintained.

The following user stories are either new ones or variations of the first application stories:

- **Access the 30 most recent questions.** When a topic is selected, the list appears showing the 30 most recent questions.
- **Using the API 2.0.** The API version used to request the questions and answers should be 2.0 instead of 1.1.
- **Access content without internet connection.** The content (questions and answers) that was displayed previously must be available to the user even if there is no internet connection.
- **Insert new topics.** On the topic screen, the user can add new topics in order to look for questions.
- **Order the answers chronologically.** A list of answers to a question must be chronologically ordered, showing the most recent answers first.

To exemplify the development of the SPL and its unit tests, the development of the first three user stories listed above is discussed next.

<sup>1</sup> Available at <http://stackoverflow.com>

<sup>2</sup> Available at <https://github.com/iamleeg/BrowseOverflow>

### 1.2.1 Access the 30 Most Recent Questions

The steps required to develop the user story that changes the amount of questions from 20 to 30 are detailed as follows.

First of all, we need to find where this feature is tested inside the unit tests of the first application. In the proposed model for the first application, each topic contains between 0 and 20 questions. The class diagram corresponding to this user story, presented in [8], is shown in Figure 2.



Figure 2. Relationship between Topic and Question in the first application

In the class responsible for testing topics (*TopicTests*) is the method *testLimitOfTwentyQuestions* (Figure 3). This method ensures that the modifications made to accept the new value will not affect the behavior of first application.

```

1. - (void)testLimitOfTwentyQuestions
2. {
3.     Question *q1 = [[Question alloc] init];
4.     for (NSInteger i = 0; i < 25; i++) {
5.         [topic addQuestion: q1];
6.     }
7.     XCTAssertTrue(
8.         [[topic recentQuestions] count] < 21);
9. }
  
```

Figure 3. First application test limit of twenty questions method

Since the first application had a limit of 20 questions and the second application extends this limit to 30 questions, this feature has to be modified. We can generalize the limit of questions to *n*, where *n* is defined in the configuration file responsible for building each application.

Thus, a new method, called *testLimitOfQuestions* (Figure 4), is created to test the limit of questions that must be defined at compile time. A new field is created in *Topic* class to store this value and the signature of the constructor is refactored to accept a new limit parameter (Figure 5). These changes resulted in changes elsewhere. The method *setUp* in the *TopicTests* class needed to be refactored to use the new constructor. The method *addQuestions*, that previously stored the number 20 directly in code, now uses the limit defined for the application (Figure 6). After having passed all tests, we found out that the test *testLimitOfTwentyQuestions* was no longer necessary, and then it was removed.

```

1. - (void)testLimitOfQuestions
2. {
3.     Question *q1 = [[Question alloc] init];
4.     for (NSInteger i = 0; i < topic.limit; i++) {
5.         [topic addQuestion: q1];
6.     }
7.     XCTAssertTrue(
8.         [[topic recentQuestions] count] < topic.limit);
9. }
  
```

Figure 4. Added method testLimitOfQuestions

```

1. @synthesize limit;
2.
3. - (id)initWithName:(NSString *)newName
4.     tag:(NSString *)newTag
5.     limit:(int)newLimit
6. {
7.     if ((self = [super init])) {
8.         name = [newName copy];
9.         tag = [newTag copy];
10.        limit = newLimit;
11.        questions = [[NSArray alloc] init];
12.    }
13.    return self;
14. }
  
```

Figure 5. New class field and change in the constructor signature

```

1. - (void)addQuestion:(Question *)question
2. {
3.     NSArray *newQuestions =
4.         [questions arrayByAddingObject: question];
5.     if ([newQuestions count] > 20) {
6.         newQuestions =
7.             [self sortQuestionsLatestFirst:
8.                 newQuestions];
9.         newQuestions =
10.            [newQuestions subarrayWithRange:
11.                (NSMakeRange(0, 20))];
12.    }
13.    questions = newQuestions;
14. }
  
```

```

1. - (void)addQuestion:(Question *)question
2. {
3.     NSArray *newQuestions =
4.         [questions arrayByAddingObject: question];
5.     if ([newQuestions count] > limit) {
6.         newQuestions =
7.             [self sortQuestionsLatestFirst:
8.                 newQuestions];
9.         newQuestions =
10.            [newQuestions subarrayWithRange:
11.                (NSMakeRange(0, limit))];
12.    }
13.    questions = newQuestions;
14. }
  
```

Figure 6. Changes in the method addQuestions (before and after)

Since this was the first feature modified in the SPL features model, it was necessary to create the test class responsible for testing the class that will read the configuration file and instantiate the configurator object. The configurator object is accessed by the system to enable the variability points.

The necessary information to instantiate the configurator object comes from the configuration file, a JSON file that is created for each application. A JSON file to configure the number of question is shown in Figure 7. For each variability point of the SPL there is a descriptive key in the configuration file and that key references a value that has the settings required to activate that point.

```

1. {
2.   "limitOfQuestions": 30,
3. }

```

Figure 7. Example of JSON configuration file

### 1.2.2 Using the API 2.0

In the case of the API variation, the modification is more complex. In the first application, a facade [9] called *StackOverflowManager* is responsible for the communication with the class *StackOverflowCommunicator* (Figure 8). The requests to the API 1.1 are centralized in this communicator class.

Now we need a class that tests the communicator that will be used independently of the API version. The tests guide the construction of an adapter [9] *StackOverflowCommunicator*. This adapter is responsible for calling the communicators with the API 1.1 (*StackOverflowCommunicatorV11*, the old *StackOverflowCommunicator* class of the first application) and with the API 2.0 (*StackOverflowCommunicatorV20*) (Figure 9). This technique of extracting an adapter during refactoring was proposed by Kerievsky in [10].

However, other problems arise from the API change because the JSON format of the response from each API is different. Despite the *PersonBuilder*, *QuestionBuilder* and *AnswerBuilder* classes not knowing about the requests, they assume they will receive a JSON file with the same predefined fields. So, in this case the builder classes must receive the JSON file and also the mapping between this file and the object to be instantiated.

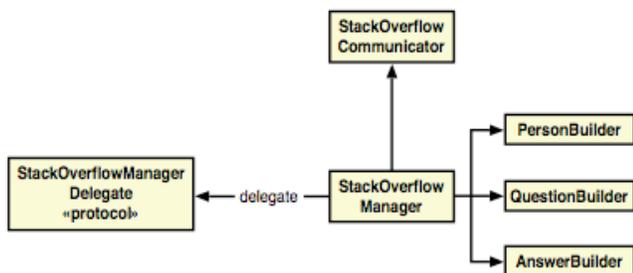


Figure 8. Relationship between the facade and other classes [8]

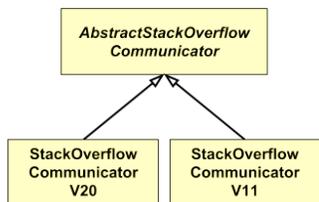


Figure 9. StackOverflowCommunicator new adapters

### 1.2.3 Access Content Without Internet Connection

The availability of the content even without internet connection leads to changes in many places, since the first application always notifies the user that a problem has occurred when there is no internet connection.

The *StackOverflowManager* class also acts as a facade between the communicator and the builder [9] classes

(*PersonBuilder*, *QuestionBuilder* and *AnswerBuilder*). The facade asks for the communicator to make a request to retrieve a JSON response from the *Stack Overflow*. When successful, the manager passes the JSON response to the builder classes. At this point, the JSON response must be saved by the application. When the request fails, the *StackOverflowManager* should verify if the JSON response was saved in memory before.

After applying all the changes in the code, the configuration files for each application are defined as shown in Figures 10 and 11.

```

1. {
2.   "limitOfQuestions": 20,
3.   "stackOverflowApiVersion": 1.1,
4.   "userDefinedTopic": false,
5.   "contentAvailability": "online",
6.   "answerSorting": "upvote"
7. }

```

Figure 10. Configuration file of the first application

```

1. {
2.   "limitOfQuestions": 30,
3.   "stackOverflowApiVersion": 2,
4.   "userDefinedTopic": true,
5.   "contentAvailability": "offline",
6.   "answerSorting": "date"
7. }

```

Figure 11. Configuration file of the second application

As a JSON file does not support comments, we need other support file to document the possible values for each key.

At the end of the development of the second application, the feature model evolves including features that are mandatory, optional or mutually exclusive alternatives, as shown in Figure 12. It is important to point out that, in the case of the feature related to the quantity of questions we now have a more generic feature, and in the case of the features related to the API version and content availability we now have alternatives.

## V. RELATED WORK

Ghanam [11] proposed the use of a test-driven approach to introduce variability through the refactoring of the existing code. Tests can guide the insertion of new forms of system variability as required. He proposed the use of the factory pattern [9] in the refactoring process in order to generate new feature alternatives and the use of the decorator pattern [9] to generate options. Then the variabilities are configured in a specific class of the code.

Our work differs from Ghanam's work [11] because we explore more design patterns to generate variability. There are cases where the factory and decorator patterns are not enough or they are not the best option to insert variability. Furthermore, the solution presented in [11] is platform-dependent, whereas the configuration file of our work was designed to be used in other platforms too.

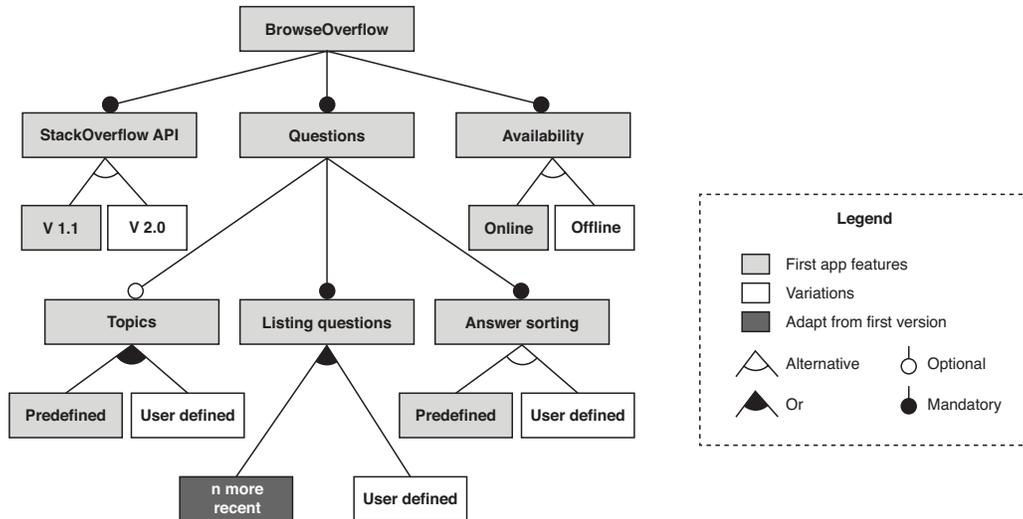


Figure 12. Feature model of the second application developed

Kakarontzas [12] proposed a systematic approach to create new quality and functional variant components through the customization of existing core assets of a SPL. This work also uses the TDD to assist with the evolution of SPL software components. It suggests the creation of a new component, which is an extension of a pure component, when new features are added. Thus, when a new functionality is needed, just a pure component or a component in the higher hierarchy has to be used.

The component hierarchy presented in [12] increases the complexity. With a large hierarchy, it is more difficult to select alternative and requirements options to compose a new product. The configuration file we use in our work seems to be a simpler solution to centralize the variabilities choice as well as to serve as a guide for building a new application.

## VI. CONCLUSIONS

Traditional SPLs have the disadvantages of high initial investment, the development of artifacts that may not be used, and the difficulty of dealing with unknown segments. The APLE arises to address these disadvantages using agile practices.

This paper showed how TDD and refactoring agile practices could make a reactive SPL evolve and acquire variability points on demand. This is done through a defined process and the use of the configuration file as the variability mechanism.

In our proposed process the first step is the development of an application using TDD, without considering the creation of other applications belonging to the same SPL. Then new features and variations of existing features are added in the form of user stories. To develop the features we create new variability points from new tests, and we also create the configuration file to activate the desired variability of each application. Then we implement the code necessary to pass all tests. The configuration file allows visualizing the application variability points in an easy way and centralizing the variability

mechanisms in a unique place. It can also be used as a guide for the construction of a new SPL application.

As future work we intend to apply the proposed process practices in a company of the mobile applications segment, addressing the cross platform development challenges. We will develop one iOS application using the Objective-C language and another Android application using the Java language so that the variabilities are specified in the configuration file.

## REFERENCES

- [1] Clements, P., Northrop, L. *A Framework for Software Product Line Practice, Version 5.0*. [http://www.sei.cmu.edu/productlines/frame\\_report/index.html](http://www.sei.cmu.edu/productlines/frame_report/index.html), Pittsburgh, PA, USA: Software Engineering Institute, 2012.
- [2] Dfáz, J., Perez, J., Alarcón, P. P., Garbajosa, J. *Agile Product Line Engineering - A Systematic Literature Review*. Software, Practice & Experience (Print), v. 41, p. 921-941, 2011.
- [3] Constantinou, A., Camilleri, E., Kapetanakis, M. *Developer Economics 2012: The new mobile app economy*. London: Visionmobile, 2012.
- [4] Milano, D. T. *Android Application Testing Guide*. Birmingham: Packt Publishing, 2011.
- [5] Silva, I. F., Neto, P. A. M. S., O'Leary, P., Almeida, E. S., Meira, S. R. L. *Agile Software Product Lines: A Systematic Mapping Study*. Software, Practice & Experience (Print), v. 41, p. 899-920, 2011.
- [6] Beck, K. *Test-Driven Development By Example*. 2002.
- [7] JSON. *Introducing JSON*. <http://www.json.org/>. 2013.
- [8] Lee, G. *Test-Driven iOS Development*. Crawfordsville: Addison-Wesley, 2012.
- [9] Gamma E., Helm R., Johnson R., Vlissides J. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 1995.
- [10] Kerievsky, J. *Refactoring to Patterns*. Crawfordsville: Addison-Wesley Professional; 1 edition, 2004.
- [11] Ghanam, Y. *An Agile Framework for Variability Management in Software Product Line Engineering*. Doctor Thesis. Calgary, Alberta, Canada: University of Calgary, 2012.
- [12] Kakarontzas, G., Stamelos, I., Katsaros, P. *Product line variability with elastic components and test-driven development*. CIMCA '08: Proceedings of the 2008 International Conference on Computational Intelligence for Modeling Control and Automation. IEEE Computer Society: Washington, DC, U.S.A., 2008: 146-15.

# A Controlled Experiment to Explore Potentially Undetectable Defects for Testing Techniques

Martín Solari, Santiago Matalonga  
Universidad ORT Uruguay  
martin.solari@ort.edu.uy, smatalonga@uni.ort.edu.uy

**Abstract**— Software testing practitioners have an array of testing techniques to choose from to test their software. Nevertheless, there is little empirical evidence about the capability of each technique to detect specific types of defects. As a result, when selecting and combining the testing techniques for a project, practitioners must rely on their own experience. This paper studies the behaviour of two specific techniques, equivalence partitioning and decision coverage, to determine which types of defect are potentially undetectable to either one. This paper presents a differentiated experiment replication based on a previous experimental design, but using different artifacts. The experiment confirms the hypothesis that some defect types are undetectable to each technique. Even with a correct application of each technique, some defects will only be detected by chance. This study adds new empirical evidence for constructing a classification of defects that takes into account technique detection capabilities.

**Keywords:** *Software testing, experiment, defect detection.*

## I. INTRODUCTION

Software testing is one of the software quality assurance's key activities. Nowadays, the software testing practitioner has a myriad of tools and techniques at his disposal to carry out his task effectively. And yet, our experimental knowledge on these tools and techniques is limited [1].

An important contribution to the experimental knowledge on defect detection techniques has been achieved by the execution of a family of experiments started by Basili [2]. This family of experiments has been adapted and replicated by other researchers [3]–[6]. The objective of this family of experiments is to study the impact of the human factor on the application of different defect detection techniques. In the different iterations of the experiment, several manual static (inspection) and dynamic (testing) defect detection techniques were compared. The most common result is that these techniques have similar defect detecting capability, and that the efficacy is dependent of the defect and the context where the technique is applied [6].

This paper presents an experimental replication that uses a set of seeded defects to compare the efficacy of functional and structural manual testing. These seeded defects were designed to test the hypothesis that certain defects could be potentially undetectable for the functional or structural testing technique.

The differentiated replication presented in this paper is based on an previous experiment designed by Juristo and Vegas [5]. We call this replication differentiated because the experiment was modified in two key aspects. First,

experimental programs were adapted to fit the knowledge and training of the experimental subjects. Secondly, seeded defects were modified to test a new hypothesis that defects detected by each technique are different.

The results presented by this replication confirm previous replications of this family of experiments regarding technique efficacy, but also adds new data about the types of defects detected by each technique. Our results show that some defects types are potentially undetectable to structural or functional techniques. Furthermore, the techniques efficacy to detect a defect is dependent to the program and the specific seeded defect.

This paper uses Carver's guidelines to report replications of software engineering controlled experiments [7]. These guidelines suggest describing in separate sections the previous experiment and the replication, and also including an interpretation of results in the context of the experimental family. Following this, the rest of the paper is structured as follows. Section II details the history of the family of experiments that this replication is part of. Section III presents the information about the previous experiment. Section IV describes this experimental replication. Section V discusses the potentially undetectable defects for each technique. Section 0 presents a comparative analysis of the replication with the previous results. Section VII presents the interpretation of the results in the context of the experimental family and finally in Section VIII we present our conclusions.

## II. HISTORY OF THE FAMILY OF EXPERIMENTS

Basili started the family of experiments to which this replication is part of in the 1980s. Since then, this experiment has been extensively evolved and replicated [3]–[6]. Some of these replications have introduced changes in the techniques under study or have adapted the materials to suit different contexts. Some replications have focused on code reviews, and other in testing techniques. Some have even considered comparing these approaches.

We consider the first stage of the experiment to be the replications carried out at Maryland University and the Software Engineering Laboratory at NASA Goddard Centre [2]. The second stage of this family of experiments is based on the laboratory package that was built by Kamsties y Lott at Kaiserslautern University [3], and replicated by Roper et al [4]. Juristo and Vegas carried out the following stage at Universidad Politécnica de Madrid (UPM) [5]. This stage produced the laboratory package that has driven the replication

presented in this paper. Arguably, this family of experiments is one of the longest running families of experiments within the experimental software engineering community.

In addition to providing another replication to this family of experiments, the motivation for this work is to identify new empirical evidence about which types of defects could be potentially undetectable for structural or functional testing techniques. It has been hypothesized that functional and structural techniques complement each other, but there is no empirical evidence to confirm this hypothesis. This replication uses defects that have been seeded in order to evaluate if they are undetectable to either technique. We define a defect to be potentially undetectable to a technique, if the technique capability to find it can only be attributed to chance, and not to the technique prescription for test case generation.

### III. INFORMATION ABOUT THE PREVIOUS EXPERIMENT

The replication reported in this paper is based on a previous experiment design proposed by Juristo and Vegas [6] performed at UPM. This experiment is in turn a replication based in the laboratory package created by Kamsties and Lott [3], later refined in multiple replications [4], [5]. When referring to the UPM experiment here, we use the term *previous experiment* in order to avoid confusion with other experiments in the same family.

One of the main evolutionary changes in the Juristo and Vegas replication was the separation of test design and test execution activities. Each subject performed two separate exercises: one for designing the test cases and other to execute the test cases against the executable program. In both activities, the influence of the human factor can be observed. This reductionist approach has enabled a more focused study of the phenomenon of applying a dynamic software evaluation technique.

Along the replication process, with the objective of studying the behaviour of the techniques against different types of defects, several defect taxonomies have been used. One of the replication used the same defect classification introduced in the Basili experiment. Different types of defects were seeded generating multiple versions of the programs. This design allowed the exploration of defect type as a factor in a significant number of cases. However, this modification did not yield to conclusive results and the defects were re-seeded following a different scheme.

The research question of the previous experiment is: ¿Which is the efficacy of different software evaluation techniques (equivalence partitioning, decision coverage and code review by stepwise abstraction) for detecting defects? This question could be expressed in a null hypothesis format:  $H_0$ : *There is no difference in defect detection efficacy between equivalence partitioning, decision coverage and code review by stepwise abstraction.*

For the dynamic evaluation techniques (equivalence partitioning and decision coverage) the response variable is measured as the percentage of subjects that generate at least one test case that can detect the failure associated with each fault. For the static evaluation technique (code review by

stepwise abstraction) efficacy is measured as the percentage of subjects that report each fault. The previous experiment uses a factorial design. The factor of the experiment (independent variable) is the evaluation technique with three alternatives: equivalence partitioning, decision coverage and code review by stepwise abstraction.

Subjects of the previous experiment were fourth-year students of the course Software Evaluation at UPM. The students profile was medium level of experience in programming. The training on the testing techniques was received during the course, so the testing experience level was low. At an operational level, the experiment was performed in Spanish language. In the previous experiment all the artefacts provided to the subject to accomplish the task are paper form. The programs are coded in C language and have between 200 and 300 lines of code. Three different programs are used: *cmdline*, *nametbl* and *nree*. Each program has 6 seeded faults, named F1 to F6. Faults F1, F2 and F3 are seed with the intention to be more difficult to find for the structural technique. Faults F4, F5 and F6 are seeded with the intention of being more difficult to find for the functional technique.

The results of the previous experiment confirm (at a statistically significant level) the hypothesis that some defects are much more difficult to detect for structural or functional testing techniques. The code review technique has an overall lower efficacy, but could potentially detect all the seeded defects.

### IV. DESCRIPTION OF OUR REPLICATION

Performing the replication had a double motivation: start an experimental research project about software testing and studying the communications mechanisms in experimental replications [9]. Our replication was performed at Universidad ORT Uruguay from 2010 to 2012. The experimental design was derived from the previous experiment, but changing the number of alternatives in the independent variable (the code review technique was not studied). New artifacts were created in Java to match subjects skills. More importantly, new defect types were seeded to test a new research hypothesis.

The experiment subjects were final year undergraduate students taking a Software Testing course. The experiment was run with three different groups, one in each year using the same experimental design and operation. The number of subjects was 8 in 2010, 9 in 2011 and 7 in 2012. All the cases were jointly analyzed with a procedure already used for obtaining bigger statistical samples in software engineering experiments [8].

To the previous experiment hypothesis about technique efficacy, we added a new null hypothesis in our replication:  $H_{0Rep}$ : *All defects are potentially detectable for equivalence partitioning and decision coverage testing techniques.*

Although, the main experimental design remains the same, the code review technique was not studied in this replication. This change has no effect in the possibility of aggregation and comparison of results with previous replications, considering only the shorter list of studied alternatives.

The subjects' background in programming is different from the previous experiment subjects. To fit this constraint, new programs in Java language were used in this replication. The two generated programs *Export* and *MesConv* are of similar size, each one having a 2 page specification and around 200 lines of source code.

Another change of the replication was in the data gathering mechanism. In the previous experiment paper forms were used, while in the replication the subjects had to upload a file with the test cases in an authenticated web system. Each subject could use JUnit as a framework to create the test cases (either functional or structural) but this step was not mandatory.

## V. DISCUSSION OF POTENTIALLY UNDETECTABLE DEFECTS

In order to present the results of the replication, a discussion regarding the potentially undetectable defects for each technique is necessary. Although there are several defect classification taxonomies, to the best of our knowledge, none of them has the capability to differentiate technique defect detection capability. Because of this, it is necessary to have a preliminary theoretical discussion about defects types potentially undetectable to structural or functional testing techniques.

Structural and functional techniques use different strategies for test case generation. In the case of equivalence partitioning the program specification is used as the main source, analyzing the semantic and dividing the input space it in different equivalence classes. This technique uses heuristics to identify the classes and generate associated test cases. In the case of the structural technique the program source code is normally the main source for generating the test cases. The code is analyzed from a control flow or data flow perspective to generate test cases that cover certain degree of these structures. For example, in decision coverage each branch of execution is tested by at least one test case. This strategies use different foundations, so the generated test cases could lead to some potentially undetectable types of defect for each technique.

Functional techniques could have problems detecting:

- Defects related to **combination of equivalence classes**. The technique does not prescribe in which way the valid classes should be combined. If the defect is only detectable if certain classes are combined in the same test case, this type of defect could be potentially undetectable.
- Defects present within **code of unspecified functions**. Since the specification is the basis for the equivalence partitioning, if there is a defect in code not associated with the specification, this type of defects are potentially undetectable for the functional technique.
- Faults associated with **implementation structures**. Is possible that the developer makes decisions in the implementation structures, not necessary mandated in the specification but following implementation constraints or developer preferences.
- Faults detectable only if **specific data set** are used in the test case. Equivalence partitioning technique

mandates to use at least one data input representative of each class in the test cases.

Structural technique could have problems detecting other types of defects:

- Faults of **complete omission of functionalities in implementation**. Since there are no internal structures to generate the test cases, the omission could only be detected if by chance the generated test cases cover the omitted function.
- Faults associated with **specific combination of predicates or test data**, not necessarily exercised by executing decision branches. This is an intrinsic limitation of the decision coverage criterion.

In our replication, 4 defects of similar type were seeded in each program. Defects e1 to e4 were seeded in the *Export* program, while defects m1 to m4 were seeded in the *MesConv* program. Based on this classification, we hypothesise that the first two faults are potentially undetectable for the structural technique and the last two potentially undetectable for the functional technique. We try to seed faults of the same type and approximately equal in detection difficulty, although this assumption should be confirmed with the collected empirical data.

## VI. ANALYSIS OF RESULTS

In a preliminary analysis a set of descriptive statistics were used to confirm the potentially invisible defects for the structural and functional techniques. The response variable is the percentage of subjects that detect each defect. Table II shows the detection rate and the classification of each defect after the preliminary analysis. Values below 50% detection rate are written in bold.

The number of participants was limited to 24 subjects, each performing two exercises. Since the sample size is near the conventional statistical practice, both parametric (ANOVA: Analysis of Variance) and non-parametric (Mann-Whitney) statistical tests were used. Both tests showed significance in the same independent variables (technique and program).

TABLE I. DETECTION RATE FOR EACH FAULT AND TECHNIQUE

Fault	Structural det. rate	Functional det. rate	Preliminary analysis
e1	<b>0,33</b>	0,92	Potentially undetect. to structural
e2	1	1	Highly detectable
e3	0,5	0,92	Potentially undetect. to structural
e4	<b>0,33</b>	0,83	Potentially undetect. to structural
m1	<b>0,25</b>	<b>0,42</b>	Potentially undetect. for both
m2	0,92	0,92	Highly detectable
m3	<b>0,08</b>	0,67	Potentially undetect. to structural
m4	<b>0,17</b>	<b>0,17</b>	Potentially undetect. for both

Regarding the general efficacy of structural and functional techniques, efficacy of the functional technique is significantly better than the structural technique in this replication. After performing the preliminary analysis, this result can be attributed to the fact that only potentially undetectable defects for the structural technique were actually seeded. Besides the

technique factor, the statistical analysis also shows that the program is a significant factor that affect efficacy. This is explained because the *MesConv* program has specific defects that are more difficult to detect than the *Export* program counterpart. This is a confirmation that the detection capability of each defect is more dependent on the specific seeded defect and program, than the intended preliminary classification.

## VII. INTERPRETATION OF RESULTS WITHIN THE FAMILY OF EXPERIMENTS

There are a considerable number of experiments and replications studying software evaluation techniques, but results are not always consistent. Variations among replications on technique and training factors are not conclusive about which technique to use. Because of this, the safest advice for a practitioner is to use a combination of techniques. The replication reported in this paper focused the investigation in which types of defects could be potentially undetectable to each testing technique.

This family of experiments has also contributed with observations about the human factor in the application of software evaluation techniques. The potential theoretical efficacy of certain technique could be considerably different with the result when applied by practitioners. With the same training, different subjects generate different set of test cases, even for well-established testing techniques. Because of this variation is sensible to promote that the evaluation of critical systems is done involving several persons to archive the maximum potential defect detection capability of the technique.

The experiments in the family show that technique efficacy is dependent of the program and the seeded defect. However, we do not have yet a complete and practical defect classification that is sensitive to technique defect detection capability.

## VIII. CONCLUSIONS

This paper has presented a replication of an experiment about software testing techniques that belongs to one the longest running families of experiments in the experimental software engineering community. The results from the replication show that the efficacy of the structural technique (decision coverage) and functional techniques (equivalence partitioning) are similar, although the last set of seeded defect yield to greater efficacy of the functional technique. The observed difference can only be statistically attributed to the natural variation of the subjects involved in the application of the techniques. This result confirms previous observations of other replications of this family of experiments. Nonetheless, this replication adds a new element to the existing empirical knowledge. Our result adds empirical evidence that there are certain types of defects that can be potentially undetectable to certain testing techniques. This means that even with the correct application of the technique, the defect can only be detected by chance when a test case is defined that specifically stimulates the conditions for that defect.

According to the current level of empirical knowledge in the field, exploring the hypothesis that different techniques can detect different types of defects is still relevant. These defect types must be mapped to real defects in production software, which will allow for the development of another classification to continue this line of research. Furthermore, the impact of the human factor in the application of each technique still requires more research. For instance: How many testers are necessary for applying a technique to reach its theoretical defect detection limit? How can testing techniques complement each other according to their patterns of potentially undetectable defects? Only by continuous experimentation and replication iterations, we will be able to produce enough variations to provide practitioners with empirical knowledge to answer these questions.

## IX. REFERENCES

- [1] N. Juristo, A. M. Moreno, S. Vegas, and M. Solari, "In Search of What We Experimentally Know about Unit Testing," *IEEE Softw.*, vol. 23, no. 6, pp. 72–80, Nov. 2006.
- [2] V. R. Basili and R. W. Selby, "Comparing the Effectiveness of Software Testing Strategies," *IEEE Trans. Softw. Eng.*, vol. SE-13, no. 12, pp. 1278–1296, 1987.
- [3] E. Kamsties and C. M. Lott, "An Empirical Evaluation of Three Defect-Detection Techniques," *European Software Engineering Conference*. 1995.
- [4] M. Roper, M. Wood, and J. Miller, "An empirical evaluation of defect detection techniques," *Inf. Softw. Technol.*, vol. 39, pp. 763–775, 1997.
- [5] N. Juristo and S. Vegas, "Functional Testing, Structural Testing, and Code Reading: What Fault Type Do They Each Detect?," in *Empirical Methods and Studies in Software Engineering - Experiences from ESERNET*, A. I. Wang, Ed. Springer, 2003, pp. 208–232.
- [6] N. Juristo, S. Vegas, M. Solari, S. Abrahao, and I. Ramos, "Comparing the Effectiveness of Equivalence Partitioning, Branch Testing and Code Reading by Stepwise Abstraction Applied by Subjects," in *International Conference on Software Testing, Verification and Validation (ICST 2012)*, 2012.
- [7] J. C. Carver, "Towards Reporting Guidelines for Experimental Replications: A Proposal," in *1st International Workshop on Replication in Empirical Software Engineering Research (RESER) [Held during ICSE 2010]*, 2010.
- [8] F. Shull, J. C. Carver, S. Vegas, and N. Juristo, "The role of replications in Empirical Software Engineering," *Empir. Softw. Eng.*, vol. 13, no. 2, pp. 211–218, 2008.
- [9] N. Juristo, S. Vegas, M. Solari, S. Abrahao, and I. Ramos, "A Process for Managing Interaction between Experimenters to Get Useful Similar Replications," *Inf. Softw. Technol.*, vol. 55, no. 2, pp. 215–225, 2013.

# Test Data Generation for Web Applications: A Constraint and Knowledge-based Approach

Hibiki Saito\*, Shingo Takada  
Dept. of Information and Computer Science  
Keio University  
Yokohama, Japan

Haruto Tanno, Morihide Oinuma  
Software Innovation Center  
NTT Corporation  
Tokyo, Japan

**Abstract**—Software testing is an important part of the software development process. Much work has been done on automating various parts of testing. In previous work, we had proposed a knowledge-based approach to generate test scenarios for Web applications. However, our previous work did not account for generation of actual test data. Thus, in order to execute the test scenarios, the user would need to (manually) create the test data. This paper proposes an approach to generate test data for our previously proposed test scenario generation tool. Our approach can generate two types of test data: constraint-based test data and database-based test data. Our tool can now automatically execute the combined test scenario and test data. We confirmed the usefulness of our approach through a case study.

*Keywords*—test data generation; Web applications; test scenario

## I. INTRODUCTION

Software testing is an important part of the software development process, but it can be time-consuming and very costly. Thus much work has been done on automating various aspects of testing, such as test case generation and test case execution.

In previous work, we had proposed a knowledge-based approach to generating test scenarios for Web applications [1]. A test scenario represents a sequence of steps that are taken to transition from one Web page to another. We generated test scenarios by using test scenario information from previous Web applications that were stored in a knowledge base. Although we had shown that our approach was able to generate many useful test scenarios that professionals were not able to create manually, there was one important drawback. We could generate test scenarios, but we could not generate the test data themselves. A tester would need to manually generate the test data based on the test scenarios to actually conduct the test.

We thus propose an approach to generate test data for our previously proposed test scenario generation tool. The main contributions of this paper are as follows:

1. Extension of our previously proposed tool to generate test data.
2. Generation of test data using constraints and knowledge-base.
3. Evaluation through a case study.

The rest of this paper first starts with a discussion of related work. Section III then describes our approach. Section IV evaluates our approach. Section V makes concluding remarks.

## II. RELATED WORK

Much work on test data generation has been done [2]-[9]. Pacheco, et al proposed feedback-directed random testing, and implemented it as Randoop [2]. Basically, test data is generated randomly, but the generation process is guided by the execution of previous inputs.

Another approach is based on symbolic testing, which was originally proposed by King [3]. In symbolic testing, a program is “executed” using symbols rather than concrete values for each variable. These symbols are collected to form expressions which can be solved to obtain concrete values, i.e., test data, each of which will result in the execution of a certain program path. Concolic testing addressed the issue of redundant execution in symbolic testing by combining concrete execution with symbolic execution [4].

Search-based software testing [5] generates test data by using optimization algorithms. A fitness function is defined to guide the search of a space of test data to find “good” data. Gross, et al [6] proposed an approach that generates test cases at the GUI level using genetic algorithm. Mariani, et al [7] proposed a tool called AutoBlackTest which uses Q-Learning to guide the test process. Alshahwan, et al [8] applied search-based software testing to Web application, but their approach is limited to PHP applications.

Finally, model-based testing generates test data using models and/or formal specifications. Fujiwara, et al [9] proposed a formal approach based on UML class diagrams and OCL constraints. It was shown to be a powerful method, but it requires a tester/modeler to be well-trained in formal modeling.

Work has been done on test data generation for Web applications. But they have issues such as being applicable only to PHP applications [8] or needing information that requires someone that is well trained [9]. Such approaches cannot be used for our test scenario generation tool. Thus, we propose our approach to work on top of our previous test scenario generation tool, which can be used for any Web applications since it does not require a certain language such as PHP, and it does not require the tester to be well trained in a certain technology.

\* Currently at Hitachi, Ltd.

### III. TEST DATA GENERATION

#### A. Previous Work

Our test data generation is based on our previous work on test scenario generation. We thus first give an overview of our previous work.

The tester first creates a base scenario of the Web application under test. This base scenario corresponds to the basic normal steps that a user takes for the current Web page. For example, a base scenario for logging in may include steps such as “input user id”, “input password” and “submit information” (Fig. 1 (a)).

Our tool then searches a database containing information on each step (called “scenario node”). The goal of this search is to find nodes that were used in previous tests that can replace nodes in the current base scenario. A dictionary is also employed to account for synonyms, e.g., “user name” and “user id” can be considered as synonyms. Our tool generates related test scenarios, each differing with the base scenario by one node which had been replaced. Thus for example, given the base scenario in Fig. 1 (a), a related scenario (shown in Fig. 1 (b)) may be generated where the first node is replaced.

#### B. Overview of our proposed approach

Figure 2 shows the overview of our proposed approach. The initial step of constructing a base scenario with our Scenario Editing Tool is the same as our previous work, except the tester will need to specify constraints and some other extra information, specifically Input Type and Concrete Value (subsection III-C), for test data generation and execution.

Our Scenario Analyzer analyzes the base scenario and searches for nodes in the scenario node database to use for replacement. The dictionary is used to check for synonyms. The Test Case Generator generates the related test scenarios (as in our previous work) and also test data. We consider a test case to be a combination of a test scenario and test data.

Our Test Case Generator generates two types of test data. The first type is constraint-based test data, which is generated by encoding constraints for each node. For example, if a node is concerned with inputting numbers within a textfield, then we encode that information within that node. When generating test data, we can then (randomly) generate a string that includes a non-number for that textfield.

The second type is database-based test data. This type of test data is reuse of data that was used in a previous test case. For example, if a node contains information that “johnny” was used in a previous test case, then that can be used.

Finally, our Test Case Executor automatically executes the set of ranked test cases. Test cases are ranked based on frequency, i.e., if nodes have been used often in previous tests, then those test cases have a higher ranking.

#### C. Scenario Node

Table I shows information that is contained within the scenario node database. The new types of information that have been added are “Input Type” and “Concrete Value”. “Input Type” was added to prevent invalid scenarios from

being generated. For example, in our previous tool, if the Process Name and Node Type matched, a textfield node may be replaced with a radio button node. This obviously would not lead to an executable test case. “Concrete Value” stores values that were used in previous tests, and is used when generating database-based test data.

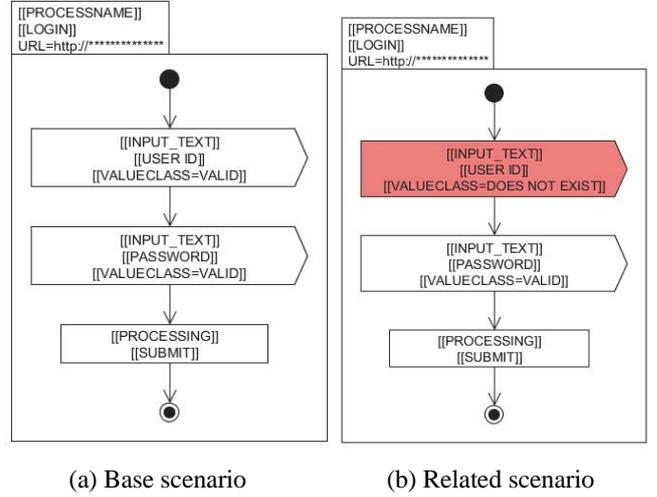


Figure 1. Base scenario and related scenario

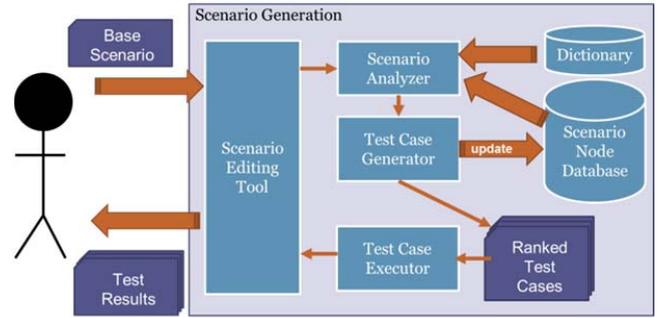


Figure 2. Overview of proposed approach

TABLE I. NODE INFORMATION

	Description
ID	ID for node.
Process Name	The type of processing done by the Web page. (e.g., login, registration)
Node Type	The type of node. (e.g., PROCESSING, INPUT)
Input Type	The type of input. (e.g., textfield, radio button)
Attribute Name	Name of attribute. (e.g., username, password)
Value Class	The type of value. (e.g., empty, exist)
Concrete Value	Values that were used in previous tests.
Frequency	The number of times this node was used in previous tests.

TABLE II. CONSTRAINTS FOR TEXT INPUT

Constraint	Description
More	String length must be greater than specified number
Less	String length must be less than specified number
Only	String can only take a certain type of character
Must	String must include a certain type of character
Must Not	String must not include a certain type of character

When a tester specifies a base scenario, he/she may describe constraints for each node. Table II shows text input constraints that the tester can encode for nodes in the base scenario. This did not exist in our previous work, and enables the generation of constraint-based test data. Note that this information is not saved in the database.

For example, the constraint “more” specifies that the length of a string must be greater than a certain number. Thus, in case of a password, the tester can specify “more=7”, which means that the length of the password must be greater than or equal to 7. The constraint “must” specifies that the string must include a certain type of character. Thus, in case of a password, the tester can specify “must=NUMBER AND ALPHABET AND SYMBOL”, which means that the password must have at least one number, one alphabetical letter, and one symbol. If the tester wants to specify a specific character, then he/she can specify “must=“AbC””, which means that the string “AbC” must be included.

These constraints could have been specified using existing constraint languages, such as OCL, but we intentionally kept this simple so that testers can easily specify the constraints without special training.

**D. Test Data Generation**

We generate test data by combining the base scenario with information in the knowledge base. Our previous work generated related scenarios by replacing nodes [1]. In this paper, we go one step further; we automatically generate test data based on two approaches: (1) database-based approach and (2) constraint-based approach.

In the database-based approach, there are three ways to generate data:

1. Manual Input: The tester manually specifies the test data.
2. Altered Data: The value in the original node of the base scenario is altered. In the case of text input, one character in the value (string) is randomly changed.
3. Previous Data: We check the Concrete Value field of the node that we will use to replace. If a value exists, then it is used. Of course, if there are no values in the Concrete Value field, then our tool cannot automatically generate test data.

In the constraint-based approach, we can generate data as follows:

1. Manual Input: The tester manually specifies the test data.

2. Automatic Generation: Randomly generate data each of which does not satisfy one constraint, but satisfies all other constraints.

Based on the above two test data generation approaches, our tool operates as follows. Our tool searches for nodes in the node database that match the Process Name and Attribute Name, i.e., this search corresponds to the search for nodes that can be used to replace nodes in the base scenario, and generate related scenarios.

Next, a data selection window pops up (Fig. 3). In this example, the Process Name is “LOGIN”, Attribute Name is “USER ID”, the Value Class is “DOES NOT EXIST”, and the data in the original node is “johnny825”. The data selection window shows four options: (1) skip generation of test data, (2) manually specify test data, (3) use test data that has been automatically generated by altering the data in the original node (which results in “johuny825”), and (4) use data that exists in the Concrete Value field (which results in “johnnyjohnny”). The tester chooses one of these options. This process of data selection will continue until all nodes that can be replaced have been processed.

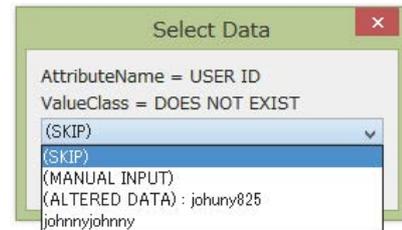


Figure 3. Concrete value selection menu for database-based approach

After generating database-based test data, we generate constraint-based test data. We take text input node as an example. Text input nodes may have constraints described. For each constraint, an abnormal node is generated where the VALUECLASS is as shown in Table III. For example, if there is a “more” constraint, then we need to generate test data that will violate that constraint. In other words we need to generate test data that will be “SHORT”. If there is a constraint “must=XX AND YY”, then “XX” and “YY” must not be included, and thus the VALUECLASS will be “DOES NOT HAVE XX” and “DOES NOT HAVE YY”.

TABLE III. CONSTRAINTS FOR TEXT INPUT

Constraint	VALUECLASS
more=XX	SHORT
less=XX	LONG
only=XX AND YY	NOT ONLY XX AND YY
must=XX AND YY	DOES NOT HAVE XX, DOES NOT HAVE YY
mustnot=XX AND YY	HAVE XX, HAVE YY

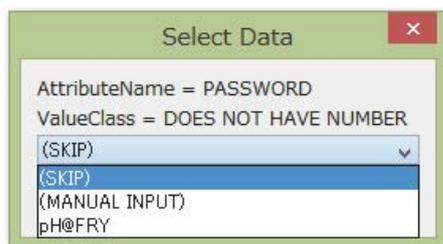


Figure 4. Concrete value selection menu for constraint-based approach

Next, a data selection window pops up (Fig. 4). In this example, the Process Name is “REGISTRATION” and the Attribute Name is “PASSWORD”. The original node had “must=NUMBER” as a constraint; thus the newly generated node will have VALUECLASS as “DOES NOT HAVE NUMBER”. The data selection window shows three options: (1) skip generation of test data, (2) manually specify test data, and (3) automatic generation of test data, which in this case is “pH@FRY”.

The automatic generation of constraint-based test data is done as follows:

1. Decide on string length: If there is a “more” or “less” constraint, then the length is set to that value. If both “more” and “less” constraints exist, then the length is set to the median value. If no constraint exists, then the length is set to the original string length.
2. Decide on the characters to be used: The decision is made based on the “only”, “must”, and “mustnot” constraints.
3. Generate string: Based on the above, a string is randomly generated.
4. Confirm: If the generated string does not satisfy a string constraint, then go back to step 3. This last step is done as there may be multiple constraints for a node, and we need to check if the randomly generated string will not violate the other constraints.

#### E. Test Case Execution

The generated test data is executed with our *Test Case Executer*. Our Test Case Executer uses Selenium WebDriver [10], which enables the automatic operation of a Web page. For each test data, the following four steps are executed:

1. Launch a Web browser.
2. Execute the operation specified in each node (e.g., text input, button press).
3. Save a screenshot of the result.
4. Close the Web browser.

A screenshot of the result is saved in step 3 so that the tester can check the result as necessary.

## IV. CASE STUDY

We conducted a case study to evaluate our approach focusing on the following three research questions:

- RQ1: When different users use our tool to generate test data for the same Web application, how different are they?
- RQ2: How different are the tool-generated test data and test data that are manually created by a professional?
- RQ3: For test data that were created manually, can they be used to update the knowledge base?

RQ1 is posed to check how the tool can be used. Since the test data is generated with a knowledge base, if all users use the same knowledge base, then there should not be much of a difference between different users.

RQ2 checks what types of test data cannot be generated. But since this is likely tied to how well the knowledge base is made, RQ3 is a follow up to RQ2, i.e., if the knowledge base had been made better, what would the difference between the tool-generated test data and manually created test data be?

#### A. RQ1: Difference between Different Tool Users

Six students majoring in computer science took part in our case study. After first getting instructions on how to use our tool, each student “played” with the tool. They were then instructed to generate test data for the login, registration and search Web pages for three Web applications: Cyclos [11], Tapestry5 [12], and Redmine [13].

Table IV shows the results for the total number of generated test data and the total amount of time that each subject took. The result shows that overall there was not much of a difference between the subjects. The time taken to draw the base scenario and then generate the test cases was between 73 and 86 minutes for a difference of 13 minutes (15%). The total number of generated test data ranged between 118 and 132 for a difference of 14 (11%).

TABLE IV. TOOL GENERATED TEST DATA RESULTS

Subject	A	B	C	D	E	F
Total #	118	123	123	129	120	132
Time (min)	76	86	81	80	78	73

A closer inspection of the generated test data found that the total number of types of test cases, i.e., scenarios, was 134. Among these, all six subjects generated the same 106 types (79%), while five out of the six subjects generated 121 types (90%). The actual test data itself was not always the same, because different concrete values from past test data were chosen for the same scenario, and the constraint-based test data were different because they were randomly generated. This clearly indicates that regardless of the tester, most of the generated test cases will be the same.

We also checked for invalid test cases, and found three. These occurred when the user erroneously manually created test data. For example, the VALUECLASS stated “HAVE NUMBER”, but the test data was “ABCD”, which clearly does not have a number included. All cases where the user decided to automatically generate the test data were valid.

### B. RQ2: Difference between Tool-Generated Test Data and Manually Created Test Data

We asked a professional software engineer to manually create test cases for the three Web applications. The software engineer took a total of about 175 minutes to create the test cases, which is more than twice as long as it took the students to generate the test cases. The total number of test cases that the professional engineer created was 114. This is less than the number of test cases that each of the subject created. Thus we can say that our tool can be used by testers that do not necessarily have deep knowledge of how to create tests.

We next check the type of test cases that were generated. Fig. 5 (a) shows the breakdown in Venn diagram format.

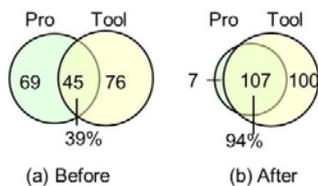


Figure 5. Generated test cases before and after knowledge base update

In Fig. 5 (a), “Pro” shows the number of test cases that the professional engineer manually created, while “Tool” shows the number of test cases that the students generated with our tool. Note that for the tool generated test cases, we did not use the entire set; instead, we used the 121 test cases that five out of the six subjects generated, since we believe that this will better represent the tool generated set of test cases.

From Fig. 5 (a), 69 test cases were created only by the professional engineer, while 76 test cases were generated only by our tool. There was an overlap of 45 test cases that both the professional and our tool created. In other words, 39% ( $=45/(69+45)$ ) of the test cases created by the professional were also generated by our tool. This is not necessarily a high number. But since our tool is knowledge-based, this is likely due to not enough information being stored in our knowledge base. We consider this further in the next subsection.

### C. RQ3: Updating the Knowledge Base

Since our tool is knowledge-based, the results in the previous subsection may have been better if the knowledge base was populated better. We thus added the knowledge base with the information from the test cases that the professional created, but our tools did not. We then automatically generated the test cases again, using the base scenarios that the student subjects had made. Fig. 5 (b) shows the results.

As Fig. 5 (b) shows, by populating the knowledge base, 94% ( $=107/(7+107)$ ) of the test cases that the professional created can be generated by our tool. The remaining seven can be categorized into the following two types: (1) No concrete value possible due to constraints, and (2) impossible to add to database due to ambiguity

Concerning the first type, when our tool generates database-based test data, the test data must satisfy the constraints that have been set for that node. Test data that do not satisfy

constraints can only be generated through constraint-based generation. For example, the professional engineer specified a test case where the input text is one letter long. In our tool, this would correspond to a test case where VALUECLASS=ONE LETTER. Unfortunately, if the node has a constraint which states that the input text must be three or more letters long, it will be impossible to generate test data, since the length of a text cannot be both one letter and more than three letters long.

As for the second type, in some cases, the professional created a test case where the meaning will depend on the context as well as constraints. In one case, the professional specified “invalid characters for user name”. An invalid character can differ between applications, and thus can only be handled through constraint-based test data generation.

## V. CONCLUSION

We proposed a test data generation scheme based on database and constraints. Along with our previous work on test scenario generation, we can automatically generate test cases, i.e., combination of test scenarios and test data. Our tool can automatically execute these test cases. We showed the results of a case study.

The main part of future work concerns the knowledge base itself. As with any knowledge-based approaches, our tool is completely reliant on what is currently included in the knowledge base. We thus need to consider how we can systematically update the knowledge base. We also need to consider scalability issues; updating the database will lead to more test cases, but will there be any tendencies as to what type of test cases are more useful than others?

## REFERENCES

- [1] R. Lacanienta, S. Takada, H. Tanno, and M. Oinuma, “A Knowledge-based Approach for Generating Test Scenarios for Web Applications”, Proc. of SEKE2013, pp.166-171, 2013.
- [2] C. Pacheco and M. Ernst, “Randoop: Feedback-directed Random Testing for Java.” Proc. of OOPSLA 2007 Companion, 2007.
- [3] J.C. King, “Symbolic Execution and Program Testing”, Comm. Of ACM, Vol. 19, No. 7, pp.385-394, 1976.]
- [4] K. Sen, D. Marinov, G. Agha, “CUTE: A Concolic Unit Testing Engine for C”, Proc. of ESEC-FSE '05, pp.263-272, 2005.
- [5] P. McMinn, “Search-Based Software Testing: Past, Present and Future”, Proc. of SBST 2011, pp.153-163, 2011.
- [6] F Gross, G. Fraser, A. Zeller, “Search-based system testing: high coverage, no false alarms”, Proc. of ISSTA 2012, pp.67-77, 2012.
- [7] L. Mariani, M. Pezze, O. Riganeli, M. Santoro, “AutoBlackTest: Automatic Black Box Testing of Interactive Applications”, Proc. of ICST 2012, pp.81-90, 2012.
- [8] N. Alshahwan, M. Harman, “Automated Web Application Testing using Search Based Software Engineering”, Proc. of ASE 2011, pp.3-12, 2011.
- [9] S.Fujiwara, K. Munakata, Y. Maeda, A. Katayama, T. Uehara, “Test data generation for web application using a UML class diagram with OCL constraints”, Innovations in Systems and Software Engineering, Vol. 7, No. 4, pp.275-282, 2011.
- [10] Selenium WebDriver, <http://docs.seleniumhq.org/projects/webdriver/>
- [11] Cyclos, <http://www.cyclos.org>
- [12] Tapestry5 Hotel Booking, <http://tapestry.zones.apache.org:8180/tapestry5-hotel-booking/>.
- [13] Redmine, <http://demo.redmine.org>

# Towards a Unified Metrics Suite for JUnit Test Cases

Fadel Toure, Mourad Badri

Software Engineering Research Laboratory  
Department of Mathematics and Computer Science  
University of Quebec, Trois-Rivières, Quebec, Canada  
{Fadel.Toure, Mourad.Badri}@uqtr.ca

Luc Lamontagne

Department of Computer Science and Software Engineering  
Laval University, Quebec, Canada  
Luc.Lamontagne@ift.ulaval.ca

**Abstract** — This paper aims at proposing a unified metrics suite that can be used to quantify different perspectives related to the code of JUnit test cases. We extended existing JUnit test case metrics by introducing two new metrics. We analyzed the code of JUnit test cases of two open source Java software systems (ANT and JFREECHART). We used in total five metrics. We used the Principal Component Analysis (PCA) method in order: (1) to better understand the underlying orthogonal dimensions captured by the suite of unit test case metrics, and (2) to find whether the metrics are independent or are measuring similar structural aspects of the JUnit test code. Overall, results show that: (1) the new introduced unit test case metrics are relevant, (2) the studied unit test case metrics are not independent, and (3) the best subset (a couple) of unit test case metrics that maximizes the variance varies from one system to the other. The new introduced metrics are, however, each in the best subset of unit test case metrics that provide the best independent information that maximizes the variance for each system.

**Keywords** - Software Testing, Unit Testing, JUnit, Code, Metrics, Principal Components Analysis.

## I. INTRODUCTION

Software testing plays an important role in software quality assurance. It is, however, a time and resource consuming process. The overall effort spent on testing depends, in fact, on many different factors, including human factors, testing techniques, used tools, characteristics of the software development artifacts, and so forth. We focus, in this paper, on the effort involved to write unit tests. Software metrics can be used to quantify different perspectives related to unit test case construction.

This paper aims at proposing a unified metrics suite that can be used to quantify different perspectives related to the code of JUnit test cases. In practice, such metrics can be used to evaluate the unit testing effort. We analyzed the code of JUnit test cases of two open source Java software systems (ANT and JFREECHART). We used in total five metrics. We extended, in fact, existing JUnit test case metrics by introducing two new metrics.

In order to better understand the underlying orthogonal dimensions captured by the studied suite of unit test case metrics, we performed a Principal Component Analysis (PCA). We used this technique to find whether the unit test case

metrics are independent or are measuring similar structural aspects of the unit test code. The main goal of the study is, in fact, to identify a subset of independent unit test case metrics that can be used to quantify different perspectives related to the code of JUnit test cases. Such quantification support can be used to explore (and validate) the relationships between the characteristics of software development artifacts, particularly the source code, and different perspectives related to the effort involved in the construction of corresponding unit tests.

The rest of this paper is organized as follows: Section 2 gives a brief survey of related work. The unit test case metrics are introduced in Section 3. Section 4 presents the empirical study we performed to better understand the underlying orthogonal dimensions captured by the suite of unit test case metrics. Finally, Section 5 concludes the paper.

## II. RELATED WORK

Only few studies in the literature have addressed the quantification of JUnit test cases. JUnit test code has, however, been used in various studies addressing for example the testing coverage [1] or the relationships between the units under test and the corresponding test code [2, 3].

Bruntink and Van Deursen [4, 5] investigate factors of testability of object-oriented software systems. The authors studied five open source Java software systems in order to explore the relationships between object-oriented metrics and some characteristics of JUnit test cases. Testability was measured by the number of lines of test code and the number of *assert* statements in the test code. Results show that there is a significant relationship between the used object-oriented metrics and the measured characteristics of JUnit test classes.

Singh and Saha [6] focus on the prediction of the testability of Eclipse at the package level. Testability was measured using several metrics including the number of lines of test code, the number of *assert* statements in the test code, the number of test methods and the number of test classes. Results show that there is a significant relationship between the used object-oriented metrics and test metrics.

Badri *et al.* [7] explore the relationship between lack of cohesion metrics and testability in object-oriented software systems. In [8], Badri *et al.* investigate the capability of lack of cohesion metrics to predict testability of classes using logistic regression methods. In these studies also, testability was

measured by the number of lines of test code and the number of *assert* statements in the test code. Results show that lack of cohesion is a significant predictor of unit testability of classes.

Badri and Toure [9] explore the capacity of object-oriented metrics to predict the unit testing effort of classes using logistic regression analysis. Results indicate, among others, that multivariate regression models based on object-oriented design metrics are able to accurately predict the unit testing effort of classes. The same test case metrics have been used in this study.

Zhou et al. [10] investigate the relationships between the object-oriented metrics measuring structural properties and unit testability of a class. The investigated structural metrics cover five property dimensions, including size, cohesion, coupling, inheritance and complexity. In this study, the size of a test class is used to indicate the effort involved in unit testing.

### III. JUNIT TEST CASE METRICS

In order to quantify a JUnit test class, we used the following three metrics:

- TLOC: this metric gives the number of lines of code of a test class [4]. It is used to indicate a perspective of the size of the test class.
- TASSERT: this metric gives the number of invocations of JUnit *assert* methods that occur in the code of a test class [4]. JUnit assert calls are, in fact, used by the testers to compare the expected behavior of the class under test to its current behavior. This metric is used to indicate another perspective of the size of a test class. It is directly related to the construction of test cases.
- TNOO: this metric counts the number of methods in a test class [6]. It indicates another perspective of the size of a test class.

The metrics TLOC and TASSERT have been introduced by Bruntink and Van Deursen in [4, 5] to indicate particularly the size of a test suite. The authors used an adapted version of the fish bone diagram developed by Binder in [11] to identify testability factors. The used test case metrics reflect, in fact, different source code factors [4, 5]: factors that influence the number of required test cases and factors that influence the effort involved to develop each individual test case. These two categories have been referred as test case generation and test case construction factors.

Moreover, some classes, depending on the design and particularly on the collaboration between classes, will require drivers and/or monitors to achieve unit testing. We believe that this will also affect the effort involved in the construction of test cases. By analyzing the source code of the JUnit test classes of the systems we selected for our study, we observed that some characteristics related to the interactions between classes and/or objects creation are not captured by the three unit test case metrics TLOC, TASSERT and TNOO. In order to capture these additional dimensions, which also affect in our opinion the effort involved to developing each individual test case, we used two additional metrics:

- TINVOK: this metric counts the number of method invocations in a test class. It captures particularly the dependencies needed to run the test class.
- TDATA: this metric gives the number of new data (objects) created in a test class. These data are required to initialize the test.

We assume that the effort necessary to write a test class  $C_t$  corresponding to a source code class  $C_s$  is proportional to the characteristics measured by the selected unit test case metrics.

## IV. EMPIRICAL STUDY

### A. The Case Studies

Two open source Java software systems were selected for the study:

- ANT (<http://www.apache.org/>) is a Java library and command-line tool that drives processes described in build files as targets and extension points dependent upon each other. This system consists of 713 classes with a total of roughly 64 000 lines of code. JUnit test classes have been developed for 111 source classes, which represents a percentage of 15.60%.
- JFREECHART (JFC) (<http://www.jfree.org/jfreechart/>) is a free chart library for Java platform. This system consists of 496 classes with a total of roughly 68 000 lines of code. JUnit test classes have been developed for 226 source classes, which represents a percentage of 45.60%.

### B. Goals, Research Methodology and Data Collection

In order to better understand the underlying orthogonal dimensions captured by the suite of unit test case metrics, we performed a Principal Component Analysis (PCA). PCA is a technique that has been widely used in software engineering to identify important underlying dimensions captured by a set of software metrics. We used this technique to find whether the unit test case metrics are independent or are capturing the same underlying dimensions (properties) of the JUnit test cases.

We selected from each of the investigated systems only the classes for which JUnit test cases exist. We noticed that developers usually name the JUnit test case classes by adding the prefix (suffix) “Test” (“TestCase”) into the name of the classes for which JUnit test cases were developed. Only classes that have such name-matching mechanism with the test case class name are included in the analysis. This approach has already been adopted in other studies [1].

JUnit (<http://www.junit.org/>) is, in fact, a simple Framework for writing and running automated unit tests for Java classes. Test cases in JUnit are written by testers in Java. JUnit gives testers some support so that they can write those test cases more conveniently. A typical usage of JUnit is to test each source code class  $C_s$  of the program by means of a dedicated test class  $C_t$ . To actually test a class  $C_s$ , we need to execute its test class  $C_t$ . This is done by calling JUnit’s test runner tool. JUnit will report how many of the test methods in  $C_t$  succeed, and how many fail.

However, we noticed by analyzing the JUnit test case classes of the subject systems that in some cases there is no one-to-one relationship between JUnit classes and tested classes. This has also been noted in other previous studies (e.g., [2, 3]). In these cases, several JUnit test cases have been related to a same tested class. The matching procedure has been performed on the subject systems by three research assistants separately (a Ph.D. student (first author of this paper) and two Master students, both in computer science). For each software class  $C_s$  selected, we used the suite of unit test case metrics to quantify the corresponding JUnit test class (classes)  $C_t$ . We used a tool that we developed (JUnit code static analyzer).

### C. Principal Components Analysis

Principal Component Analysis (PCA) is a statistical technique that has been widely used in software engineering to identify important underlying dimensions captured by a set of software metrics (variables). It is a useful technique that aims to reduce variables. PCA is, in fact, a standard technique to identify the underlying, independent/orthogonal dimensions that explain relationships between variables in a data set [12].

From  $M_1, M_2, M_3, \dots, M_n$  metrics, PCA creates new artificial components  $P_1, P_2, P_3, \dots, P_m$  such as:

- $P_i$  are independent,
- $P_i$  are linear combinations of  $M_i$ ,
- and each  $P_i$  maximizes the total variance.

The linear factors are called loadings and the variables with high loadings require some degree of interpretation. In order to find out these variables and interpret the new components, we focused on rotated component. Orthogonal rotation is performed to improve the interpretation of results. There are various strategies to perform such rotation. According to literature, Varimax is the most frequently used strategy [13, 14]. The sum of squared values of loadings that describe the dimension is referred to as eigenvalue.

Since PCA is a projection method in a smaller dimension space, projected variables may seem close in the small space but far from each other in the real space, according to the projection direction. In order to avoid misinterpretation of the new components, the square cosines are computed. A value closed to zero indicates that the point is far from the projection axe. A large proportion of the total variance (information captured by unit test case metrics) is usually explained by the first few PCs. We reduce the metrics without a substantial loss of the explained information by selecting the first PCs. Three criteria are generally used to determine the factors to retain for interpretation:

- (1) The Scree test [15] is based on the decreasing curve of eigenvalues analysis. Only the factors that appear before the first inflection point detected on the curve are considered.
- (2) The cumulative amount of variance criterion considers only the first components that cumulative amount of variance is greater than a given value (in most cases 80%).
- (3) The eigenvalue criterion considers only factors with associated eigenvalue greater than 1 [12].

We used criterion (2) in our case, which guarantee us to consider at least 80% of variance information captured by all metrics. We used the XLSTAT tool (<http://www.xlstat.com>) to perform the PCA analysis.

### D. Results and Discussion

#### ANT

Table 1 gives the descriptive statistics of the unit test case metrics for ANT. Table 2 presents the correlations (Pearson) values between the unit test case metrics. We applied the typical significance threshold ( $\alpha = 0.05$ ) to decide whether the correlations between the metrics values were significant. The correlation values that are significant are in boldface. The Pearson's correlation coefficient is widely used for measuring the degree of linear relationship between two variables. Correlation coefficients will take a value between -1 and +1. A positive correlation is one in which the variables increase (or decrease) together. A negative correlation is one in which one variable increases as the other variable decreases. A correlation of +1 or -1 will arise if the relationship between the variables is exactly linear. A correlation close to zero means that there is no linear relationship between the variables.

Table 1: Descriptive statistics of the metrics – ANT.

	Min	Max	Mean	$\sigma$
TINVOK	20.000	118.000	83.721	11.747
TDATA	0.000	47.000	4.559	7.572
TASSERT	0.000	165.000	12.027	20.684
TLOC	8.000	493.000	73.162	82.419
TNOO	1.000	40.000	6.432	5.986

Table 2: Correlations between metrics – ANT.

	TINVOK	TDATA	TASSERT	TLOC	TNOO
TINVOK	<b>1</b>	0.150	0.072	<b>0.377</b>	<b>0.462</b>
TDATA	0.150	<b>1</b>	<b>0.739</b>	<b>0.616</b>	0.115
TASSERT	0.072	<b>0.739</b>	<b>1</b>	<b>0.746</b>	<b>0.318</b>
TLOC	<b>0.377</b>	<b>0.616</b>	<b>0.746</b>	<b>1</b>	<b>0.588</b>
TNOO	<b>0.462</b>	0.115	<b>0.318</b>	<b>0.588</b>	<b>1</b>

It can be seen from Table 2 that the obtained correlation values between the unit test case metrics are not all significant. Overall, we can observe that the TLOC metric is significantly related to the four other unit test case metrics, which is a plausible finding. We can also observe that the highest correlation values are obtained for the pairs of metrics (TLOC, TASSERT), (TASSERT, TDADA) and (TDADA, TLOC). Moreover, it can be seen that the TINVOK metric is the metric that is the least correlated to the other unit test case metrics. In addition, as we can see, the correlation values between the unit test case metrics are positive. A positive correlation is one in which both variables increase (or decrease) together. These results are plausible and not surprising.

Table 3: PCA results – ANT.

	F1	F2	F3	F4	F5
<b>EigenValue</b>	2,765	1,252	0,602	0,212	0,169
<b>Variability(%)</b>	46,6	33,737	12,049	4,235	3,379
<b>% Cumulated</b>	46,6	80,337	92,386	96,621	100
	<b>Correlation</b>			<b>Square cosine</b>	
	<i>F1</i>	<i>F2</i>		<i>F1</i>	<i>F2</i>
<b>TINVOK</b>	0,013	0,843		0	0,71
<b>TDATA</b>	0,899	-0,002		0,809	0
<b>TASSERT</b>	0,934	0,102		0,873	0,01
<b>TLOC</b>	0,775	0,515		0,601	0,265
<b>TNOO</b>	0,217	0,837		0,047	0,701

Table 4: Multicollinearity analysis – ANT.

ANT	TINVOK	TDATA	TASSERT	TLOC	TNOO
<b>R<sup>2</sup></b>	0.319	0.617	0.719	0.739	0.499
<b>Tolerance</b>	0.681	0.383	0.281	0.261	0.501
<b>VIF</b>	1.469	2.612	3.555	3.828	1.997

Table 3 presents the PCA results for ANT. It gives the variability of new components, their correlation with unit test case metrics, and the square cosine of projection (metrics) in the new components. From Table 3, it can be seen that the components F1 and F2 cumulate more than 80 % of total variance (80.337%), which leads us to interpret only F1 and F2. The component F1 is represented by the metrics TASSERT (0.934), TDATA (0.899) and TLOC (0.775). The component F2 is represented by the metrics TINVOK (0.843) and TNOO (0.837).

The two components F1 and F2 oppose, in fact, the group of large test classes (high TLOC) having relatively a high verification effort and data creation (high values of TASSERT and TDATA) to the group of classes that contains many method invocations (TINVOK) with high number of operations (TNOO). High contribution of the metrics TDATA, TASSERT and TLOC in the first component indicates that, in the large majority of test classes, data creation and number of assertions increase with the size of test classes (line of codes). The independence between F1 and F2 indicates that, in some test classes, the number of methods and invocations increase together independently of the metrics TDATA, TLOC, and TASSERT.

The overall information (related to unit testing writing effort) captured by the suite of unit test case metrics is distributed in the two dimensions F1 and F2, which can be represented by one of the couples {TASSERT, TDATA, TLOC} × {TINVOK, TNOO}. The couple (TASSERT, TINVOK) provides, however, the best independent information that maximizes the variance.

Moreover, we performed a multicollinearity analysis between the unit test case metrics. Table 4 gives the results. Multicollinearity is, in fact, a statistical phenomenon in which two or more predictor variables (unit test case metrics in our case) in a multiple regression model are highly correlated. Multicollinearity suggests, in fact, that several of the independent variables are closely linked in some way. This means that one variable can be linearly predicted from the others with a non-trivial degree of accuracy. The simplest way to resolve multicollinearity problems is to reduce the number of collinear variables until there is only one remaining out of the set. Some authors have suggested a formal detection of multicollinearity using the VIF (variance inflation factor). The VIF is defined as :  $VIF = 1 / \text{tolerance}$ , where  $\text{tolerance} = 1 - R^2$  and  $R^2$  is the coefficient of determination of a regression of variable  $j$  on all the other variables. A tolerance of less than 0.20 indicates a multicollinearity problem. The problem with such data redundancy is that of over fitting in regression analysis models. As we can see from Table 4, the VIF (variance inflation factor) of the unit test case metrics are all less than 4. Moreover, all tolerance values are greater than 0.20, which indicates that there is no multicollinearity problem in the case of ANT. From Table 4, it can also be seen that the metric TINVOK has the lowest VIF.

#### JFC

Table 5: Descriptive statistics of the metrics – JFC.

	Min	Max	Mean	$\sigma$
<b>TINVOK</b>	5.000	118.000	22.146	13.183
<b>TDATA</b>	4.000	265.000	23.925	30.900
<b>TASSERT</b>	1.000	143.000	17.956	21.807
<b>TLOC</b>	18.000	635.000	91.403	82.214
<b>TNOO</b>	2.000	45.000	5.774	4.541

Table 6: Correlations between metrics – JFC.

	TINVOK	TDATA	TASSERT	TLOC	TNOO
<b>TINVOK</b>	<b>1</b>	<b>0.491</b>	<b>0.792</b>	<b>0.836</b>	<b>0.671</b>
<b>TDATA</b>	<b>0.491</b>	<b>1</b>	<b>0.735</b>	<b>0.742</b>	<b>0.444</b>
<b>TASSERT</b>	<b>0.792</b>	<b>0.735</b>	<b>1</b>	<b>0.922</b>	<b>0.652</b>
<b>TLOC</b>	<b>0.836</b>	<b>0.742</b>	<b>0.922</b>	<b>1</b>	<b>0.772</b>
<b>TNOO</b>	<b>0.671</b>	<b>0.444</b>	<b>0.652</b>	<b>0.772</b>	<b>1</b>

Table 5 gives the descriptive statistics of the unit test case metrics for JFC. Table 6 presents the correlations (Pearson) values between the unit test case metrics. Here also, we applied the typical significance threshold ( $\alpha = 0.05$ ) to decide whether the correlations between the metrics values were significant. It can be seen, from Table 6, that the obtained correlation values between the unit test case metrics are all significant (in boldface). Overall, we can observe that, here also, the TLOC metric is significantly related (with relatively high correlation values) to the four other unit test case metrics. We can also observe that the highest correlation values are obtained for the pairs of metrics (TLOC, TASSERT), (TLOC, TINVOK) and

(TASSERT, TINVOK). Moreover, it can be seen that the lowest value of correlation is observed for the pair (TNOO, TDATA). We can also observe that the metrics TNOO and TDATA are the less correlated to the other unit test case metrics. Moreover, as we can see, the correlation values between the unit test case metrics are here also positive.

In the case of JFC (see Table 7), the cumulated variance of F1 and F2 (89.28%) suggests to limit the interpretation to the two first components. The component F1 regroups the metrics TNOO (0.887), TINVOK (0.837) and TLOC (0.746). The component F2 is represented by TDATA (0.951). TASSERT, in spite of its relative high correlation with the component F2 (0.694), is far from the projection axe as shown by its low square cosine ( $0.481 < 0.5$ ). TASSERT provides insignificant information in the considered set of unit test case metrics.

Table 7: PCA results – JFC.

	F1	F2	F3	F4	F5
<b>Eigenvalue</b>	3,852	0,612	0,346	0,139	0,051
<b>Variability(%)</b>	50,152	39,124	6,923	2,771	1,029
<b>% Cumulated</b>	50,15	89,28	96,2	98,97	100
	<b>Correlation</b>			<b>Square cosine</b>	
	<b>F1</b>	<b>F2</b>		<b>F1</b>	<b>F2</b>
<b>TINVOK</b>	0,837	0,358		0,701	0,129
<b>TDATA</b>	0,209	0,951		0,044	0,905
<b>TASSERT</b>	0,647	0,694		0,419	0,481
<b>TLOC</b>	0,746	0,634		0,556	0,402
<b>TNOO</b>	0,887	0,197		0,787	0,039

Table 8: Multicollinearity analysis – JFC.

JFC	TINVOK	TDATA	TASSERT	TLOC	TNOO
<b>R<sup>2</sup></b>	0.746	0.653	0.866	0.930	0.647
<b>Tolerance</b>	0.254	0.347	0.134	0.070	0.353
<b>VIF</b>	3.931	2.882	7.471	14.318	2.830

For JFC, the two first components oppose, in fact, the most important set of classes containing a relatively high number of methods, having many invocations and large number of lines of code, to the set of classes with many data creation. One of the couples {TNOO, TINVOK, TLOC} × {TDATA} could represent the set of unit test case metrics for JFC. The couple (TNOO, TDATA) is, however, the best representative of the suite of unit test case metrics.

In the case of JFC also, we performed a multicollinearity analysis between the unit test case metrics. Table 8 gives the results. As we can see, the VIF (variance inflation factor) of the metrics TNOO, TDATA and TINVOK (respectively 2.830, 2.882 and 3.931) are less than 4, unlike those of the metrics TLOC and TASSERT (respectively 14.318 and 7.471). This indicates that there is a multicollinearity problem in the case of JFC, unlike ANT. Indeed, the metrics TLOC and TASSERT

are the most linearly related to the other unit test case metrics. The tolerance of the two metrics is less than 0.20 (respectively 0.070 and 0.134). From Table 8, it can also be seen that the metric TNOO has the lowest VIF, followed by the metric TDATA.

Overall, results show that the studied unit test case metrics are not independent. There is, in fact, a certain redundancy in the information captured by the metrics. Results also show that the best representative subset of (independent) unit test case metrics varies from one system to the other. However, as we have seen, the new introduced metrics TDATA and TINVOK are each in the couples that provide the best independent information that maximizes the variance for each system. In fact, the design of the analyzed systems may have an impact on the effort involved to test each class. Moreover, the style adopted by the developers while writing the code of JUnit test cases could also significantly impact the observed values of studied unit test case metrics. We observed, indeed, that the test development style, in general, differs from one system to the other. Further investigations are, however, needed in order to validate these observations and draw more general conclusions.

However, in all results, we can see that the effort of writing test code, in terms of lines of code (TLOC), is significantly correlated with the first component in the case of the two systems, which is a plausible finding. Results show also that the new introduced unit test case metrics, TDATA and TINVOK, capture a part of information that is not captured by the metrics TLOC, TASSERT and TNOO.

#### E. Threats to validity

The study presented in this paper should be replicated using many other case studies in order to draw more general conclusions. The achieved results are based on the data set we collected from only two case studies. The findings in this paper should be viewed as exploratory and indicative rather than conclusive.

It is also possible that facts such as the development style used by the developers for writing test cases and the criteria they used while selecting classes for which they developed JUnit test classes (randomly or depending on their size or complexity e.g., or on other criteria) may affect the results or produce different results for specific applications. Results show, at least, that the new introduced metrics are relevant.

## V. CONCLUSIONS AND FUTURE WORK

We analyzed, in this paper, the JUnit test classes of two case studies. We used five metrics to quantify different perspectives related to their code. We extended, in fact, existing JUnit test case metrics by introducing two new metrics. We used the Principal Component Analysis technique in order to analyze the underlying orthogonal dimensions captured by the studied suite of unit test case metrics. The main goal of the study was to find whether the unit test case metrics are independent or are capturing the same underlying dimensions (properties), and particularly to identify a subset of independent unit test case metrics that can be used to quantify the code of JUnit test classes.

Results show that: (1) the new introduced unit test case metrics are relevant, and (2) the best subset (a couple) of unit test case metrics that maximizes the variance varies from one system to the other. Moreover, the new introduced metrics are each in the best subset of unit test case metrics that provide the best independent information that maximizes the variance for each system.

The performed study should, however, be replicated using many other case studies in order to draw more general conclusions. Also, it would be interesting to investigate the impact of the systems design, and particularly the style adopted by the developers in writing the code of test cases, on the distribution of the unit test case metrics.

#### ACKNOWLEDGMENTS

This work was supported by NSERC (Natural Sciences and Engineering Research Council of Canada) grant.

#### REFERENCES

- [1] A. Mockus, N. Nagappan, and T. T. Dinh-Trong, "Test coverage and post-verification defects: a multiple case study," in Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM '09), pp. 291–301, October 2009.
- [2] B. V. Rompaey and S. Demeyer, "Establishing traceability links between unit test cases and units under test," in Proceedings of the 13th European Conference on Software Maintenance and Reengineering (CSMR '09), pp. 209–218, March 2009.
- [3] A. Qusef, G. Bavota, R. Oliveto, A. De Lucia, and D. Binkley, "SCOTCH: test-to-code traceability using slicing and conceptual coupling," in Proceedings of the International Conference on Software Maintenance (ICSM '11), 2011.
- [4] M. Bruntink and A. Van Deursen, "Predicting class testability using object-oriented metrics", in Proceedings of the 4<sup>th</sup> IEEE International Workshop on Source Code Analysis and Manipulation (SCAM '04), pp. 136–145, September 2004.
- [5] M. Bruntink and A. van Deursen, "An empirical study into class testability", Journal of Systems and Software, vol. 79, no. 9, pp. 1219–1232, 2006.
- [6] Y. Singh and A. Saha, "Predicting testability of eclipse: a case study", Journal of Software Engineering, vol. 4, no. 2, 2010.
- [7] L. Badri, M. Badri, and F. Toure, "Exploring empirically the relationship between lack of cohesion and testability in object-oriented systems", in Advances in Software Engineering, T.-h. Kim, H.-K. Kim, M. K. Khan et al., Eds., vol. 117 of Communications in Computer and Information Science, Springer, Berlin, Germany, 2010.
- [8] L. Badri, M. Badri, and F. Toure, "An empirical analysis of lack of cohesion metrics for predicting testability of classes", International Journal of Software Engineering and Its Applications, vol. 5, no. 2, 2011.
- [9] M. Badri and F. Toure, "Empirical Analysis of Object-Oriented Design Metrics for Predicting Unit Testing Effort of Classes", Journal of Software Engineering and Applications (JSEA), Volume 5, Number 7, July 2012.
- [10] ZHOU YuMing, LEUNG Hareton, SONG QinBao, ZHAO JianJun, LU HongMin, CHEN Lin, and XU BaoWen, "An in-depth investigation into the relationships between structural metrics and unit testability in object-oriented systems", in SCIENCE CHINA, Information Sciences, Vol. 55, No. 12, December 2012.
- [11] Binder, R.V., "Design for Testability in Object-Oriented Systems", Communications of the ACM, Vol. 37, 1994.
- [12] J. T. S. Quah, M. M. T. Thwin, "Application of Neural Networks for Software Quality Prediction Using Object-Oriented Metrics", Proceedings of the International Conference on Software Maintenance (ICSM'03), IEEE Computer Society, 2003.
- [13] Y. Dash, S. K. Dubey, " Application of Principal Component Analysis in Software Quality Improvement", In International Journal of Advanced Research in Computer Science and Software, Engineering Vol. 2, Issue 4, April 2012.
- [14] K.K. Aggarwal, Y. Singh, A. Kaur and R. Malhotra, "Empirical study of object-oriented metrics", In Journal of Object Technology, vol. 5, no. 8, November-December 2006.
- [15] R. B. Cattell, "The scree test for the number of factors", In *Multivariate Behavioral Research*, Volume 1, Issue 2, 1966.

# How to Do Tie-breaking in Prioritization of Interaction Test Suites?

Rubing Huang\*, Jinfu Chen

School of Computer Science and Telecommunication Engineering  
Jiangsu University  
Zhenjiang, Jiangsu 212013, P.R.China

Rongcun Wang, Deng Chen

School of Computer Science and Technology  
Huazhong University of Science and Technology  
Wuhan, Hubei 430074, P.R.China

**Abstract**—The prioritization of interaction test suites has received more attention in the field of combinatorial interaction testing, especially when testing resources are limited to allow the part of combinatorial test cases to be executed. Many strategies have been proposed to prioritize interaction test suites according to different evaluation measures. However, most of these strategies may face a challenge to choose more than one “best” candidate with the largest evaluation measure value. In this case, there is a tie among all “best” candidates. How to do tie-breaking? Intuitively speaking, random tie-breaking could be a reasonable choice, which has also been applied to many research papers. In this paper, we investigate different tie-breaking techniques including random tie-breaking, first-element tie-breaking, last-element tie-breaking, higher-strength tie-breaking, and lower-strength tie-breaking, and also conduct experiments on a well-known prioritization strategy of interaction test suites, namely *interaction coverage based prioritization*, in order to present a guideline of choosing tie-breaking techniques for testers in practical testing. The experimental results show that although no tie-breaking technique always performs best, in many cases random tie-breaking and last-element tie-breaking have best performance, so that they would be best choices for testers in the prioritization of interaction test suites.

**Keywords**—Combinatorial interaction testing, interaction test suite, test case prioritization, tie-breaking, guideline

## I. INTRODUCTION

*Combinatorial interaction testing* (CIT) [1] aims at generating an *interaction test suite* [1], in order to identify failures that are caused by parameter interactions. Intuitively speaking, combinatorial interaction testing presents a tradeoff between testing effectiveness and testing efficiency, because it only focuses on interaction coverage of fixed *strengths* (the levels of interaction among parameters) rather than that of all strengths.

When an interaction test suite  $T$  has been already constructed by CIT, traditional CIT would directly run  $T$  but does not consider the execution order of test cases in  $T$ . However, due to limited test resources in practice, only part of test cases in  $T$  could be executed. In such case, the execution order of test cases in  $T$  would be critical for the whole testing process, because a well-prioritized order of test case execution may be able to identify failures earlier, and thus enable fault characterization, diagnosis and revision

as earlier as possible [1]. The process of determining the order of test cases in  $T$  is generally called test case prioritization. A prioritized interaction test suite is also called an interaction test sequence. As shown in Figure 1, the testing process involved in the dashed box is the traditional CIT, and an enhanced testing process adds test case prioritization between test suite construction and test suite execution in the traditional CIT [2].

To date, many strategies have been proposed to guide the prioritization of interaction test suite according to different evaluation measures, for example *interaction coverage based prioritization* [2–8] and *incremental interaction coverage based prioritization* [9, 10]. However, most of prioritization strategies may face a challenge<sup>1</sup>: During the prioritization process, there may exist more than one “best” candidate such that they have the same largest evaluation measure value. In this case, there is a tie among all “best” candidates. How to do tie-breaking? In this paper, we investigate different tie-breaking techniques such as random tie-breaking [5, 7, 10], first-element tie-breaking, last-element tie-breaking, higher-strength tie-breaking, and lower-strength tie-breaking, and also conduct experiments on the ICBP prioritization algorithm to analyze the effectiveness of tie-breaking techniques, so as to present a guideline of choosing tie-breaking techniques for testers in practical testing. The experimental results indicate that although there is no best tie-breaking technique, in many cases random tie-breaking and last-element tie-breaking have best performance, so that they would be best

<sup>1</sup>Some prioritization strategies do not face this challenge, for example *random test case prioritization* [6], because it only prioritizes an interaction test suite at a random manner. In this paper, therefore, we assume that our study focus on the prioritization strategies with such challenge.

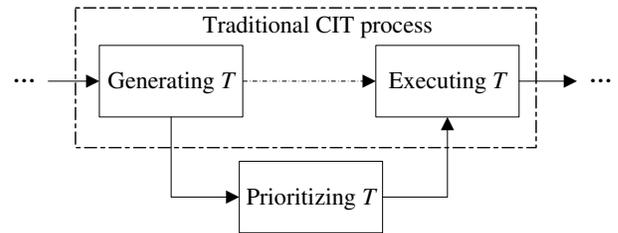


Figure 1. Traditional CIT with test case prioritization [2].

\*Corresponding author: rbhuang@ujs.edu.cn.

choices for testers in the prioritization of interaction test suites.

The remaining part of this paper is organized as follows: Section 2 simply introduces some background information about CIT and test case prioritization. Section 3 presents some related work about our study. Section 4 investigates various methods that can be used to prioritize arbitrary interaction test suites. Section 5 conducts some empirical studies to evaluate different prioritization methods investigated in Section 4. Section 6 analyzes the application range of each method. Finally, Section 7 concludes and discusses future work.

## II. BACKGROUND

In this section, some background information will be described, including CIT, interaction test suite, and test case prioritization.

### A. CIT and Interaction Test Suite

CIT aims at covering some specific combinations of parametric values by using an interaction test suite with the small number of test cases, so as to detect failures that are triggered by parameter interactions.

To clearly describe some notions, we first model the test object in CIT. Assume the SUT has  $k$  parameters that constitute a parameter set  $P = \{p_1, p_2, \dots, p_k\}$ , and each parameter  $p_i$  has some valid values or levels from the finite set  $V_i$  ( $i = 1, 2, \dots, k$ ). In practice, parameters may represent any factors that influence the performance of the SUT, such as components, configuration options, user inputs, etc. Suppose  $\mathcal{C}$  be the set of constraints on combinations of parameter values.

**Definition 1** (Test profile). A *test profile*, denoted as  $TP(k, |V_1||V_2|\dots|V_k|, \mathcal{C})$ , is the model of the SUT, including  $k$  parameters,  $|V_i|$  parameter values for each  $i$ -th parameter, and value combination constraints  $\mathcal{C}$ .

Unless specifically stated, the definitions illustrated in this paper are based on a  $TP(k, |V_1||V_2|\dots|V_k|, \mathcal{C})$ .

**Definition 2** (Covering array). A *covering array* (CA), denoted  $CA(N; \tau, k, |V_1||V_2|\dots|V_k|)$ , is an  $N \times k$  matrix, which satisfies the following properties: (1) each column  $i$  ( $1 \leq i \leq k$ ) contains elements only from the set  $V_i$ ; and (2) the rows of each  $N \times \tau$  sub-matrix cover all possible  $\tau$ -tuples (referred to as  $\tau$ -wise value combinations or  $\tau$ -wise value schemas [1]) from the  $\tau$  columns at least once.

Here,  $\tau$  is called *strength*. Since the strength of a covering array is fixed, a covering array at strength  $\tau$  is also usually called a  $\tau$ -wise covering array.

In a covering array, each row represents a test case; while each column represents a parameter. Generally speaking, in the field of combinatorial interaction testing, a covering array represents an interaction test suite. In this paper, therefore, we assume that a covering array is equivalent to an interaction test suite.

### B. Test Case Prioritization

Test case prioritization seeks to schedule test cases so that those with the highest priority, according to some criterion, are executed earlier in testing than lower priority test cases. When testing resources are limited or insufficient to execute all test cases in a test suite, a well-designed execution order of test cases seems especially significant. A prioritized test suite is generally called a test sequence. The problem of test case prioritization is defined as follows [11].

**Definition 5** (Test case prioritization). Given a tuple  $(T, \Omega, f)$ , where  $T$  is a test suite,  $\Omega$  is the set of all possible permutations of  $T$ , and  $f$  is a function from  $\Omega$  to real numbers, the *test case prioritization* problem is to find a test sequence  $S \in \Omega$  such that:

$$(\forall S')(S' \in \Omega)(S' \neq S)[f(S) \geq f(S')]. \quad (1)$$

According to Rothermel's investigations [11], there are many possible goals of prioritization (that is, different functions of  $f$ ). For example, a well-known function, namely *average percentage of faults detected* (APFD), is related to fault detection, which is widely used in regression testing. The APFD function needs to obtain the fault-detection capability of each executed test case.

## III. TIE-BREAKING TECHNIQUES IN PRIORITIZATION OF INTERACTION TEST SUITES

In this section, we investigate different tie-breaking techniques used in the prioritization of interaction test suites. In order to describe techniques clearly, we apply different tie-breaking techniques to a well-known prioritization strategy of interaction test suites, namely *interaction coverage based prioritization* (in short ICBP). Figure 2 presents the detailed algorithm of ICBP, from which the "best" element is chosen from candidates in  $T$  as the next test case in  $S$  such that it covers the largest number of  $\tau$ -wise value combinations that have not yet covered by  $S$ .

Consider an interaction test suite  $T$  (its original test sequence  $T'$  is obtained according to its generation order), the prioritization strength  $\tau$  for algorithm ICBP, that is, ICBP uses the  $\tau$ -wise interaction coverage, or *uncovered  $\tau$ -wise value combinations distance* (in short  $UVCD_\tau$ ) [12] to guide the prioritization, and an interaction test sequence of  $T$

**Input:** A  $\tau$ -wise CA, denoted as  $T$ .

**Output:** An interaction test sequence  $S$  of  $T$ .

- 1: Initialize  $S$ ;
- 2: **while** ( $T$  is not empty)
- 3:     Select the "best" element  $e$  from  $T$ .
- 4:     Remove  $e$  from  $T$ ;
- 5:     Insert  $e$  into  $S$ ;
- 6: **end\_while**
- 7: **return**  $S$ .

Figure 2. The detailed algorithm description of ICBP.

prioritized by ICBP. We will describe different tie-breaking techniques as follows.

#### A. Random Tie-breaking

Obviously, *random tie-breaking* is an intuitive technique, because it randomly breaks the tie among all “best” candidates. For example, two combinatorial test cases  $tc_1$  and  $tc_2$ , in  $T$ , have the same  $UVCD_\tau$  value, ICBP will randomly choose one from  $tc_1$  and  $tc_2$  as the next test case in  $S$ . In other words,  $tc_1$  and  $tc_2$  have the same probability to be selected.

#### B. First-element Tie-breaking

The *first-element tie-breaking* technique denotes that among all “best” candidates, ICBP would select a candidate as the next test case such that it occurs in the original test sequence  $T'$  first.

#### C. Last-element Tie-breaking

Similar to the first-element tie-breaking technique, *Last-element Tie-breaking* chooses the element as the next test case such that it occurs in the original test sequence  $T'$  lastly when facing the challenge of more than one “best” candidate.

As we know, ICBP uses the fixed strength  $\tau$  to guide the prioritization of interaction test suites. When there are two candidates  $tc_1$  and  $tc_2$  of which have the largest  $UVCD_\tau$  value, the strength  $\tau$  is unavailable to distinguish  $tc_1$  and  $tc_2$ . In this case, it is reasonable to consider other strength to assist ICBP in choosing the next test case. Therefore, we consider the following two cases: (1) the tie-breaking technique using the strength higher than  $\tau$ , namely *Higher-strength tie-breaking*; and (2) the tie-breaking technique using the strength lower than  $\tau$ , namely *lower-interaction-coverage tie-breaking*. We will describe them in turn.

#### D. Higher-strength Tie-breaking

When the strength  $\tau$  is unavailable to assist ICBP in selecting the next test case from the “best” candidates, the *Higher-strength tie-breaking* technique uses the strength  $(\tau + 1)$  for ICBP to further calculate each “best” candidate, and then choose one as the next test case in  $S$ . It can be noted that if strength  $(\tau + 1)$  is also unavailable (that is, after using higher interaction coverage, there still exists more than one “best” candidate), ICBP will randomly choose an element from the remaining “best” candidates as the next test case in  $S$ .

#### E. Lower-strength Tie-breaking

Compared to the Higher-strength tie-breaking technique, *Lower-strength tie-breaking* uses the strength  $(\tau - 1)$  to further guide the prioritization when strength  $\tau$  is unavailable. Similar to Higher-strength tie-breaking, if strength  $(\tau - 1)$  is also unavailable, ICBP chooses one element from the

remaining “best” candidates as the next test case at a random manner.

Among all tie-breaking techniques, first-element tie-breaking and last-element tie-breaking are deterministic, which means that given an original interaction test sequence, each of them could obtain a unique interaction test sequence. However, other three tie-breaking techniques are nondeterministic, because they may randomly choose an element from all “best” candidates as the next test case.

## IV. EMPIRICAL CASE STUDY

In this section, an empirical case study is presented to analyze the effectiveness of different tie-breaking techniques in the prioritization of interaction test suites, according to fault detection. We have designed the empirical study to answer the following research questions:

**RQ1:** Among all tie-breaking techniques, which technique is *best* for the prioritization of interaction test suites?

**RQ2:** How to choose the tie-breaking technique in practical testing?

#### A. Setup

We use a medium-sized real-life program from a lexical analyzer system (`flex`), which is obtained from the Software Infrastructure Repository (SIR) [13]. This program involves five versions, and contains 9,581 sin 11,470 un-commented lines of C language code, which are also augmented with a seeded fault library. In this paper, we have used 34 seeded faults. According to Petke’s investigation [8], the test profile of program `flex` is  $TP(9, 2^6 3^2 5^1, \mathcal{C})$ , where  $\mathcal{C} \neq \emptyset$ .

The original covering arrays were generated by two widely-used tools: *Advanced Combinatorial Testing System* (ACTS) [14]; and *Pairwise Independent Combinatorial Testing* (PICT) [15], both of which are supported by greedy algorithms, and respectively implemented by the *In-Parameter-Order* (IPO) method and the *one-test-at-a-time* approach. We focused on covering arrays with strength  $\tau = 2, 3, 4, 5$ . Since some tie-breaking techniques involve randomization such as random tie-breaking, we ran the experiment 100 times for each interaction test suite and report the average.

In practical testing, testing resources may be limited to allow only part of interaction test sequence to be executed. In this study, therefore, we consider different budgets by considering different percentages ( $p$ ) of each interaction test sequence, for instance,  $p = 5\%, 10\%, 25\%, 50\%, 75\%$ , and  $100\%$ .

#### B. Metrics

Generally speaking, APFD has been used to evaluate different prioritization techniques. However, it has two requirements [2, 6, 9, 10], so as to fail to be used as the evaluation metric. Therefore, in this study we used an alternative of APFD, namely *Normalized APFD* (or NAPFD)

[6], in order to evaluate the fault detection rate for each tie-breaking technique.

### C. Results and Analysis

Table I presents the NAPFD metric values for ICBP with different tie-breaking techniques when executing the certain percentage of each interaction test sequence. It can be noted that the bold datum in this table is largest in each sub-column. From this table, we can have the following observations.

1) Among all tie-breaking techniques, which technique is best for the prioritization of interaction test suites? According to data shown in Table I, it can be clearly seen that there is no *best* tie-breaking technique for the prioritization of interaction test suites. In other words, each tie-breaking technique performs best in some cases. For example, random tie-breaking obtains the best NAPFD metric values for ACTS covering arrays at strengths  $\tau = 2$  and  $\tau = 4$  when  $p$  is high; first-element tie-breaking performs best for PICT covering array at strength  $\tau = 5$  regardless of  $p$  value; last-element tie-breaking has the best performance when prioritizing PICT covering array at strength  $\tau = 3$ ; higher-strength tie-breaking has the best rates of fault detection for ACTS covering arrays at strength  $\tau = 3$  when  $p$  is low; and lower-strength tie-breaking behaves best NAPFD metric values for PICT covering array at  $\tau = 2$  when  $p$  is high.

However, on the whole, first-element tie-breaking performs worst in many cases. Therefore, first-element tie-breaking is the last choice. Higher-strength tie-breaking could be a better choice when executing fewer number of combinatorial test cases in the interaction test sequence. Additionally, random tie-breaking and last-element tie-breaking could be best in many cases.

2) How to choose the tie-breaking technique in practical testing? According to the definition of each tie-breaking technique, higher-strength and lower-strength tie-breaking techniques are more time-consuming than other tie-breaking techniques, because they need to count information at higher strength (or lower strength). As discussed before, random tie-breaking and last-element tie-breaking would be better choices according to fault detection rates. As a consequence, in practical testing when testing resources are sufficient, testers would choose the higher-strength tie-breaking technique in the prioritization of interaction test suites, because it needs more prioritization time; when testing resources are limited, random tie-breaking and last-element tie-breaking would be better alternatives, because they has less prioritization time. On the other hand, random tie-breaking and higher-strength tie-breaking are un-deterministic; while last-element tie-breaking is deterministic. Therefore, when testers need to compare their methods with deterministic algorithms, last-element tie-breaking is a better choice; otherwise, random tie-breaking or higher-strength tie-breaking would be better.

### D. Threats to Validity

Despite our best efforts, our experiments may face some threats to validity.

The first threat is the selection of experimental data – in this paper only a medium-sized subject program has been used to investigate the effectiveness of different tie-breaking techniques. Additionally, two tools used to construct interaction test suites are widely used but both of them are greedy. Finally, we applied different tie-breaking techniques to only ICBP algorithm. To address this threat, additional

Table I  
THE NAPFD METRIC (%) FOR ICBP WITH DIFFERENT TIE-BREAKING TECHNIQUES FOR SUBJECT PROGRAM FLEX WHEN EXECUTING THE PERCENTAGE OF INTERACTION TEST SEQUENCE.

Method	Strength	$p$ of ACTS Interaction Test Sequence Executed						$p$ of PICT Interaction Test Sequence Executed					
		5%	10%	25%	50%	75%	100%	5%	10%	25%	50%	75%	100%
<i>Random Tie-breaking</i>	$\tau = 2$	15.12	31.10	<b>62.96</b>	<b>76.10</b>	<b>80.29</b>	<b>82.35</b>	14.94	29.69	61.31	75.12	79.62	82.66
<i>First-element Tie-breaking</i>		5.88	21.32	59.56	74.66	79.41	81.70	8.83	21.32	57.60	73.98	78.48	81.62
<i>Last-element Tie-breaking</i>		13.24	25.74	55.15	72.86	78.24	80.83	7.35	19.85	56.62	74.32	<b>79.64</b>	<b>82.75</b>
<i>Higher-strength Tie-breaking</i>		<b>15.82</b>	<b>31.29</b>	62.15	75.13	79.37	82.26	14.24	28.96	61.12	<b>75.13</b>	79.55	82.49
<i>Lower-strength Tie-breaking</i>		15.38	31.18	62.83	75.87	80.14	82.24	<b>15.29</b>	<b>30.27</b>	<b>61.68</b>	75.12	79.54	82.57
<i>Random Tie-breaking</i>	$\tau = 3$	43.57	62.09	<b>77.66</b>	83.66	85.97	87.88	41.41	60.42	76.65	82.44	84.65	86.18
<i>First-element Tie-breaking</i>		35.29	57.84	76.84	82.71	84.87	87.25	25.00	47.55	70.96	79.60	82.48	84.32
<i>Last-element Tie-breaking</i>		33.82	57.60	76.47	82.75	85.50	87.70	<b>49.51</b>	<b>65.20</b>	<b>79.60</b>	<b>83.92</b>	<b>86.00</b>	<b>87.35</b>
<i>Higher-strength Tie-breaking</i>		<b>43.60</b>	<b>62.10</b>	<b>77.66</b>	83.66	85.95	87.95	41.74	60.95	77.06	82.64	84.62	85.95
<i>Lower-strength Tie-breaking</i>		41.84	61.06	77.39	<b>83.67</b>	<b>86.10</b>	<b>88.02</b>	41.71	60.71	76.96	82.65	84.78	86.28
<i>Random Tie-breaking</i>	$\tau = 4$	59.98	74.36	<b>83.96</b>	<b>88.33</b>	<b>90.54</b>	92.72	<b>61.04</b>	<b>73.40</b>	82.97	86.72	88.86	90.10
<i>First-element Tie-breaking</i>		54.66	70.81	82.22	88.01	<b>90.54</b>	<b>92.94</b>	61.03	73.16	82.77	86.97	89.31	90.54
<i>Last-element Tie-breaking</i>		59.56	72.40	81.80	86.49	88.04	90.84	62.01	73.04	<b>84.33</b>	<b>87.82</b>	<b>89.92</b>	<b>90.99</b>
<i>Higher-strength Tie-breaking</i>		<b>60.41</b>	<b>74.69</b>	83.95	88.22	90.53	92.79	59.47	72.73	82.86	86.57	88.64	89.89
<i>Lower-strength Tie-breaking</i>		59.96	74.35	83.67	88.07	90.20	92.48	60.63	73.36	83.05	86.80	88.82	90.00
<i>Random Tie-breaking</i>	$\tau = 5$	<b>71.39</b>	80.69	87.50	91.03	93.02	94.67	69.31	79.40	86.58	90.49	92.95	94.71
<i>First-element Tie-breaking</i>		65.37	76.73	86.44	90.31	91.95	93.98	<b>70.00</b>	<b>79.55</b>	<b>87.42</b>	<b>91.41</b>	<b>94.28</b>	<b>95.72</b>
<i>Last-element Tie-breaking</i>		70.05	<b>81.46</b>	<b>89.18</b>	<b>91.67</b>	<b>93.26</b>	<b>94.96</b>	<b>70.00</b>	<b>79.55</b>	86.11	90.15	92.44	94.24
<i>Higher-strength Tie-breaking</i>		70.56	80.14	87.28	90.71	92.53	94.32	69.25	79.25	86.67	90.75	93.16	94.85
<i>Lower-strength Tie-breaking</i>		70.51	80.18	87.40	90.98	92.91	94.63	68.79	78.86	86.34	90.13	92.47	94.33

studies will be conducted in the future using more real-life programs, more interaction test suite construction tools, and more prioritization algorithms of interaction test suites.

Another threat is the evaluation of experimental results – a metric named APFD or NAPFD was used to evaluate the rates of fault detection for different tie-breaking techniques. The NAPFD metric is commonly used in the study of test case prioritization.

## V. CONCLUSIONS AND FUTURE WORK

The prioritization of interaction test suites has been widely studied in recent years, especially when testing resources are limited. During the process of prioritization of interaction test suites, the corresponding algorithm may face a challenge that there may exist more than one “best” candidate, how to choose it from them (or do tie-breaking)? In this paper, we investigate different techniques used to choose the next test case from “best” candidates, including random tie-breaking, first-element tie-breaking, last-element tie-breaking, higher-strength tie-breaking, and lower-strength tie-breaking. An empirical study shows that random tie-breaking and last-element tie-breaking would be the better choices than other tie-breaking techniques for testers.

This study only investigates different tie-breaking techniques on the certain prioritization strategy (that is, interaction coverage based prioritization), it is necessary to apply these tie-breaking techniques to other prioritization strategies of interaction test suites. We will study it in the future.

## VI. ACKNOWLEDGMENTS

We would like to thank D. R. Kuhn for providing us the ACTS tool, and the Software-artifact Infrastructure Repository (SIR) [13] which provided the source code and fault data for the subject program (`flex`). This work is in part supported by the National Natural Science Foundation of China (Grant No. 61202110), the Natural Science Foundation of Jiangsu Province (Grant No. BK2012284), and the Senior Personnel Scientific Research Foundation of Jiangsu University (Grant No. 14JDG039).

## REFERENCES

- [1] C. Nie and H. Leung, “A survey of combinatorial testing,” *ACM Computer Survey*, vol. 43, no. 2, pp. 11:1–11:29, January 2011.
- [2] R. Huang, J. Chen, Z. Li, R. Wang, and Y. Lu, “Adaptive random prioritization for interaction test suites,” in *Proceedings of the 29th Symposium On Applied Computing (SAC’14)*, 2014, pp. 1058–1063.
- [3] R. C. Bryce and C. J. Colbourn, “Test prioritization for pairwise interaction coverage,” in *Proceedings of the 1st International Workshop on Advances in Model-based Testing (A-MOST’05)*, 2005, pp. 1–7.
- [4] —, “Prioritized interaction testing for pairwise coverage with seeding and constraints,” *Information and Software Technology*, vol. 48, no. 10, pp. 960–970, October 2006.
- [5] R. C. Bryce and A. M. Memon, “Test suite prioritization by interaction coverage,” in *Proceedings of the Workshop on Domain Specific Approaches to Software Test Automation (DoSTA’07)*, 2007, pp. 1–7.
- [6] X. Qu, M. B. Cohen, and K. M. Woolf, “Combinatorial interaction regression testing: A study of test case generation and prioritization,” in *Proceedings of the 23rd International Conference on Software Maintenance (ICSM’07)*, 2007, pp. 255–264.
- [7] Z. Wang, L. Chen, B. Xu, and Y. Huang, “Cost-cognizant combinatorial test case prioritization,” *International Journal of Software Engineering and Knowledge Engineering*, vol. 21, no. 6, pp. 829–854, September 2011.
- [8] J. Petke, S. Yoo, M. B. Cohen, and M. Harman, “Efficiency and early fault detection with lower and higher strength combinatorial interaction testing,” in *Proceedings of the 12th Joint Meeting on European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE’13)*, 2013, pp. 26–36.
- [9] R. Huang, J. Chen, T. Zhang, R. Wang, and Y. Lu, “Prioritizing variable-strength covering array,” in *Proceedings of the IEEE 37th Annual Computer Software and Applications Conference (COMPSAC’13)*, 2013, pp. 502–601.
- [10] R. Huang, X. Xie, D. Towey, T. Y. Chen, Y. Lu, and J. Chen, “Prioritization of combinatorial test cases by incremental interaction coverage,” *International Journal of Software Engineering and Knowledge Engineering*, vol. 23, no. 10, pp. 1427–1457, 2014.
- [11] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, “Prioritizing test cases for regression testing,” *IEEE Transactions on Software Engineering*, vol. 27, no. 10, pp. 929–948, October 2001.
- [12] R. Huang, X. Xie, T. Y. Chen, and Y. Lu, “Adaptive random test case generation for combinatorial testing,” in *Proceedings of the IEEE 36th Annual Computer Software and Applications Conference (COMPSAC’12)*, 2012, pp. 52–61.
- [13] H. Do, S. G. Elbaum, and G. Rothermel, “Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact,” *Empirical Software Engineering*, vol. 10, no. 4, pp. 405–435, 2005.
- [14] Y. Lei, R. Kacker, D. R. Kuhn, and V. Okun, “I-pog/ipod: Efficient test generation for multi-way software testing,” *Software Testing, Verification, and Reliability*, vol. 18, no. 3, pp. 125–148, 2008.
- [15] J. Czerwonka, “Pairwise testing in real world: Practical extensions to test case generators,” in *Proceedings of the 24th Pacific Northwest Software Quality Conference (PNSQC’06)*, 2006, pp. 419–430.

# An Application of Adaptive Random Sequence in Test Case Prioritization

Xiaofang Zhang

School of Computer Science and  
Technology  
Soochow University  
Suzhou, China  
xfzhang@suda.edu.cn

Tsong Yueh Chen

Department of Computer Science  
and Software Engineering  
Swinburne University of Technology  
Hawthorn, Australia  
tychen@swin.edu.au

Huai Liu

Australia-India Research Centre for  
Automation Software Engineering  
RMIT University  
Melbourne, Australia  
huai.liu@rmit.edu.au

**Abstract**—Test case prioritization aims to schedule test cases in a certain order such that the effectiveness of regression testing can be improved. Prioritization using random sequence is a basic and simple technique, and normally acts as a benchmark to evaluate other prioritization techniques. Adaptive Random Sequence (ARS) makes use of extra information to improve the diversity of random sequence. Some researchers have proposed prioritization techniques using ARS with white-box code coverage information that is normally related to the test execution history of previous versions. In this paper, we propose several ARS-based prioritization techniques using black-box information. The proposed techniques schedule test cases based on the string distances of the input data, without referring to the execution history. Our experimental studies show that these new techniques deliver higher fault-detection effectiveness than random prioritization. In addition, as compared with an existing black-box prioritization technique, the new techniques have similar fault-detection effectiveness but much lower computation overhead, and thus are more cost-effective.

**Keywords** – test case prioritization; adaptive random sequence; random sequence; random testing; adaptive random testing

## I. INTRODUCTION

Regression testing is used to ensure that changes to the program do not negatively impact its correctness. A main task in regression testing is the prioritization of test cases, which reorders the existing test cases to meet some performance goal, such as detecting the faults as early as possible. Previous studies on test case prioritization aim at providing earlier feedback to testers, and allow them to begin debugging earlier.

Various test case prioritization techniques have been developed, which are designed to achieve different objectives. Most of the test case prioritization techniques require white-box information or test history to facilitate their prioritization operations. Essentially, they use the information derived from the previous versions, such as program source code coverage or fault detection history. However, the white-box information and test history are not always available, and sometimes such a kind of information is incomplete or inaccurate, or even costly or difficult to obtain.

To address this problem, Ledru et al. [1] propose a prioritization technique based on string distances between test cases, which does not depend on the availability of white-box

information or test history. Their technique is based on the concept of test case diversity in terms of four classic string distance metrics. However, their prioritization technique needs to calculate the distances for each pair of test cases in the given test suite. As a consequence, the computation overhead is considerably expensive.

Our study aims to reduce the high computation overhead while preserving the test case diversity in the prioritized sequence. We investigate a more cost-effective test case prioritization methodology using the black-box information of the program under test. Our prioritization techniques make use of the concept of adaptive random sequence to achieve diversity of test cases through the notion of evenly spreading across the input domain.

The rest of the paper is organized as follows. Section II introduces the background information of test case prioritization and adaptive random sequence. The previous work related to our study is discussed in Section III. Section IV gives the details of our new prioritization techniques. Section V reports our empirical study for evaluating the new techniques. The paper is finally summarized in Section VI.

## II. BACKGROUND

### A. Test Case Prioritization Strategies

Test case prioritization schedules test cases so that those with the higher priority, according to some criterion, are executed earlier in the regression testing process. Given a test suite, test case prioritization will find a permutation of the original test suite, aiming to maximize the objective function. There are various strategies based on different intuitions. For example, history-based prioritization techniques use information from previous executions to determine test priorities; knowledge-based techniques use human knowledge to determine test priorities and model-based techniques use a model of the system to determine test priorities.

To measure the performance of different prioritization strategies, APFD has been proposed to measure the weighted average of the percentage of faults detected during the execution of the test suite. Let  $T$  be a test suite which contains  $n$  test cases, and let  $F$  be a set of  $m$  faults revealed by  $T$ . Let  $T'$  be the prioritized test sequence for  $T$ . Let  $TF_i$  be the sequence

index of the first test case in  $T'$  which reveals fault  $i$ . The APFD for test suite  $T'$  could be given by the following equation:

$$APFD = 1 - \frac{TF_1 + TF_2 + \dots + TF_m}{nm} + \frac{1}{2n} \quad (1)$$

Among all prioritization strategies, random sequence is the most simple and basic strategy. It is simple in concept and is easy to apply even when source code, specification or test history is unavailable or incomplete. Therefore, random sequence has been used as a benchmark for evaluating other prioritization strategies.

### B. Adaptive Random Sequence

Adaptive Random Sequence (ARS), originated from the concept of Adaptive Random Testing (ART) [2], is basically a random sequence embedding the notion of diversity. ARS has been argued as a possible alternative to random sequence.

ART is aimed to improve the fault-detection effectiveness of random testing through the concept of even spreading of test cases in the input domain [2]. It is motivated by the empirical observation that failure-causing inputs are frequently clustered into contiguous failure regions. In other words, if a test case is found to be non-failure-causing, it is very likely that its neighbors will not reveal any failures. Thus, preference should be given to select the input far away from the non-failure-causing inputs as the next test case. ART can be implemented using various notions of even spread, such as Fixed Size Candidate Set ART(FSCS-ART) [2], restricted random testing [3], ART by dynamic partitioning [4], lattice-based ART [5], and so on. In order to reduce the generation overhead for these algorithms, some general reduction techniques have been developed, such as clustering [6], mirroring [7], and forgetting [8]. Since its inception, ART has been applied into many different types of programs [6, 9].

There is a close relationship between ARS and ART. In the context of test case generation, ART is a test case generator to select test cases from the pool of possible inputs. However, if ART exhausts the whole pool of possible inputs, the generated order can be treated as a prioritized order. It means that ART can be used as a test suite ‘‘prioritizer’’ to deliver a prioritized sequence. Technically speaking, the test sequence generated by ART is an adaptive random sequence (ARS), which embeds the concept of even spread across the input domain. Moreover, ART is based on random testing with the objective of revealing the failures as early as possible, which is consistent with the objective of prioritization. As a consequence, ARS can be applied in test case prioritization.

### III. RELATED WORK

Test case prioritization schedules test cases with an intention to achieve some performance goal. Various test case prioritization techniques have been proposed using different intuitions.

Among these techniques, Rothermel et al. [10] emphasized using execution information acquired in previous test runs to define test case priority and they defined various techniques. Their techniques are shown to be effective at achieving higher values for APFD. Furthermore, Li et al. [11] proposed several

non-greedy algorithms, including hill climbing algorithm and genetic algorithm. Evidently, all of these prioritizations require the test history information of the previous versions.

Similar to our investigation of test case prioritization by ARS, Jiang et al. [12] and Zhou et al. [13] proposed the family of adaptive random test case prioritization using code coverage. They used Jaccard distance and Manhattan distance respectively, to measure the difference of code coverage. The empirical results showed that they are statistically superior to the random sequence in detecting faults. Furthermore, Zhou et al. [14] applied ART to prioritize test cases based on execution frequency profiles using frequency Manhattan distance. Recently, Fang et al. [15] proposed a similarity-based test case prioritization technique based on farthest-first ordered sequence, which is similar to adaptive random sequence. However, these white-box methods assume the availability of certain coverage information or execution frequency profiles.

Ledru’s prioritization technique used string distances to measure the test case diversity, and hence solely depended on the black-box information [1]. However, their algorithm computes the distances for each pair of test cases to find the first test case with the maximum distance, and then it repeatedly chooses a test case which is most distant from the set of already ordered test cases. Therefore, their prioritized test sequence is deterministic but incurs expensive overhead.

### IV. ARS-BASED TEST CASE PRIORITIZATION

In this section, we present our offline ARS-based test case prioritization algorithm, denoted as ARS-all, and online ARS-based test case prioritization algorithm, denoted as ARS-pass, respectively.

Before presenting our prioritization algorithms, we first give some definitions. Suppose  $T = \{t_1, t_2, \dots, t_n\}$  is a regression test suite with  $n$  test cases. A test sequence  $PT$  is an ordered list of test cases. If  $t$  is a test case, and  $PT = \langle p_1, p_2, \dots, p_k \rangle$ , we define  $PT \hat{\ } t$  to be  $\langle p_1, p_2, \dots, p_k, t \rangle$ .

#### A. Offline Prioritization Algorithm: ARS-all

Majority of the existing test case prioritization techniques are applied offline. That is, after the prioritization is completed, the test case sequence is finalized, and then the regression testing is conducted according to the prioritized test cases until testing resources exhaust.

Our offline prioritization algorithm ARS-all can be done in a batch-mode as other existing test case prioritization techniques. During the prioritization, two sets of test cases are maintained. One set is the *already prioritized set*  $P$ , the other set is the *not-yet-prioritized set*  $NP$ . Obviously,  $T = NP \cup P$ .

Our algorithm ARS-all aims at selecting a test case farthest away from all already prioritized test cases, which is summarized in Fig. 1: Initially, we randomly select a test case from  $T$ . Then, we use FSCS-ART algorithm to decide the next test case. The algorithm constructs the *candidate set*  $CS$  by randomly select  $k$  test cases from  $NP$ , and then a candidate from  $CS$  will be selected as the next test case if it has the longest distance to its nearest neighbor in  $P$ . The process is repeated until all the test cases are ordered in sequence. In this paper, let  $k=10$  according to the previous studies [2].

## V. EMPIRICAL STUDY

In this section, empirical evaluations of real-life program are presented to assess the performance of our algorithms.

### A. Peer Techniques for Comparison

Besides random sequence, we compare our prioritization techniques with the technique proposed by Ledru et al. [1], denoted as *Ledru*.

In this study, two classic distance metrics, namely Manhattan and Hamming distances, are used to measure the diversity of test cases. Given two test cases represented by strings  $A$  and  $B$ , the Manhattan distance between them is calculated as  $\sum_{i=1}^n |\alpha_i - \beta_i|$ , where  $\alpha_i$  and  $\beta_i$  denote the ASCII code for each character in  $A$  and  $B$ , respectively, and  $n$  is the longer length of  $A$  and  $B$ . When  $A$  and  $B$  have different lengths, the shorter one will be filled with NULL characters, whose ASCII code value is 0. For example, given  $A = \text{"ac"}$  and  $B = \text{"abd"}$ , the Manhattan distance between  $A$  and  $B$  is  $|97-97| + |99-98| + |0-100| = 101$ . Manhattan distance was recommended as the best choice for the Ledru prioritization technique [1]. Hamming distance is the simplest distance metric. Given two test cases  $A$  and  $B$ , the Hamming distance between them is calculated as the number of characters different in these strings. When  $A$  and  $B$  have different lengths, the shorter string will be completed by a list of NULL characters in order to have the same length of the longer string. For the previous example  $A = \text{"ac"}$  and  $B = \text{"abd"}$ , their Hamming distance is 2.

Combining different prioritization algorithms and distance metrics, we have seven prioritization strategies as follows, summarized in Table I.

TABLE I. PRIORITIZATION TECHNIQUES

Ref.	Name	Algorithm	Distance
T1	Random	Random	--
T2	Ledru-M	Ledru	Manhattan
T3	Ledru-H	Ledru	Hamming
T4	ARS-all-M	ARS-all	Manhattan
T5	ARS-all-H	ARS-all	Hamming
T6	ARS-pass-M	ARS-pass	Manhattan
T7	ARS-pass-H	ARS-pass	Hamming

### B. Research Questions

We study the following research questions in the empirical study.

**RQ1:** Do ARS-based techniques perform better than prioritization using random sequence?

**RQ2:** Are ARS-based techniques more cost-effective than Ledru technique?

### C. Subject Program and Test Suites

#### Algorithm: ARS-all

**Inputs:**  $T: \{t_1, t_2, \dots\}$  is a set of test cases  
 $k$ : the number of candidates

**Output:**  $PT: \langle p_1, p_2, \dots \rangle$  is a sequence of test cases

1. Initialize:  $PT \leftarrow \phi, P \leftarrow \phi, NP \leftarrow \phi, CS \leftarrow \phi$
2.  $t \leftarrow$  a test case randomly selected from  $T$
3. **do**
4.     Initialize:  $CS \leftarrow \phi$
5.      $CS \leftarrow$  randomly select  $k$  test cases from  $NP$
6.     **for** each candidate test cases  $t_j$ , where  $j=1,2,\dots,k$
7.         calculate its distance  $d_j$  to its nearest neighbor in  $P$
8.     **end for**
9.     find  $t_b \in CS$  such that  $\forall j=1,2,\dots,k, d_b \geq d_j$
10.     $t \leftarrow t_b$
11.     $PT \leftarrow PT \cup t$
12.     $P \leftarrow P \cup \{t\}$
13.     $NP \leftarrow T - P$
14. **while** ( $NP \neq \phi$ )
15. return  $PT$

Figure 1. The prioritization algorithm: ARS-all

### B. Online Prioritization Algorithm: ARS-pass

Different from most existing prioritization techniques that are applied in a batch mode, ARS-pass requires online feedback information. Technically speaking, the next prioritized test case depends on the previous execution results of the current version. Hence, the prioritization must be conducted interleaving with program execution.

Suppose that previously prioritized and executed test cases have not revealed any failures. The next prioritized test case should be far away from them, aiming at revealing failures more quickly. Thus, the prioritized test case sequence is generated according to the results of executed test cases.

Our prioritization algorithm ARS-pass can be easily implemented using the concept of forgetting, which is also referred to as ART with selective memory [16]. The subset of  $P$  will only memorize the non-failure-causing test cases using the online feedback information of program execution. By forgetting the failure-causing test cases and only focusing on the passed test cases, ARS-pass not only reduces the distance computation overhead, but also ensures that each new test case is far away from all already executed but non-failure-causing test cases. As a consequence, the probability of failure detection will be increased.

Note that we have proposed ARS-all and ARS-pass to reduce the prioritization overhead, and they can be applied in different execution modes. Particularly, ARS-pass will further reduce the computation overhead and improve the diversity of test cases by using the online feedback execution information of the program version under test.

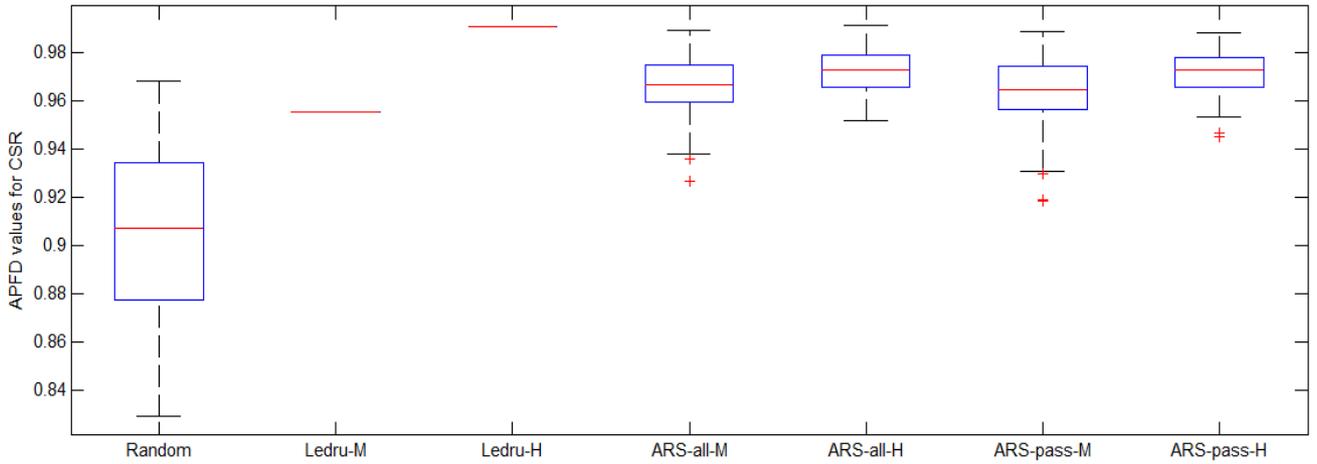


Figure 2. APFD values for test suite CSR

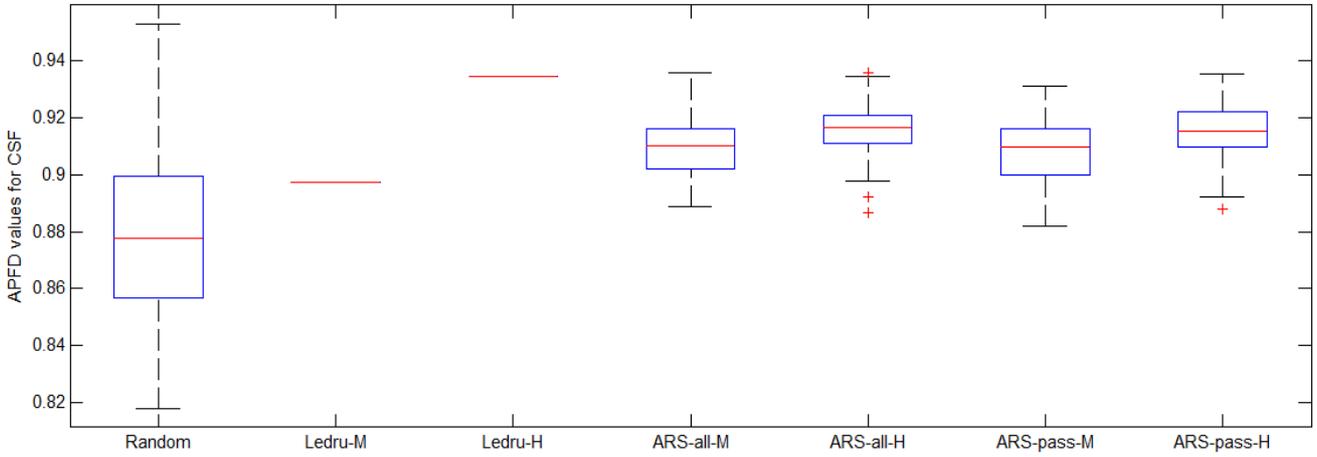


Figure 3. APFD values for test suite CSF

TABLE II. INFORMATION OF SUBJECT PROGRAM AND TEST SUITES

Application	<i>Crossword Sage</i>	
Version	0.3.3	
#Widgets	80	
#Faults	14	
#Test cases	RT	738
	FT	1278
#Detected faults	RT	12
	FT	14

Note: RT refers to Random Testing;  
FT refers to Functional Testing.

We use a GUI application *Crossword Sage* as subject program. This subject program is free and open source in SourceForge. It presents unambiguous graphical user interfaces for testers to design the test cases. This application, written in Java, was widely used in some other studies [17-19].

Furthermore, this subject program contains real-life faults and exceptions.

Yang et al. [17] conducted an empirical study to compare random testing and functional testing in GUI testing. Thousands of random and functional test cases, in the form of Java scripts, had been created for this GUI application. With some necessary transformations, these existing test suites will be used for our prioritization. The basic information of our subject program and test suites is shown in Table II, and the details can be found in [17].

#### D. Experiment Design

For each prioritization strategy, we prioritize the existing two test suites, that is, *Crossword Sage Random* test suite and *Crossword Sage Functional* test suite, denoted as CSR and CSF, respectively. Thus, there are two sets of results for each prioritization strategy. The results include the prioritized sequence, the APFD value, the prioritization time cost, the detected faults report and so on. Since some of the prioritization strategies involve random selection, we repeat each of the prioritization strategies 100 times to obtain the averages.

### E. Analysis of the Results

In this section, we analyze the experiment results to answer the research questions. As shown in Fig. 2 and Fig. 3, we calculate the APFD values for each strategy and draw box-and-whisker plots for the two test suites, respectively. For each box-and-whisker plot, the  $x$ -axis represents prioritization strategies and the  $y$ -axis represents their APFD values. The horizontal lines in the boxes indicate the lower quartile, median and upper quartile values. Furthermore, we present the average prioritization time cost (in milliseconds) of 100 trials for each strategy, summarized in Table III.

To address **RQ1**, we compare the APFD values between our ARS-based strategies and random sequence.

From Fig. 2 and Fig. 3, we observe that, for test suites CSR and CSF, all of our ARS-based strategies outperform the random sequence. We further conduct binomial t-tests for the APFD values for our ARS-based strategies and random sequence respectively. As shown in Table IV, all  $p$ -values are smaller than the significant level of 0.05. It means that all the null hypotheses ( $H_0$ : ARS-all-M/ ARS-all-H/ ARS-pass-M/ ARS-pass-H does not outperform Random) are rejected.

As a result, it is statistically significant that all of our ARS-based techniques perform better than random prioritization with respect to the APFD values.

TABLE III. INFORMATION OF PRIORITIZATION TIME (IN MILLISECONDS)

Technique	CSR	CSF
Random	0.20	0.34
Ledru-M	1632.67	10014.33
Ledru-H	1581.33	9480.00
ARS-all-M	56.90	194.51
ARS-all-H	58.76	194.67
ARS-pass-M	26.97	122.88
ARS-pass-H	29.08	120.14

To address **RQ2**, we compare the APFD values and prioritization time cost between ARS-based strategies and Ledru technique using these two test suites.

From Fig. 2, for test suite CSR, we observe that, with Manhattan distance, both ARS-all and ARS-pass significantly outperform Ledru. Whereas, for Hamming distance, Ledru-H has the best APFD value. Similar results also exist for CSF, as shown in Fig. 3. However, the differences between their APFD values are less than 5%. In other words, the performance of ARS-all, ARS-pass, and Ledru technique can be said to be comparable with respect to the APFD values.

For prioritization time cost (refer to Table III), it is obvious that all of ARS-based strategies use much less time than Ledru technique. In the best case, the computation time for ARS-based technique is about 1.23% of that for Ledru's technique (ARS-pass-M: 122.88 vs. Ledru-M: 10014.33 in CSF), while in

the worst case the computation time for ARS-based technique is about 3.72% of that for Ledru's technique (ARS-pass-H: 58.76 vs. Ledru-H: 1581.33 in CSR). The reason for the dramatic cost reduction by our prioritization is that Ledru technique has to calculate all the distances of each pair of test cases before selecting next test case but not our ARS-based strategies.

Briefly speaking, given that our ARS-based techniques and Ledru technique have comparable APFD values, the former's lower overhead suggests that our ARS-based techniques are more cost-effective than Ledru technique.

TABLE IV. APFD COMPARISONS BETWEEN RANDOM SEQUENCE AND FOUR ARS-BASED TECHNIQUES

Algorithm(x)	Algorithm(y)	CSR		CSF	
		Mean Diff.(x-y)	Sig.	Mean Diff.(x-y)	Sig.
Random	ARS-all-M	-0.0611	0.0000	-0.0298	0.0000
Random	ARS-all-H	-0.0676	0.0000	-0.0367	0.0000
Random	ARS-pass-M	-0.0588	0.0000	-0.0285	0.0000
Random	ARS-pass-H	-0.0668	0.0000	-0.0356	0.0000

TABLE V. APFD COMPARISONS BETWEEN ARS-ALL AND ARS-PASS

Algorithm(x)	Algorithm(y)	CSR		CSF	
		Mean Diff.(x-y)	Sig.	Mean Diff.(x-y)	Sig.
ARS-all-M	ARS-pass-M	0.0016	0.2090	0.0013	0.3805
ARS-all-H	ARS-pass-H	0.0008	0.5531	0.0011	0.4136

We further compare the APFD values and prioritization time cost between ARS-all and ARS-pass. As shown in Fig. 2 and Fig. 3, ARS-all-M/H and ARS-pass-M/H have the similar APFD values. We conduct binomial t-tests of the APFD values for ARS-all and ARS-pass respectively, as shown in Table V. The online ARS-based algorithm, namely, ARS-pass, has comparable performance to the offline ARS-based algorithm, namely, ARS-all, in terms of the APFD values.

Next, we discuss their prioritization time cost. From Table III, we observe that, by making use of online feedback information of execution results and only focusing on the passed test cases, ARS-pass uses much less computation time. The computation time for ARS-pass is about 48% and 60% of that for ARS-all in CSR and CSF, respectively. Please note that, there are 302 passed test cases and 436 failed test cases in CSR while 749 passed test cases versus 529 failed test cases in CSF. In fact, the ratios of the time cost are consistent with the ratios of passed test cases in these two test suites.

In conclusion, ARS-pass has the comparable APFD values with ARS-all while using less computation time.

As explained in the above discussions, our adaptive random sequence outperforms random sequence and furthermore is cost-effective to apply in practice.

## F. Threats to Validity

The major threat to internal validity is the potential faults in our tools. We use Java to implement the tools for GUI test case transformation, distance calculation, prioritization algorithm, and results calculation. Actually, we have carefully tested our tools on small examples to assure correctness.

Threats to external validity correspond to the subject programs and test suites. For the space limit, although we have carried out some other case studies, we just present the empirical study results of one GUI application, which is open source and widely used in several research works. Moreover, the used two test suites cannot sufficiently represent the real-life situations. Further experiments on other subject programs and test suites in different types and languages may help generalize our findings.

The threat to construct validity involves the measurement. In this paper, we just use the most commonly adopted effectiveness metric in prioritization, namely, APFD to measure the effectiveness of a prioritized test suite. Other measures should probably be considered in further work.

There is little threat to conclusion validity. A large number of experimental runs have been executed to get reliable average APFD values for the prioritization strategies involving (adaptive) random sequences. Statistical tests were conducted to validate the statistical significance of our experimental results.

## VI. CONCLUSION AND FUTURE WORK

This paper reported the first attempt to use adaptive random sequence and black-box information in test case prioritization. It is easy to implement, and it just requires the information about inputs. We carried out the experiments on an open source GUI application. The results of experiments and statistical analyses provided clear evidence that our ARS-based techniques are cost-effective in prioritizing test suites. Their performances of APFD are better than that of the random sequence, which was used as a baseline. Moreover, our techniques are much more cost-effective than Ledru technique. The ARS-pass algorithm achieves a good balance between APFD effectiveness and efficiency by using the online feedback information.

In future, we will conduct more experiments on various subject programs and test suites with different types, sizes, languages and other attributes. Furthermore, other efficiency improvement techniques for ART and other types of distance calculation will be considered. Finally, adaptive random sequence with different objectives to conduct prioritization is a promising research topic.

## ACKNOWLEDGMENT

This work is partially supported by the National Natural Science Foundation of China (61103045), China Scholarship Council and Australian Research Council (DP 120104773). We would like to thank the authors of reference [17] to provide the test suites for our experiments.

## REFERENCES

- [1] Y. Ledru, A. Petrenko, S. Boroday and N. Mandran. "Prioritizing test cases with string distances," *Automated Software Engineering*, 19(1): 65-95, 2012.
- [2] T. Y. Chen, F.-C. Kuo, R. Merkel, and T. H. Tse. "Adaptive random testing: The ART of test case diversity," *Journal of Systems and Software*, 83(1): 60-66, 2010.
- [3] K. Chan, T. Y. Chen, and D. Towey. "Restricted random testing: Adaptive random testing by exclusion," *International Journal of Software Engineering and Knowledge Engineering*, 16(4):553-584, 2006.
- [4] T. Y. Chen, G. Eddy, R. Merkel, and P. Wong. "Adaptive random testing through dynamic partitioning," In *Proceedings of the 4th International Conference on Quality Software*, pp. 79-86, 2004.
- [5] J. Mayer. "Lattice-based adaptive random testing," In *Proceedings of the 20th International Conference on Automated Software Engineering*, pp. 333-336, 2005.
- [6] I. Ciupa, A. Leitner, M. Oriol, and B. Meyer. "ARTOO: Adaptive random testing for object-oriented software," In *Proceedings of the 30th International Conference on Software Engineering*, pp. 71-80, 2008.
- [7] T. Y. Chen, F. Kuo, R. Merkel, and S. Ng. "Mirror adaptive random testing," *Information and Software Technology*, 46(15): 1001-1010, 2004.
- [8] K. Chan, T. Y. Chen, and D. Towey. "Forgetting test cases," In *Proceedings of the 30th Annual International Computer Software and Applications Conference*, pp. 485-492, 2006.
- [9] Y. Lin, X. Tang, Y. Chen, and J. Zhao. "A divergence-oriented approach to adaptive random testing of Java programs," In *Proceedings of the 24th International Conference on Automated Software Engineering*, pp. 221-232, 2009.
- [10] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold. "Prioritizing Test Cases for Regression Testing," *IEEE Transactions on Software Engineering*, 27(10): 929-948, 2001.
- [11] Z. Li, M. Harman, and R. Hierons, "Search Algorithms for Regression Test Case Prioritization," *IEEE Transactions on Software Engineering*, 33(4): 225-237, 2007.
- [12] B. Jiang, Z. Zhang, W. K. Chan, and T. H. Tse. "Adaptive random test case prioritization," In *Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering*, pp. 233-244, 2009.
- [13] Z. Zhou. "Using coverage information to guide test case selection in adaptive random testing," In *Proceedings of the 34th Annual International Computer Software and Applications Conference, 7th International Workshop on Software Cybernetics*, pp. 208-213, 2010.
- [14] Z. Zhou, A. Sinaga, and W. Susilo. "On the Fault-Detection Capabilities of Adaptive Random Test Case Prioritization Case Studies with Large Test Suites," In *Proceedings of the 45th Hawaii International Conference on System Sciences*, pp. 5584-5593, 2012.
- [15] C. Fang, Z. Chen, K. Wu, and Z. Zhao. "Similarity-based test case prioritization using ordered sequences of program entities," accepted by *Software Quality Journal*, published online Nov. 2013. DOI: 10.1007/s1219-013-9224-0.
- [16] H. Liu, F. Kuo, and T. Y. Chen. "Comparison of adaptive random testing and random testing under various testing and debugging scenarios," *Software: Practice and Experience*, 42(8):1055-1074, 2012.
- [17] W. Yang, Z. Chen, Z. Gao, Y. Zou, and X. Xu. "GUI testing assisted by human knowledge: Random vs. functional," *Journal of Systems and Software*, 89(3):76-86, 2014.
- [18] A. Memon, M. Pollack, and M. Soffa. "Hierarchical GUI test case generation using automated planning," *IEEE Transactions on Software Engineering*, 27 (2):144-155, 2001.
- [19] A. Memon. "Automatically repairing event sequence-based GUI test suites for regression testing," *ACM Transactions on Software Engineering and Methodology*, 18 (2), 4:1-4:36, 2008.

# An Empirical Study on Inter-Commit Times in SVN

Qiuju Hou<sup>1</sup>, Yutao Ma<sup>3,4,\*</sup>, Jianxun Chen<sup>1,2</sup>, Youwei Xu<sup>5</sup>

1. College of Computer Science and Technology, Wuhan University of Science and Technology, Wuhan 430065, China
2. Hubei Provincial Key Laboratory of Intelligent Information Processing and Real-time Industrial System, Wuhan 430065, China
3. State Key Laboratory of Software Engineering, Wuhan University, Wuhan 430072, China
4. WISET Automation Co., Ltd., Wuhan Iron and Steel Group Corporation, Wuhan 430080, China
5. Information Center, Wuhan Housing Security and Management Bureau, Wuhan 430015, China

\*E-mail: ytma@whu.edu.cn

**Abstract**—Until now, centralized revision control systems such as Subversion (SVN) have been widely used in open-source software (OSS) development. Commit is a basic and important operation for revision control, and it has attracted the attention of a large number of researchers. As far as we know, few of prior studies investigated the distribution of inter-commit times (known as commit intervals), which reveals the development dynamics of an OSS project to some extent. To gain a better understanding on OSS development processes, we conducted an empirical study on two representative projects written in Java, and found that (1) the distributions of commit intervals in the two projects in question roughly follow power laws, with commit bursts (i.e., the revisions in a SVN repository are updated quickly over a period of time) and heavy tails, and (2) the working mode of SVN and (full-time or volunteer) developers' work habits contribute to commit bursts, while active committer's individual behavior (such as his/her tasks completion and illness) and long vacations are the primary factors that result in long inter-commit times. The findings could provide a new insight into schedule planning for OSS projects based on developers' historical commit behavior.

*Keywords*-subversion; development dynamics; commit interval; burst; heavy tail

## I. INTRODUCTION

Over the past two decades, open source as a new model has been deemed as a trend of effective software development to improve the function and quality of software [1]. In an open Internet-based environment, free and open source software (OSS) attracts the developers from all over the world to work together in a collaborative manner. For an OSS project with multiple developers, revision control (also known as version control) is an essential ingredient to software code management. Revision control systems such as CVS (Concurrent Versions System) and SVN (Subversion) are often centralized, with a single authoritative code repository, and check-outs and check-ins done with reference to such a central repository [2].

As we know, both centralized and decentralized revision control systems can track and provide control over changes to source code. Commit, also known as code contribution, is an important operation for these systems, since it tells a revision control system that a developer wants to make the change(s) final and available to all developers. In this sense, the development and maintenance process of an OSS project that is under centralized revision control could be regarded to be composed of a series of commits [3, 4]. The importance of mining historical commits of OSS projects is twofold: on one hand, it provides a new insight into the evolutionary aspects of an OSS project as well as its components [5, 6], which

contributes to a better understanding of OSS development and maintenance process [7]; on the other hand, it offers a feasible and reliable way to investigate how the cooperation among collective developers promotes OSS development [8, 9, 10], which may facilitate the organization of project teams [11].

To the best of our knowledge, a majority of prior studies on commit focus on commit size distribution as well as commit classification. The former describes the probability that a given commit is of a particular size in terms of the number of files, LOC (Lines of Code), or other measures [3, 7, 8, 12], while the latter classifies commits according to their features and relates a given commit to certain types of software activities such as code management and bug fixing [8, 13, 14, 15]. Until now, very little attention has been paid to exploring the impact of developers' collective commit behavior on OSS development and maintenance process using statistical methods [4, 10]. Actually, this is an important issue within the field of OSS research and practice [16].

In order to gain a deeper understanding of the interplay between developers' collective behavior and OSS development process, the main goal of this paper is to conduct an empirical study on modeling the development dynamics of OSS projects hosted by a SVN server from the perspective of developer's commit behavior. Moreover, we analyzed two representative projects written in Java on the Apache.org in an attempt to answer the following questions: 1) to what extent can the development dynamics of an OSS project be modeled in terms of developers' commit behavior, and 2) if such a model does exist, what factors may affect its primary (statistical) features? We hope our empirical findings could offer a better understanding of OSS development and maintenance process, as well as novel ideas for the solution to the above-mention issues.

The remainder of this paper is structured as follows. Section II introduces related work. Section III addresses research questions, and explains the experimental methods we followed. Section IV presents primary results, and discusses the implications of our findings. Finally, Section V concludes this paper and puts forward future work.

## II. RELATED WORK

A commit is a basic unit of work performed by a developer. Previous studies about commit size distribution found that the distributions of commit size in terms of specific measures roughly followed power laws [3, 7, 8, 12], implying that large-sized commits do exist, though they are less likely to occur. Meanwhile some of researchers began to categorize commits according to their features such as size and comment (also

known as log message) [8, 13, 14, 15], and found that the category of commit size was able to be a sound indicator for the types of maintenance activities being performed [14]. For example, Hindle *et al.* [15] found that large-sized commits were more perfective while small-sized commits were more corrective. However, none of these studies took the dynamics of developer’s commit behavior into account.

As mentioned before, the development of an OSS project could be deemed as a collaborative process of developers’ collective commit behavior [4, 10]. Human behavior, as one of the significant issues in science, has a history of about one century since the time of Watson [17]. Based on the increasing evidence from communication to entertainment and work patterns, Barabási *et al.* found that the timing of many human activities within these fields followed non-Poisson statistics, characterized by bursts of rapidly occurring events separated by long periods of inactivity [18]. Interestingly, such heavy-tailed distributions of inter-event times have also been demonstrated in computer science [19], e.g., email communication, website access, instant messaging, and Linux command logs. Recently, only a few of researchers began to investigate such a problem in terms of mining historical commits in OSS repositories [4, 5, 10, 11], and their work laid a good foundation for this paper.

### III. RESEARCH QUESTIONS AND EXPERIMENTAL METHODS

It is worth noting that this paper uses the GQM (Goal, Question, Metric) method (<http://en.wikipedia.org/wiki/GQM>) to carry out an empirical investigation. In order to accomplish the research goal, research questions, the metrics associated with each question, and the experimental methods we used are described in detail in this section.

#### A. Research Questions

**RQ1:** *to what extent can the development dynamics of an OSS project be modeled?*

For an OSS project that is under centralized revision control, its (central) repository is indeed a special kind of file server, which can record the history of changes to every file. The development and maintenance process of the project is actually an overall picture of what has been happening in the repository [4]. Thus, the goal of the first research question is to investigate how we model the changes in the repository based on the metrics for developers’ commit behavior. Moreover, if such a model does exist, what are its primary (statistical) features? We argue this would provide a better understanding of general laws for the development of an OSS project.

**RQ2:** *what factors may affect the primary features of the model, and what are implications for OSS development process?*

Intuitively speaking, statistical features of the model obtained from *RQ1* may be determined by those factors that affect developers’ commit behavior, e.g., work pattern, team organizational structure, as well as developer’s social properties such as nationality, occupation, hobbies and geographical location. For example, Singh found that the small-world developer collaboration networks positively affected the productivity of the member developers of an OSS project [20]. So, the goal of the second research question is to explore the factors influencing the model and their implications for OSS development process. We believe that this may facilitate a more reasonable schedule planning based on historical commits.

#### B. Metrics associated with the research questions

Each time the repository of an OSS project accepts a commit submitted by a SVN client, a new revision will be created to represent a “snapshot” of the entire repository tree instead of an individual file, with a unique natural number. The initial revision number is 0, and each subsequent commit successfully accepted by the repository increases the revision number by one [21]. Therefore, the history of revision logs sorted by the revision number in ascending order reveals the development process of the project in question in essence. To answer the above research questions, we make use of the following metrics for developer’s commit behavior.

**Definition 1.** For a project, *commit interval* is the time difference between two consecutive revisions in the repository that hosts the project [4].

**Definition 2.** For a project, *commit frequency* is the number of new revisions created in the repository in a given period of time [5, 10].

In general, commit frequency measures how often developers commit changes to the project they are in charge of, while commit interval describes how long a project will receive a new commit by a developer. The former is roughly in inverse proportion to the latter. That is to say, the smaller an average of commit intervals in a given period of time is, the greater the corresponding commit frequency becomes, indicating that the project under discussion is more active.

#### C. Experimental Methods

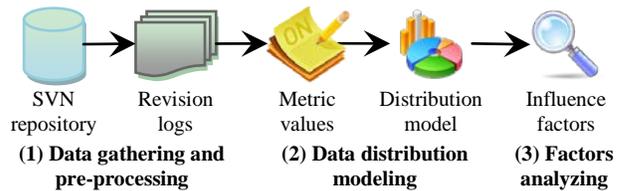


Figure 1. The process of our experiment

The overall process of our experiment is shown in Figure 1. Firstly, we extracted all historical revisions of a given project from its repository with a SVN client, and sorted them by the number in ascending order. Secondly, we calculated each commit interval of every pair of two consecutive revisions, and modeled the distribution of commit intervals by curve fitting methods. Thirdly, we sought the factors that may affect the primary (statistical) features of the model obtained in the above step, and presented their implications for OSS development.

In the second step, the distribution of commit intervals describes the probability that a given interval between two consecutive commits is of a particular length in terms of units of time. In probability theory and statistics, such discrete (probability) distributions can be represented in terms of probability mass functions or cumulative distribution functions (CDFs). Because the visual form of the CDF is, generally speaking, more robust than that of the probability distribution function against fluctuations due to finite sample sizes, especially in the tail of the distribution [22], in this paper we utilized the CDF and estimated the power-law exponent with the method in [22], so as to reduce noise levels. Besides power-law function, other common functions such as exponential function were also used to fit the discrete data in our data set.

In the third step, in order to determine the factors that may affect the distribution of commit intervals, we conducted an experiment to calculate the differences among each day of a week, as well as the difference between weekdays and weekend, with regard to commit interval and commit frequency. As we know, the median can be used as a measure of location when a distribution is skewed, when end-values are not known. Generally, in order to estimate the difference between two populations, a simple way is to compare their medians rather than means by using the standardized box plot. Moreover, in statistics, the Mann-Whitney  $U$  test [23] is a non-parametric test of the *null hypothesis* that two populations are the same. If a particular population tends to have larger values than the other, the *null hypothesis* is rejected, suggesting that their difference is statistically significant.

#### IV. PRIMARY RESULTS AND DISCUSSION

##### A. Data Collection

In this paper, our analysis is based on case studies, and we selected two OSS projects written in Java from the Apache.org, namely, Apache POI and Tomcat. Apache POI is used to create and maintain Java Application Programming Interfaces (APIs) for manipulating various file formats. Tomcat is an open source web server and servlet container developed by the Apache Software Foundation (ASF). These two projects were selected in that they are from different application domains, and each one is long-lived and active. Table I shows a brief introduction to the two representative projects, including the numbers of class files, commits and committers.

TABLE I. BRIEF INTRODUCTION TO THE PROJECTS ANALYZED

Project	Description	Class	Commit	Committer
POI	APIs for file processing	2655	2147	14
Tomcat	Servlet container	2595	9048	19

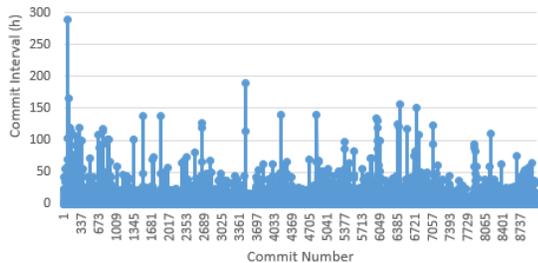


Figure 2. Line chart of commit intervals in Tomcat

Note that, we mined just a few of the historical revisions from their SVN repositories between January 1, 2009 and September 20, 2013, and then calculated all intervals between every pair of two consecutive commits. For example, the line chart of commit intervals in Tomcat is presented in Figure 2, characterized by bursts of rapidly occurring commits separated by long periods of inactivity (more than 150 hours).

##### B. Result for RQ1: modeling the development dynamics of an OSS project in terms of commit interval

The log-log scatter plot of the distributions of commit intervals in Tomcat and POI is shown in Figure 3, where X axis represents the length of a commit interval in hours and Y axis indicates the probability that a given commit interval

takes on a value less than or equal to  $x$ . Actually, the probability was calculated as the ratio of the number of commit intervals whose values are not greater than  $x$  to the total number of commit intervals under discussion. It is obvious from Figure 3 that the distributions of commit intervals in the two projects are heavy-tailed, implying that long inter-commit times do exist, though most of intervals are short, for example, about 80% of commit intervals in Tomcat are less than one hour.

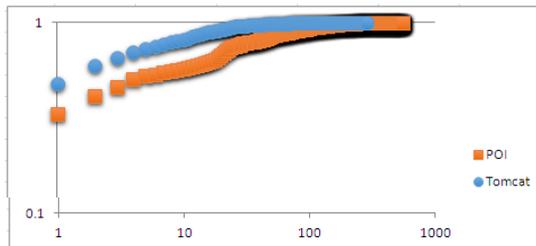


Figure 3. Log-log plot of the distributions of intervals in both projects

Then, we made a careful analysis of (curve) fitting functions for the distributions of commit intervals (see Figure 3), shown in Table II (where  $R^2$  is the goodness of fit). The fitting statistics indicate that the distributions in both of the two projects are best fitted by power functions, which suggests that they roughly follow power-law distributions. What we found reveals that a SVN repository often receives commits quickly over a period of time (called *commit burst*) except for a small number of ones for special reasons. The occurrence of commit bursts is mainly due to the working mode of centralized revision control [4] and developers' work habits. That is, a central SVN repository stores only the latest version of each file, so that the changes to HEAD of the trunk committed by different developers are always completed in a short time to ensure that everyone is working on the up-to-date files. In the following sub-section, we will discuss the factors that contribute to long periods of inactivity.

TABLE II. FITTING FUNCTIONS FOR DISTRIBUTION OF INTERVALS

Projects	Fitting functions	R2
Poi	$y = 0.0009902x + 0.8018$	0.5367
	$y = -7.404e - 06x^2 + 0.003088x + 0.7086$	0.8157
	$y = 0.06774 \ln(x) + 0.6064$	0.8469
	$y = 0.7975e^{0.000851x}$	0.4992
Tomcat	$y = 0.5583x^{0.1118}$	<b>0.9254</b>
	$y = 0.001712x + 0.8732$	0.4886
	$y = -5.202e - 05x^2 + 0.006887x + 0.7865$	0.7853
	$y = 0.3833 \ln(x) + 0.806$	0.6731
	$y = 0.8785e^{0.000725x}$	0.4716
	$y = 0.7284x^{0.07512}$	<b>0.8568</b>

*Finding: The distributions of commit intervals in the two OSS projects in question can be best modeled by power laws, with commit bursts and heavy tails.*

##### C. Result for RQ2: what may cause long periods of inactivity and their implications for software development

Although our prior work [4] conjectured that the delivery of a new release leads to a long inter-commit time, there is no evidence to support such a hypothesis. Intuitively speaking, this

is indeed one of the factors that cause long periods of inactivity. Besides the delivery of a new release, we guess weekend, holiday and active developer’s behavior also result in the occurrence of long commit intervals. Then, we are going to examine the influence of these factors on the length of commit intervals one after another, and then determine which factors are the primary ones.

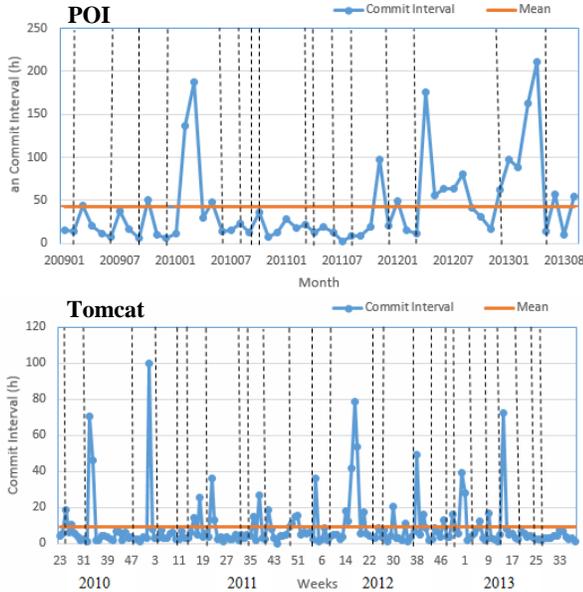


Figure 4. Changes of average commit intervals over time

Firstly, the changes of average commit intervals per time unit over time are presented in Figure 4, where each vertical dashed line indicate the date that a new version of the project was released, extracted from the history of changes (also known as changes log) on the homepage of each project under discussion. Considering the different level of activity of the two projects, we selected month and week for POI and Tomcat respectively on purpose. It is obvious that there are several abnormal points that represent large average commit intervals in Figure 4. Thus, we carefully examined these anomalies. To our surprise, a small number of them (in POI) are caused by the delivery of a new release, but others are not.

As shown in Figure 5, for POI there does exist a period of inactivity that exceeds 100 hours after a new version was released. This implies that, on one hand, the delivery of a new release of an active project (such as Tomcat) has little effect on developers’ normal commits; on the other hand, it is not the primary factor that results in very long inter-commit times.

Secondly, we tested whether weekends lead to long commit intervals. Because the average interval of POI is over 48 hours (i.e., the length), we just conducted an experiment on Tomcat. The standardized box plots of commit frequency and commit interval are presented in Figure 6. According to the comparison among the median of commit frequency and commit interval in each day of a week, the difference on commit frequency between weekend and weekdays is rather obvious, while the median of commit intervals on Sunday is the smallest in a week. Note that (1) we ignored the days without a commit or with only one commit, and (2) we excluded any pair of consecutive commits that don’t happen in the same day.

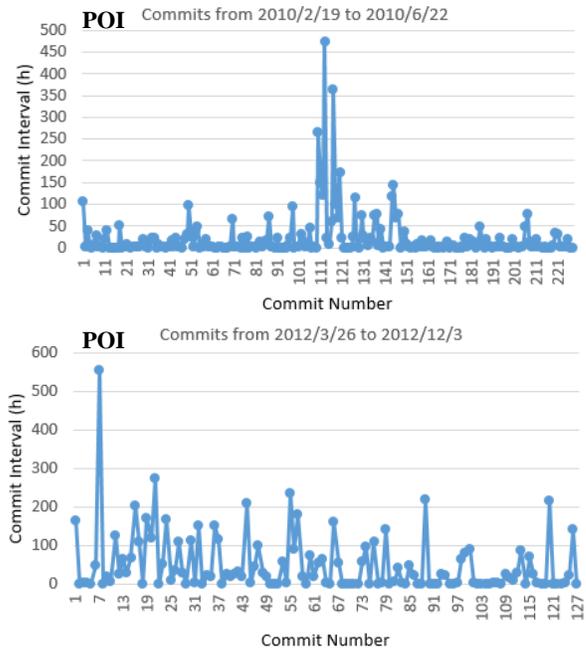


Figure 5. Inter-commit time over 100 hours caused by a new release

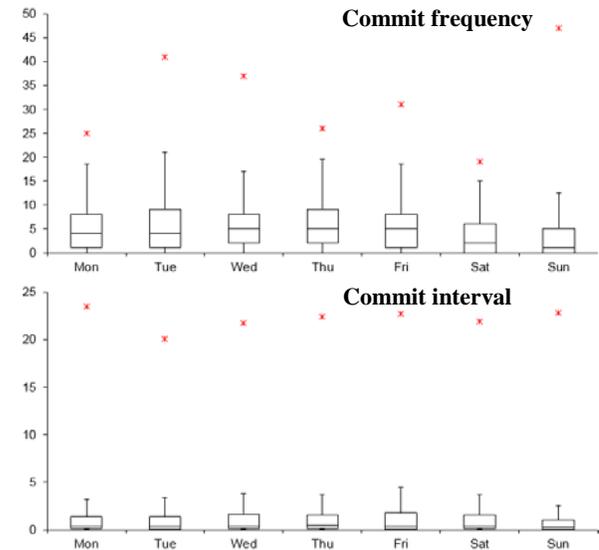


Figure 6. Standardized box plots for each day in a week

Moreover, we confirmed the result of Figure 6 by using the Mann Whitney  $U$  test (under the significance level of 0.05). The symbol of M-X in Table III means two groups of data on Monday vs. the other day in a week (from Tuesday to Sunday). If we make the data of commit frequency or commit interval on Monday as a reference, for commit frequency, both M-Ss reject the *null hypothesis*; for commit interval, M-S (Sunday) rejects the *null hypothesis*. The finding indicates that their differences are statistically significant.

TABLE III. MANN WHITNEY  $U$  TEST FOR MON. VS. THE OTHER DAY

		M-T	M-W	M-T	M-F	M-S	M-S
Commit frequency	$P$	0.144	0.096	0.081	0.395	<b>0.001</b>	<b>&lt; 0.001</b>
	$Z$	1.461	1.665	1.854	0.851	<b>-3.410</b>	<b>-4.593</b>
Commit interval	$P$	0.799	0.159	0.111	0.361	0.135	<b>0.001</b>
	$Z$	0.254	1.410	1.593	0.914	1.495	<b>-3.323</b>

Because two consecutive commits sometimes do not occur in the same day, it may affect the result of our experiment. Hence, we re-classified the data into two groups, namely *Weekdays* and *Weekend* to reduce such a bias. That is, all pairs of two consecutive commits that occur on weekdays (from Monday to Friday) were categorized into one group. The similar results are presented in Figure 7 and Table IV, indicating that the commit frequency and commit interval over a weekend are different from those on weekdays.

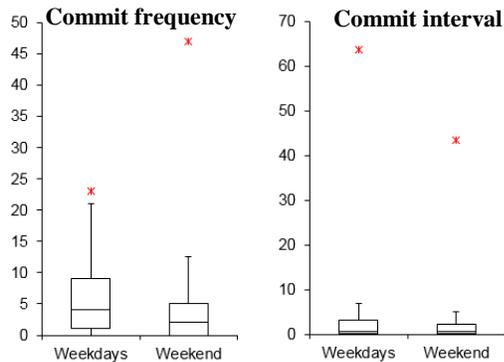


Figure 7. Standardized box plots for weekdays and a weekend

TABLE IV. MANN WHITNEY *U* TEST FOR A WEEKEND VS. WEEKDAYS

	<i>P</i> -value	<i>Z</i> -value
Commit frequency	< 0.001	-9.068
Commit interval	0.004	-2.876

The statistics for the SVN repository of Tomcat show that there are only 19.4% of commits completed on weekends, which accounts for low commit frequency values shown in the above figures. To our surprise, we also found that the group *Weekend* tends to have smaller values of commit interval than the other group *Weekdays*, perhaps due to that developers feel inclined to finish the task of committing a small number of changes at a certain time, and then have a rest in the remainder of the weekend. This may cause periods of inactivity. So, we selected the top 5% of long commit intervals in Tomcat (see the heavy tail in Figure 3), and found that 31.4% and 27.3% of these intervals occur on weekdays and weekends respectively. Interestingly, the ratios of the number of long commit intervals on weekends and weekdays to the corresponding total number of intervals are 5.24% and 1.19%, respectively. This finding indicates that for active OSS projects weekends do affect the normal development process and would result in large commit intervals (less than 48 hours) as a result of weekend breaks.

Thirdly, since the developers of the two projects are from the West, we selected three relatively long holidays, namely Christmas, New Year’s Day and Thanksgiving Day, to prove our conjecture. As New Year’s Day is too soon after Christmas, they could be deemed as one vacation lasting from December 24 to January 6. For POI and Tomcat, the longest commit intervals in hours during the two long vacations (i.e., Christmas & New Year’s Day and Thanksgiving Day) are listed in Table V, where *CN* and *T* represent Christmas & New Year’s Day and Thanksgiving Day, respectively. As we expected, these intervals fall into the top 5% of long commit intervals, implying that long vacations do affect the regular development and maintenance processes of OSS projects and result in long periods of inactivity.

TABLE V. LONGEST COMMIT INTERVALS DURING LONG VACATIONS

	2009	2010	2011	2012
POI_CN	97.26	98.58	95.48	191.66
Tomcat_CN	100.73	190.25	63.51	61.12
POI_T	59.30	81.31	75.06	84.92
Tomcat_T	40.02	44.33	48.59	50.21

Finally, we analyzed the influence of active developer’s behavior on commit intervals. Our prior work [4] found that a minority of committers contribute to the vast majority of commits. Hence, active committers can be defined in terms of the number of commits. In this paper, active committers were selected as the top three committers who contribute the most number of commits. Figure 8 presents a comparison of the number of commits per month between active committers and all committers. The curve about active committers is roughly in coincidence with that of all committers, suggesting that active committers play an important role in the development and maintenance of the two projects.

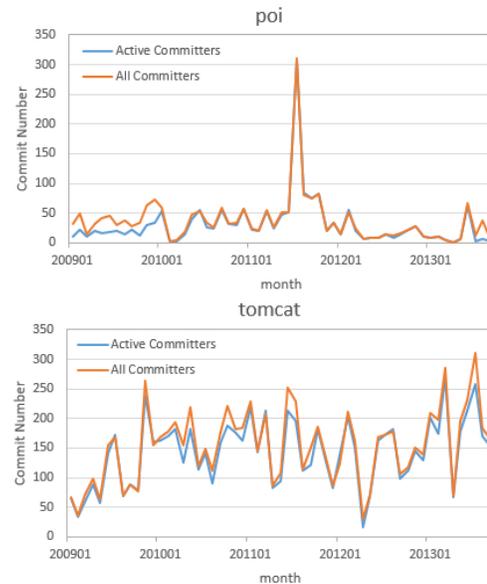


Figure 8. Comparison of commits per month between different committers

As we know, a commit interval between two consecutive commits is related to two committers. Then, according to the type of committers, we divided the top 5% of long commit intervals into three types of intervals: (1) a commit interval that two committers (active or not) are the same person is classified into group *A*, (2) if two committers are active and the same person, a commit interval is classified into group *B*, and (3) a commit interval belongs to group *C* in case two committers are both active regardless of whether they are the same person.

Table VI shows the proportions of the number of a specific category of commit intervals to the total number of the top 5% of long commit intervals. To our surprise, the proportion of the type *C* is considerable, especially for Tomcat. This indicates that active committers’ collective behavior can influence the length of commit intervals, though they have different tasks, backgrounds, habits and expertise. Interestingly, the top three longest intervals in both POI and Tomcat are all included in the group *C*. Very long inter-commit times (e.g., several months) between two consecutive commits to the projects in question do exist, perhaps because developers lose interest in the

projects, go on vacation, complete their tasks and wait for new tasks, and other accidental events.

TABLE VI. PROPORTIONS OF DIFFERENT TYPES OF COMMIT INTERVALS

Projects	A	B	C
Poi	46.73%	39.25%	71.96%
Tomcat	49.16%	46.37%	87.59%

*Finding: The working mode of SVN and developers' work habits contribute to commit bursts, while active committer's individual behavior and long vacations are the primary factors that result in long inter-commit times.*

#### D. Threats to Validity

Because the results of this paper were obtained based on two case studies of OSS projects written in Java, they might not be generalizable for closed-source projects. Moreover, it is possible that we accidentally selected the two projects that exhibit such characteristics and regular patterns. Thus, we have to validate the generality of our findings based on more OSS projects. On the other hand, whether the results are still suitable for decentralized control system such as Git or not need to be further proved.

#### V. CONCLUSION

This paper conducted an empirical study on two OSS projects on the apache.org that are under centralized revision control in terms of commit interval, i.e., the time difference between two consecutive commits. In summary, our primary findings and contributions are described as follows:

(1) The distributions of commit intervals in both POI and Tomcat roughly follow power laws, with commit bursts and heavy tails, implying very long inter-commit times do exist, though a vast majority of commits are completed in a short time. This finding validates Barabási's conclusion [18] within the field of software development, and indicates that the development dynamics of an OSS project can be modeled by means of developers' commit behavior;

(2) We carefully examined the top 5% of long commit intervals using statistical methods, and found that active committer's individual behavior (unpredictable factor) and long vacations (regular factor) are the primary factors that cause very long periods of inactivity. This finding implies that the administrator(s) of an OSS project should pay more attention to active committers' accidental events and regular holidays, and make a reasonable schedule after their departure.

Because many OSS projects have become fully or partially commercial [11], the future work is to design an algorithm for mining anomalies [24] in commit time series, so as to optimize software processes and improve development efficiency.

#### ACKNOWLEDGMENT

This work is supported by the National Basic Research Program of China (No. 2014CB340401), the National Natural Science Foundation of China (Nos. 61272111 and 61273216), the Youth Chenguang Project of Science and Technology of Wuhan City in China (No. 2014070404010232), and the open foundation of Hubei Provincial Key Laboratory of Intelligent Information Processing and Real-time Industrial System (No. zns2013B017).

#### REFERENCES

- [1] C. DiBona, S. Ockman, and M. Stone, Open Sources: Voices from the Open Source Revolution. Sebastopol, CA: O'Reilly, 1999.
- [2] B. O'Sullivan, "Making sense of revision-control systems," Communications of the ACM, 52(9): 56–62, 2009.
- [3] O. Arafat and D. Riehle, "The Commit Size Distribution of Open Source Software," In Proc. the 42<sup>nd</sup> Hawaii Int'l Conf. Syst. Sci. (HICSS'09), USA, 2009, pp. 1–8.
- [4] Y.T. Ma, Y. Wu, and Y.W. Xu, "Dynamics of Open-Source Software Developer's Commit Behavior: An Empirical Investigation of Subversion," arXiv:1309.0897, 2013.
- [5] S.H. Lin, Y.T. Ma, and J.X. Chen, "Empirical Evidence on Developer's Commit Activity for Open-Source Software Projects," In Proc. the 25<sup>th</sup> Int'l Conf. on Softw. Eng. and Knowl. Eng. (SEKE'13), USA, 2013, pp. 455–460.
- [6] 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," Journal of Software Maintenance and Evolution: Research and Practice, 19(2): 77–131, 2007.
- [7] C. Kolassa, D. Riehle, and M. Salim, "A Model of the Commit Size Distribution of Open Source," In Proc. the 39<sup>th</sup> Int'l Conf. Current Trends in Theory and Practice of Comput. Sci. (SOFSEM'13), Czech Republic, 2013, pp. 52–66.
- [8] L. Hattori and M. Lanza, "On the nature of commits," In Proc. the 4<sup>th</sup> Int'l ERCIM Wksp. Softw. Evol. and Evolvability (EVOL'08), Italy, 2008, pp. 63–71.
- [9] J. Choi, J. Moon, J. Hahn, and J. Kim, "Herding in open source software development: an exploratory study," In Proc. the 16<sup>th</sup> ACM Conf. Comput. Supported Cooperative Work (CSCW'13), USA, 2013, pp. 129–134.
- [10] C. Kolassa, D. Riehle, and M. Salim, "The Empirical Commit Frequency Distribution of Open Source Projects," In Proc. the 9<sup>th</sup> Int'l Symp. Open Collaboration (OpenSym'13), China, 2013, p. 18.
- [11] D. Riehle, P. Riemer, C. Kolassa, and M. Schmidt, "Paid vs. Volunteer Work in Open Source," In Proc. the 47<sup>th</sup> Hawaii Int'l Conf. Syst. Sci. (HICSS'14), USA, 2014, pp. 3286–3295.
- [12] P. Hofmann and D. Riehle, "Estimating Commit Sizes Efficiently," In Proc. the 5<sup>th</sup> IFIP WG 2.13 Int'l Conf. Open Source Systems (OSS'09), Sweden, 2009, pp. 105–115.
- [13] R. Purushothaman and D.E. Perry, "Toward Understanding the Rhetoric of Small Source Code Changes," IEEE Transactions on Software Engineering, 31(6): 511–526, 2005.
- [14] A. Alali, H. Kagdi, and J. Maletic, "What's a Typical Commit? A Characterization of Open Source Software Repositories," In Proc. the 16<sup>th</sup> IEEE Int'l Conf. Program Comprehension (ICPC'08), The Netherlands, 2008, pp. 182–191.
- [15] A. Hindle, D. Germán, and R. Holt, "What do large commits tell us?: a taxonomical study of large commits," In Proc. the 5<sup>th</sup> Int'l Working Conf. Mining Softw. Repos. (MSR'08), Germany, 2008, pp. 99–108.
- [16] K. Crowston, K. Wei, J. Howison, and A. Wiggins, "Free/Libre Open Source Software Development: What We Know and What We Do Not Know," ACM Computing Surveys, 44(2): Article No. 7, 2012.
- [17] J. Watson, "Psychology as the Behaviorist Views It," Psychological Review, 20(2): 158–177, 1913.
- [18] A.-L. Barabási, "The origin of bursts and heavy tails in human dynamics," Nature, 435(7039): 207–211, 2005.
- [19] C. Castellano, S. Fortunato, and V. Loreto, "Statistical physics of social dynamics," Reviews of Modern Physics, 81(2): 591–646, 2009.
- [20] P. Singh, "The small-world effect: The influence of macro-level properties of developer collaboration networks on open-source project success," ACM Transactions on Software Engineering and Methodology, 20(2): Article No. 6, 2010.
- [21] B. Collins-Sussman, "The subversion project: buiding a better CVS," Linux Journal, 94 (2): Article No. 3, 2002.
- [22] A. Clauset, C. Shalizi, and M. Newman, "Power-law distributions in empirical data," SIAM Review, 51(4): 661–703, 2009.
- [23] H.B. Mann and D.R. Whitney, "On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other," Annals of Mathematical Statistics, 18(1): 50–60, 1947.
- [24] J. Eyolfson, L. Tan, P. Lam, "Do time of day and developer experience affect commit bugginess," In Proc. the 8<sup>th</sup> Working Conf. Mining Softw. Repos. (MSR 2011), USA, 2011, pp. 153–162.

# Documenting the Mined Feature Implementations from the Object-oriented Source Code of a Collection of Software Product Variants

R. AL-msie'deen<sup>1</sup>, A.-D. Seriai<sup>1</sup>, M. Huchard<sup>1</sup>, C. Urtado<sup>2</sup>, and S. Vauttier<sup>2</sup>

<sup>1</sup>LIRMM / CNRS & Montpellier 2 University, France, {al-msiedee, seriai, huchard}@lirmm.fr

<sup>2</sup>LGI2P / Ecole des Mines d'Alès, Nîmes, France, {Christelle.Urtado, Sylvain.Vauttier}@mines-ales.fr

## Abstract

*Companies often develop a set of software variants that share some features and differ in other ones to meet specific requirements. To exploit existing software variants and build a Software Product Line (SPL), a Feature Model (FM) of this SPL must be built as a first step. To do so, it is necessary to mine optional and mandatory features in addition to associating the FM with its documentation. In our previous work, we mined a set of feature implementations as identified sets of source code elements. In this paper, we propose a complementary approach, which aims to document the mined feature implementations by giving them names and descriptions, based on the source code elements that form feature implementations and use-case diagrams of software variants. The novelty of our approach is that it exploits commonality and variability across software variants, at feature implementations and use-cases levels, to run Information Retrieval methods in an efficient way. To validate our approach, we applied it on Mobile media and ArgoUML-SPL case studies. The results of this evaluation showed that most of the features have been documented correctly.*

**Keywords:** *Software variants, Software Product Line, Feature documentation, Code comprehension, Formal Concept Analysis, Relational Concept Analysis, Use-case diagram, Latent Semantic Indexing.*

## 1 Introduction

Software variants often evolve from an initial product, developed for and successfully used by the first customer. These product variants usually share some common features and differ regarding others. As the number of features

and the number of software variants grows, it is worth re-engineering them into a Software Product Line (SPL) for systematic reuse [1]. The first step towards re-engineering software variants into SPL is to mine the Feature Model (FM) of these variants. To obtain such a FM, common and optional features for software variants have to be identified. This consists in identifying among source code elements, groups of such elements that implement candidate features and associating them with their documentation (*i.e.*, a feature name and description). In our previous work [2], we proposed an approach for feature mining from the object-oriented source code of software variants (REVPLINE approach<sup>1</sup>). REVPLINE allows us mining of functional features as a set of Source Code Elements (SCEs) (*e.g.*, package, class, attribute, method or method body elements).

To assist a human expert to document the mined feature implementations, we propose an automatic approach which associates names and descriptions using source code elements of feature implementations and use-case diagrams of software variants. Compared with existing work that documents source code (*cf.* section 6), the novelty of our approach is that we exploit commonality and variability across software variants at feature implementation and use-case levels, to apply Information Retrieval (IR) methods in an efficient way. Considering commonality and variability across software variants enables us to cluster the use-cases and feature implementations into *disjoint* and *minimal* clusters based on Relational Concept Analysis (RCA); where each cluster is disjoint from the others and consists of a minimal subset of feature implementations and their corresponding use-cases. Then, we use Latent Semantic Indexing (LSI) to define a similarity measure that enables us to identify which use-cases characterize the name and de-

---

<sup>1</sup>REVPLINE stands for RE-engineering Software Variants into Software Product Line.

scription of each feature implementation by using Formal Concept Analysis (FCA).

The remainder of this paper is structured as follows: Section 2 briefly presents the background. Section 3 shows an overview of the feature documentation process. Section 4 presents the feature documentation process step by step. Section 5 describes the experimentation. Section 6 discusses the related work. Finally, section 7 concludes and provides perspectives for this work.

## 2 Background

This section provides a glimpse on FCA, RCA and LSI. It also shortly describes the example that illustrates the remaining sections of the paper.

### 2.1 Formal and Relational Concept Analysis

FCA [3] is a classification technique that extracts a partially ordered set of concepts (the concept lattice) from a dataset composed of objects described by attributes (the formal context). A concept is composed of two sets: an object set called the concept's extent and an attribute set called the concept's intent. The extent is the maximal set of objects which share the maximal set of attributes of the intent (*cf.* Section 4.3). We use the concepts of the AOC-poset, namely the concepts that introduce at least one object or one attribute. The interested reader can find more information about our use of FCA in [2].

RCA [4] is an iterative version of FCA in which the objects are classified not only according to the attributes they share, but also according to the relations between them (*cf.* Section 4.1). Data are encoded into a *Relational Context Family* (RCF), which is a pair  $(K, R)$ , where  $K$  is a set of formal (object-attribute) contexts  $K_i = (O_i, A_i, I_i)$  and  $R$  is a set of relational (object-object) contexts  $r_{ij} \subseteq O_i \times O_j$ , where  $O_i$  (domain of  $r_{ij}$ ) and  $O_j$  (range of  $r_{ij}$ ) are the object sets of the contexts  $K_i$  and  $K_j$ , respectively (*cf.* Table 1). A RCF is used in an iterative process to generate, at each step, a set of concept lattices. Firstly, concept lattices are built, using the formal contexts only. Then, in the following steps, a scaling mechanism translates the links between objects into conventional FCA attributes and derives a collection of lattices whose concepts are linked by relations (*cf.* Figure 2). The interested reader can find more information about RCA in [4]. For applying FCA and RCA we used the Eclipse eRCA platform<sup>2</sup>.

### 2.2 Latent Semantic Indexing

LSI is an advanced Information Retrieval (IR) method [1]. LSI assumes that software artifacts can be regarded

<sup>2</sup>The eRCA: <http://code.google.com/p/erca/>

as textual documents. Occurrences of terms are extracted from the documents in order to calculate similarities between them and then to classify together a set of similar documents as related to a common concept (*cf.* Section 4.2). The heart of LSI is the singular value decomposition technique. This technique is used to mitigate noise introduced by stop words (like "the", "an", "above") and to overcome two issues arising in natural languages processing: *synonymy* and *polysemy*. The effectiveness of IR methods is usually measured by metrics including *recall*, *precision* and *F-measure*. In our context, for a given use-case (query), recall is the percentage of correctly retrieved feature implementations (documents) to the total number of relevant feature implementations, while precision is the percentage of correctly retrieved feature implementations to the total number of retrieved feature implementations. F-Measure defines a trade-off between precision and recall, so that it gives a high value only in cases where both recall and precision are high. All measures have values in [0%, 100%]. If recall equals 100%, all relevant feature implementations (documents) are retrieved. However, some retrieved feature implementations might not be relevant. If precision equals 100%, all retrieved feature implementations are relevant. Nevertheless, relevant feature implementations might not be retrieved. If F-Measure equals 100%, all relevant feature implementations are retrieved. However, some retrieved feature implementations might not be relevant. The interested reader can find more information about our use of LSI in [2].

### 2.3 The Mobile Tourist Guide Example

We consider in this example four software variants of a Mobile Tourist Guide (MTG) application. These applications enable users to inquire about some tourist information on mobile devices. MTG\_1 supports core MTG functionalities: *view map*, *place marker on a map*, *view direction*, *launch Google map* and *show street view*. MTG\_2 has the core MTG functionalities and a new functionality called *download map from Google*. MTG\_3 has the core MTG functionalities and a new functionality called *show satellite view*. MTG\_4 supports *search for nearest attraction*, *show next attraction* and *retrieve data* functionalities, together with the core ones.

## 3 The Feature Documentation Process

Our goal is to document the mined feature implementations by using the use-case diagrams of these variants. In our work, we rely on the same assumption as in the work of [5] stating that each use-case represents a feature. The feature documentation process aims at identifying which

use-cases characterize the name and description of each feature implementation. We rely on lexical similarity to identify the use-cases that characterize the name and description of feature implementations. The performance and efficiency of the IR technique depends on the size of the search space. In order to apply LSI, we take advantage of the commonality and variability between software variants to group feature implementations and the corresponding use-cases in the software family into disjoint, minimal clusters (e.g., *Concept\_1* of Figure 2). We call each disjoint minimal cluster a *Hybrid Block* (HB). After reducing the search space to a set of hybrid blocks, we rely on textual similarity to identify, from each hybrid block, which use-cases depict the name and description of each feature implementation.

For a product variant, our approach takes as inputs the set of use-cases that documents the variant and the set of mined feature implementations that are produced by REVPLINE. Each use-case is identified by its name and description. This information represents domain knowledge that is usually available from software variants documentation (i.e., requirement model). In our work, the use-case description consists of a short paragraph in a natural language. Our approach provides as its output a name and description for each feature implementation based on a use-case name and description. Each use-case is mapped into a functional feature thanks to our assumption. If two or more use-cases have a relation with the same feature implementation, we consider them all as the documentation for this feature implementation.

Figure 1 shows an overview of our feature documentation process. The first step of this process aims at identifying hybrid blocks based on RCA (cf. Section 4.1). In the second step, LSI is applied to determine the similarity between use-cases and feature implementations (cf. Section 4.2). This similarity measure is used to identify use-case clusters based on FCA. Each cluster identifies the name and description for feature implementation (cf. Section 4.3).

## 4 Feature Documentation Step by Step

In this section, we describe the feature documentation process step by step. According to our approach, we identify the feature name and its description in three steps as detailed in the following.

### 4.1 Identifying Hybrid Blocks of Use-cases and Feature Implementations via RCA

We use the existing use-case diagrams of software variants to document the feature implementations mined from those variants. In order to apply LSI in an efficient way, we need to reduce the search space for use-cases and feature implementations. Starting from existing feature im-

plementations and use-cases, these elements are clustered into disjoint minimal clusters (i.e., hybrid blocks) to apply LSI. The search space is reduced based on the commonality and variability of software variants. RCA is used to cluster: the use-cases and feature implementations common to all software variants; the use-cases and feature implementations that are shared by a set of software variants, but not all variants; the use-cases and feature implementations that are held by a single variant.

A RCF for feature documentation is automatically generated from use-case diagrams and the mined feature implementations associated with software variants<sup>3</sup>. The RCF corresponding to our approach contains two formal contexts and one relational context, as illustrated in Table 1. The first formal context represents the *use-case diagrams*. The second formal context represents *feature implementations*. In the formal context of *use-case diagrams*, objects are use-cases and attributes are software variants. In the formal context of *feature implementations*, objects are feature implementations and attributes are software variants. The relational context (i.e., *appears-with*) indicates which use-case appears in the same software variants as feature implementations.

For the RCF presented in Table 1, a close-up view of two lattices of the Concept Lattice Family (CLF) is represented in Figure 2. As an example of hybrid block we can see in Figure 2 a set of use-cases (in the extent of *Concept\_1* of the *Use-case Diagrams* lattice) that always appear with a set of feature implementations (in the extent of *Concept\_6* of the *Feature Implementations* lattice). As shown in Figure 2, RCA allows us to reduce the search space by exploiting

<sup>3</sup>Source code : <https://code.google.com/p/rcafca/>

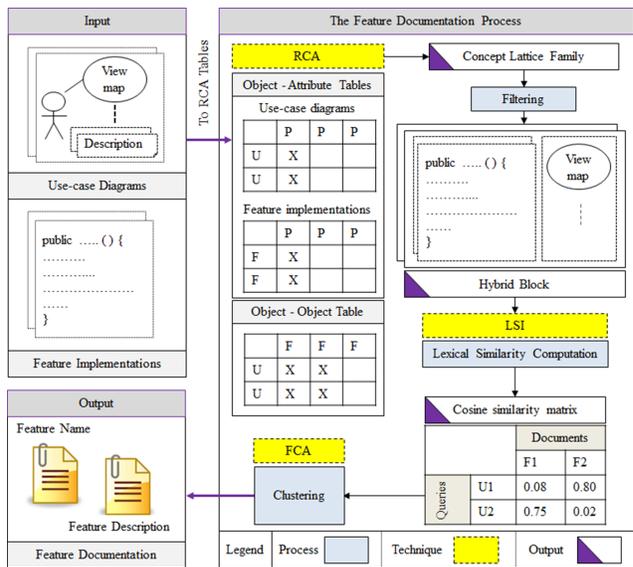


Figure 1: The feature documentation process.

Table 1: The RCF for features documentation.

Use_case_Diagrams	MTG.1	MTG.2	MTG.3	MTG.4	Feature_Implementations	MTG.1	MTG.2	MTG.3	MTG.4
View Map	X	X	X	X	Feature_Implementation_1	X	X	X	X
Launch Google Map	X	X	X	X	Feature_Implementation_2	X	X	X	X
View Direction	X	X	X	X	Feature_Implementation_3	X	X	X	X
Show Street View	X	X	X	X	Feature_Implementation_4	X	X	X	X
Place Marker on Map	X	X	X	X	Feature_Implementation_5	X	X	X	X
Download Map		X			Feature_Implementation_6		X		
Show Satellite View			X		Feature_Implementation_7			X	
Show Next Attraction				X	Feature_Implementation_8				X
Search For nearest attraction					Feature_Implementation_9				X
Retrieve Data				X	Feature_Implementation_10				X

Relational context: appears-with	Feature_Implementation_1	Feature_Implementation_2	Feature_Implementation_3	Feature_Implementation_4	Feature_Implementation_5	Feature_Implementation_6	Feature_Implementation_7	Feature_Implementation_8	Feature_Implementation_9	Feature_Implementation_10
View Map	X	X	X	X	X					
Launch Google Map	X	X	X	X	X					
View Direction	X	X	X	X	X					
Show Street View	X	X	X	X	X					
Place Marker on Map	X	X	X	X	X					
Download Map						X				
Show Satellite View							X			
Show Next Attraction								X	X	X
Search For Nearest Attraction								X	X	X
Retrieve Data								X	X	X

commonality and variability across software variants. In our work, we are filtering CLF to get a set of hybrid blocks from bottom to top<sup>4</sup>. Figure 2 shows an example of hybrid block (the dashed block).

## 4.2 Measuring the Lexical Similarity Between Use-cases and Feature Implementations via LSI

Based on the previous step, each hybrid block consists of a set of use-cases and a set of feature implementations. We need to identify from each hybrid block which use-cases characterize the name and description of each feature implementation. To do so, we use textual similarity between use-cases and feature implementations. This similarity measure is calculated using LSI. We rely on the fact that a use-case

<sup>4</sup>Source code : <https://code.google.com/p/fecola/>

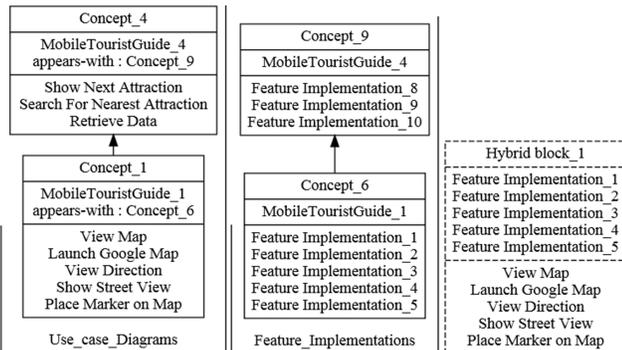


Figure 2: Parts of the CLF deduced from Table 1.

corresponding to the feature implementation is supposed to be lexically closer to this feature implementation than to the other feature implementations. Similarity between use-cases and feature implementations in the hybrid blocks is computed in three steps as detailed below.

### 4.2.1 Building the LSI Corpus

In order to apply LSI, we build a corpus that represents a collection of documents and queries. In our work, each *use-case* name and description in the hybrid block represents a *query* and each *feature implementation* represents a *document*. This document contains all the segments of SCE names as a result of splitting words into terms using the Camel-case technique. Regardless of word location (first, middle or last) in the SCE name, we store all words in the document. For example, for the SCE name *ManualTestWrapper* all words are important: *manual*, *test* and *wrapper*. We apply the same technique to all feature implementations. Our approach creates a query for each use-case. This query contains the use-case name and its description. We apply the same process to all use-cases. To be processed, documents and queries must be normalized as follows: stop words, articles, punctuation marks, or numbers are removed; text is tokenized and lower-cased; text is split into terms; stemming is performed (e.g., removing word endings); terms are sorted alphabetically. We use WordNet<sup>5</sup> to do some simple preprocessing (e.g., stemming and removal of stop words). The most important parameter of LSI is the number of term-topics (i.e., *k-Topics*) chosen. A term-topic is a collection of terms that co-occur often in the documents of the corpus, for example {*user*, *account*, *password*, *authentication*}. In our work, the number of *k-Topics* is equal to the number of feature implementations for each corpus.

### 4.2.2 Building the Term-document and the Term-query Matrices for each Hybrid Block

All hybrid blocks are considered and the same processes are applied to them. The *term-document matrix* is of size  $m \times n$ , where  $m$  is the number of terms extracted from feature implementations and  $n$  is the number of feature implementations (i.e., documents) in a corpus. The matrix values indicate the number of occurrences of a term in a document, according to a specific weighting scheme. In our work, terms are weighted with the *TF-IDF* function (the most common weighting scheme) [1]. The *term-query matrix* is of size  $m \times n$ , where  $m$  is the number of terms that are extracted from use-cases and  $n$  is the number of use-cases (i.e., queries) in a corpus. An entry of term-query matrix refers to the weight of the  $i^{th}$  term in the  $j^{th}$  query.

<sup>5</sup><http://wordnet.princeton.edu/>

### 4.2.3 Building the Cosine Similarity Matrix

Similarity between use-cases and feature implementations in each hybrid block is described by a *cosine similarity matrix* which columns (documents) represent vectors of feature implementations and rows (queries) vectors of use-cases. The textual similarity between documents and queries is measured by the cosine of the angle between their corresponding vectors [2].

### 4.3 Identifying Feature Name via FCA

Based on the cosine similarity matrix we use FCA to identify, from each hybrid block of use-cases and feature implementations, which elements are similar. To transform the (numerical) cosine similarity matrices into (binary) formal contexts, we use a 0.7 threshold (after having tested many threshold values). This means that only pairs of use-cases and feature implementations having a calculated similarity greater than or equal to 0.70 are considered similar.

For example, for the hybrid block *Concept\_1* of Figure 2 the number of term-topics of LSI is equal to 5. In the formal context associated with this hybrid block, the use-case "Launch Google Map" is linked to the feature implementation "Feature Implementation\_1" because their similarity equals 0.86, which is greater than the threshold. However, the use-case "View Direction" and the feature implementation "Feature Implementation\_5" are not linked because their similarity equals 0.10, which is less than the threshold. The resulting AOC-poset is composed of concepts which *extent* represents the use-case name and *intent* represents the feature implementation.

For the MTG example, the AOC-poset of Figure 3 shows five non comparable concepts (that correspond to five distinct features) mined from a single hybrid block (*Concept\_1* from Figure 2). The same feature documentation process is used for each hybrid block.

## 5 Experimentation

To validate our approach, we ran experiments on two Java open-source softwares: Mobile media software variants (small systems) [6] and ArgoUML-SPL (large systems) [7]. We used 4 variants for Mobile media, 10 for ArgoUML.

Concept_0	Concept_1	Concept_2
Feature Implementation_1	Feature Implementation_2	Feature Implementation_3
Launch Google Map	Place Marker on Map	Show Street View
Concept_3	Concept_4	
Feature Implementation_4	Feature Implementation_5	
View Direction	View Map	

Figure 3: The documented features from *Concept\_1*.

The advantage of having two case studies is that they implement variability at different levels: class and method levels. In addition, these case studies are well documented and their feature implementations, use-case diagrams and FMs are available for comparison of our results and validation of our proposal<sup>6</sup>. Table 2 summarizes the obtained results.

Table 2: Features documented from case studies.

#	Feature Name	Hybrid block #	k-Topics	Evaluation Metrics		
				Recall	Precision	F-Measure
Mobile Media						
1	Delete Album	HB_1	4	100%	100%	100%
2	Delete Photo	HB_1	4	100%	50%	66%
3	Add Album	HB_1	4	100%	100%	100%
4	Add Photo	HB_1	4	100%	50%	66%
5	Exception handling	HB_2	1	100%	100%	100%
6	Count Photo	HB_3	3	100%	50%	66%
7	View Sorted Photos	HB_3	3	100%	50%	66%
8	Edit Label	HB_3	3	100%	100%	100%
9	Set Favourites	HB_4	2	100%	50%	66%
10	View Favourites	HB_3	2	100%	50%	66%
ArgoUML-SPL						
1	Class diagram	HB_1	1	100%	100%	100%
2	Logging	HB_2	2	100%	50%	66%
3	Cognitive support	HB_2	2	100%	100%	100%
4	Deployment diagram	HB_3	1	100%	100%	100%
5	Collaboration diagram	HB_4	2	100%	50%	66%
6	Sequence diagram	HB_4	2	100%	50%	66%
7	State diagram	HB_5	1	100%	100%	100%
8	Activity diagram	HB_6	2	100%	100%	100%
9	Use case diagram	HB_6	2	100%	100%	100%

For the two case studies presented, we observe that the *recall* values are 100% of all the features that are documented. The recall values are an indicator for the efficiency of our approach. The values of precision are between [50% - 100%], which is high. F-Measure values rely on precision and recall values. The values of F-Measure are high too, between [66% - 100%] for the documented features. Results show that recall value in all cases is 100% and value of precision either 100% or 50% it is because of the similarity threshold (0.70) in addition, this result is due to search space reduction. In most cases, the contents of hybrid blocks are in the range of [1 - 4] use-cases and feature implementations. Another reason for this good result is that a common vocabulary is used in the use-case descriptions and feature implementations, thus lexical similarity was a suitable tool. In our work we cannot use a fixed number of topics for LSI because we have hybrid blocks (clusters) with different sizes.

The column (*k-Topics*) in Table 2 represents the *number of term-topics*. All feature names produced by our approach, in the column (*Feature Name*) of Table 2, represent the names of the use cases. For example, in the FM of Mobile media [6] there is a feature called *sorting*. The name proposed by our approach for this feature is *view sorted photos* and its description is "the device sorts the photos based on the number of times photo has been viewed".

<sup>6</sup>Case studies and code : <http://www.lirmm.fr/CaseStudy>

As a *limitation* to our approach, developers might not use the same vocabularies to name SCEs and use-cases across software variants. This means that lexical similarity may not be reliable (or should be improved with other techniques) in all cases to identify the relationship between use-cases and feature implementations. Furthermore, using FCA as clustering technique has also limits. FCA deals with binary formal context. This affects the quality of the result, since a similarity value of 0.99 (*resp.* 0.69) is treated as a similarity value of 0.70 (*resp.* 0). Selecting the appropriate number of dimensions (K) for the LSI representation is an open research question.

## 6 Related Work

Most existing approaches are designed to extract labels, names, topics or identify code to use-case traceability links in a single software system. In the context of feature documentation, most existing approaches manually assign feature names (without any description) to feature implementations. Conversely our approach is designed to automatically assign a name and a description to each feature implementation in a set of software variants based on several techniques (FCA, RCA and LSI). Feature documentation is inferred from use case names and descriptions.

Ziadi *et al.* [8] propose an approach to identify features across software variants. In their work, they manually create feature names. In our previous work [2], we manually propose feature names for the mined feature implementations. Braganca and Machado [5] describe an approach for automating the process of transforming UML use-cases into FMs. In their work, each use-case is mapped to a feature. The identification of relationships (*i.e.*, traceability links) between use-case diagrams and source code for a single software is the subject of the work by Grechanik *et al.* [9]. Kuhn *et al.* [10] present a lexical approach that uses the log-likelihood ratios of word frequencies to automatically provide labels for components of single software. Xue *et al.* [1] propose an automatic approach to identify the code-to-feature traceability link for a collection of software product variants.

## 7 Conclusion and Perspectives

In this paper, we proposed an approach for documenting the mined feature implementations of a set of software variants. We exploit commonalities and variabilities between software variants at feature implementation and use-case levels to apply IR methods in an efficient way in order to automatically document the mined features. We have implemented our approach and evaluated its results on two case studies. The results of this evaluation showed that most of

the features were documented correctly. Regarding future work, we plan to use the mined and documented features to build automatically the relations between the features of a FM (*i.e.*, reverse engineering FMs).

**Acknowledgments:** This work has been supported by project CUTTER ANR-10-BLAN-0219.

## References

- [1] Y. Xue, Z. Xing, and S. Jarzabek, "Feature location in a collection of product variants," in *WCRE*. IEEE, 2012, pp. 145–154.
- [2] R. Al-Msie'deen, A. Seriai, M. Huchard, C. Urtado, S. Vauttier, and H. E. Salman, "Mining features from the object-oriented source code of a collection of software variants using formal concept analysis and latent semantic indexing," in *SEKE*, 2013, pp. 244–249.
- [3] B. Ganter and R. Wille, *Formal concept analysis - mathematical foundations*. Springer, 1999.
- [4] M. Huchard, M. R. Hacene, C. Roume, and P. Valtchev, "Relational concept discovery in structured datasets," *Ann. Math. Artif. Intell.*, vol. 49, no. 1-4, pp. 39–76, 2007.
- [5] A. Bragança and R. J. Machado, "Automating mappings between use case diagrams and feature models for software product lines," in *SPLC*. IEEE, 2007, pp. 3–12.
- [6] L. P. Tizzei, M. O. Dias, C. M. F. Rubira, A. Garcia, and J. Lee, "Components meet aspects: Assessing design stability of a software product line," *Information & Software Technology*, vol. 53, no. 2, pp. 121–136, 2011.
- [7] M. V. Couto, M. T. Valente, and E. Figueiredo, "Extracting software product lines: A case study using conditional compilation," in *CSMR*, 2011, pp. 191–200.
- [8] T. Ziadi, L. Frias, M. A. A. da Silva, and M. Ziane, "Feature identification from the source code of product variants," in *CSMR*, 2012, pp. 417–422.
- [9] M. Grechanik, K. S. McKinley, and D. E. Perry, "Recovering and using use-case-diagram-to-source-code traceability links," in *ESEC/SIGSOFT FSE*, 2007, pp. 95–104.
- [10] A. Kuhn, "Automatic labeling of software components and their evolution using log-likelihood ratio of word frequencies in source code," in *MSR*, M. W. Godfrey and J. Whitehead, Eds. IEEE, 2009, pp. 175–178.

# An empirical study on the adoption of C++ templates: Library templates versus user defined templates

Di Wu

Lin Chen

Yuming Zhou

Baowen Xu

State Key Laboratory for Novel Software Technology at Nanjing University, Nanjing, China

nju.wudi@gmail.com

lchen@nju.edu.cn

zhouyuming@nju.edu.cn

bwxu@nju.edu.cn

**Abstract**—Template is a powerful language feature to develop reusable programs, which has long been widely used by C++ programmers. Among various templates, library templates play a key role, as they provide programmers fault-free generic units. But to our surprise, little work has been done to study how these templates are adopted in real software. In this paper, we conduct an empirical study on the adoption of templates to understand how library templates influence programming compared with user defined templates. To this end, we use the historical revisions of ten open source systems, containing 34 million lines of C++ code, to conduct the experiment. The experimental results show that: (1) the growth of using library templates does not result in a substantial decrease of using user defined templates; (2) eight templates account for over 95% of all library template uses; and (3) user defined templates tend to be derived more frequently than library templates. Based on these results, we make two practical suggestions. For C++ novices, we suggest that they focus on learning the eight most commonly-used library templates. For library developers, we suggest that they develop more library templates that can be easily derived.

*Keywords*- Programming Language, C++, Generic Programming, Templates, Empirical Study

## I. INTRODUCTION

C++ template is a representative generic programming feature for developing highly reusable and efficient software components [1]. In the last decades, C++ experiences a long way to enhance the role of templates in generic programming by continuously supplementing new template libraries. In C++ 11 specification, more than one-third contents are devoted to standard template libraries [3]. Library designers hope that these well-designed library templates could enable common programmers to use general components at a higher level of abstraction rather than developing all code from scratch [5]. Due to the safety of library templates, they are often used as the basic elements to create compound data structures in a type safe manner.

Other than allowing directly adopting library templates in programming, C++ also supports programmers to create their own generic units, namely user defined templates. However, with the expansion of C++ libraries, library templates are more powerful and can be replacers of elementary user defined templates in more and more scenarios. In other word, a lot of generic units that were needed to be implemented by common programmers are now supported in new C++ libraries. For instance, smart pointers, which are frequently used to avoid dangling pointers and resource leaks, were always manually created by programmers in the past. However, C++ 11 [3] has

formally introduced two new library templates for smart pointers, namely class `std::shared_ptr` for a pointer that implements the concept of shared ownership, and `std::weak_ptr` for a pointer that implements the concept of exclusive ownership or strict ownership [3]. Together with `std::auto_ptr` provided in C++ 98, these three kinds of smart pointer templates can now satisfy programmers' requirements in using smart pointers in most scenarios.

Since creating user defined templates brings heavy workload and are error-prone, programmers are recommended to make more use of library templates when they need [6]. Although most C++ programmers know the necessity of using library templates, there still exists a great amount of user defined templates in C++ programs. In view of this fact, there is a strong need to investigate how programmers apply templates and why they use or not use library templates. Surprisingly, few studies have been done in this area. This may lead to two problems. For library designers, they cannot know whether libraries they developed really benefit users in their programming work and which libraries are most welcomed by users. For common programmers, especially the novices, they do not understand which libraries are most helpful to them and need their attention to be paid on.

In this paper, we aim to empirically investigate how library templates and user defined templates are adopted in real software. The purpose of this is to understand how library templates influence programming compared with user defined templates. More specially, we want to investigate the following three questions:

- Whether more uses of library templates result in a decrease of using user defined templates?
- Whether programmers tend to use a few library/user-defined templates more frequently?
- Whether library templates or user defined templates are more frequently to be derived?

To answer these questions, we analyze the historical revisions of ten open source systems, containing 34 million lines of C++ code. The experimental results show that: (1) the growth of using library templates does not result in a substantial decrease of using user defined templates; (2) eight templates account for over 95% of all library template uses; and (3) user defined templates tend to be derived more frequently than library templates. Based on these results, we make two practical suggestions. For C++ novices, we suggest that they focus on learning the eight most commonly-used library templates. For library developers, we suggest that they develop more library templates that can be easily derived.

The rest of the paper is organized as follows. Section II introduces library templates and user defined templates in a nutshell. Section III describes the research questions, data sets, and analysis methods. Section IV reports the experimental results. Section V discusses related work. Section VI concludes the paper and outlines the direction for future work.

## II. TEMPLATES AND THEIR USAGES: LIBRARY VERSUS USER DEFINED

Library templates are defined in C++ libraries, which include STL [7], Boost [23], Loki [24] and ATL [25], etc. The most well-known one, STL, is the standard template library recognized by the language specification [3]. It provides a large collection of classes that meet all kinds of needs, together with several algorithms that operate on them. Since all components of the STL are templates, they can be used for arbitrary element types. Containers, iterators, and algorithms are three key components in STL [5]. By contrast, user defined templates are explicitly defined in programs and are utilized by programmers themselves. These templates are used to implement functionality which is relatively specific to users' requirements. Generally, user defined templates are not as universal as library templates.

Even though library templates and user defined templates are defined in different sources, they have same ways of use. Both library templates and user defined templates can be divided into function templates and class templates. Here, class templates also include struct templates [4]. The way to use function templates is to invoke them as ordinary functions [4], resulting in a function instance called **specialized function**. Compared with function templates, more ways are available for using class templates. According to C++ language specification [3], class templates can be initialized, specialized, or derived. Here we demonstrate these concepts by excerpting the real usages of class templates in TortoiseSVN:

- Instantiation means to create a class instance by associating template parameters with arguments. The most commonly used template arguments are non-template types. For example:

```
quick_hash<CHashFunction> index;
```

This statement denotes to declare an object *index* with a class type *CHashFunction*, which is an instance of the user defined template *quick\_hash*.

- Specialization, which can be created by a declaration introduced by "template<>", denotes to establish a separate implementation for a template. For example:

```
template <>
struct ice_not<true>
{
    BOOST_STATIC_CONSTANT(bool, value = false);
};
```

This is an example to provide the template *ice\_not* with a specific implementation through specialization. All type instances with argument "true" will be instantiated from this special implementation.

- Derivation indicates to define a child template by explicitly extending the parent class template. For example:

```
template<class B, STREAM_TYPE_ID type>
class CInStreamImpl:public
CInStreamImplBase< CInStreamImpl<B, type> , B , type>
```

In the example, *CInStreamImplBase* acts as a parent template, and *CInStreamImpl* derives it in the public manner.

## III. RESEARCH METHOD

In this section, we first formulate the three research questions about the adoption of templates. Then, we introduce the data sets used for investigating these research questions. Finally, we describe the analysis method.

### A. Research Questions

We aim to compare the adoption of library templates to user defined templates in real software. The three research questions (RQ) are hence formulated as follows.

- **RQ 1:** *Will there be a decrease of adopting user defined templates with an increase use of library templates? C++ library templates can help programmers get rid of hard work in constructing various elementary user defined templates. In other word, if library templates are more widely used, more user defined templates that have similar functionalities with existing library templates are unnecessary and thus can be replaced by using these library templates.*
- **RQ 2:** *Are library/user-defined templates biasedly used? If so, which templates are the most commonly-used ones? With so many library templates available, we intuitively conjecture that elementary library templates may be used more frequently. If this hypothesis is proven to be true, we can pick out the libraries containing commonly-used templates and give advices to programmers about their focus on these library templates.*
- **RQ 3:** *Are library class templates less often derived than user defined class templates? Similar to ordinary classes, class templates can also be used in the manner of derivation, which is an important feature in object oriented programming. If the answer of RQ3 is "Yes", it may indicate that user defined templates are more easily to be inherited than library templates. Further, there will be potential opportunities for library developers to produce more extensible templates to improve libraries' ability in supporting object oriented programming.*

### B. Data Sets

To investigate the proposed research questions, we analyze 10 projects of different categories listed on <http://sourceforge.net>. These projects are well logged and long aged, ranging from 4 to 15 years old. Due to the vast historical

revisions of the projects, it is too costly to investigate all of them. Additionally, some dump versions may also impact the result. Therefore, we regularly select the last revisions without dump in each season, that is, four revisions for a project each year since it started to be released. Finally, 317 historical revisions of the 10 projects are chosen. They totally contain over 34M lines of C++ code. Source files of these projects are obtained from open source repositories by using *svn* and *git clone* tools. The detailed information about these projects is listed in Table I.

TABLE I. OPEN SOURCE PROJECTS IN THE STUDY

Project	Age	C++ KLOC	Category
TortoiseSVN	11	405.806	Software Development
FileZilla	10	82.941	Communication
KeePass	4	56.364	Security & Utilities
MeshLab	9	247.556	Graphic
SAGA GIS	10	257.530	Science & Engineering
Bitcoin	4	66.359	Business & Enterprise
Console	7	20.959	System Administration
Cool Reader	7	232.239	Home & Education
Warzone 2100	9	180.517	Games
VLC	15	119.801	Audio & Video

### C. Analysis Method

We analyze source code of the target systems by the following steps. At the first step, filter non-C++ source files. Our target files are C++ header files and source files. To filter out those redundant ones, we use the C++ *Strict* option in *Understand* and build an udb database for each selected revision. At the second step, detect the uses of template in C++ source files. To process the udb databases, we write Perl scripts invoking *Understand* APIs to find all program points related to templates. If a template definition is found, then the template is identified as a user defined template. If the namespaces like "std" and "boost" are found, then the template is recognized as a library template. At the third step, check the ways of template uses. In this step, we run a Perl script to check out different usages of all library function/class templates. The uses of user defined templates are examined with the same approach. At the fourth step, store the data. We write a Java program to automatically process the output of running Perl scripts and store the final data in xls files. We choose xls as the file format because they are easily to do statistical analysis and to plot figures.

## IV. EMPRICAL RESULTS

In this section, we report in detail the experimental results and discuss the possible threats to the validity of our study.

### A. RQ1: Will there be a decrease of adopting user defined templates with an increase use of library templates?

Based on the data obtained, we firstly measure the correlation between the density of library template uses and the density of user defined template uses. Here, density is calculated as the number of template uses divided by C++ KLOC. We choose the density (instead of the number of template uses) as the evaluation indicator because it reflects the frequency of template uses by eliminating impacts brought by the increase of code size of projects. In our context, one use

of template is either a call to a function template or an instantiation, specialization, and derivation of a class template.

If an increase use of library templates causes a decrease in using user defined templates, a negative relationship will be shown by the two groups of data. To see if this is true, we use both observational and statistical methods. An observational result related to RQ 1 is shown in Figure 1. The figure plots the density of library template uses (red lines) versus the density of user defined template uses (blue lines). Inspecting both sub-figures for function template uses and class template uses, we cannot find a clear evidence supporting an increase of the density of using library templates results in a decrease of the density of adopting user defined templates. By contrast, we observe synchronous fluctuations of the two lines.

To precisely validate the relevance between the two densities, we use the Spearman Rank Correlation coefficient. Table 2 summarizes for each project the Spearman Rank Correlation coefficient between the density of library template uses and the density of user defined template uses. More specifically, the "Function Templates" column reports the correlation between library function templates and user defined function templates. The "Class Templates" column reports the correlation between library class templates and user defined class templates.

TABLE II. SPEARMAN RANK CORRELATIONS BETWEEN ADOPTION OF LIBRARY TEMPLATES AND ADOPTION USER DEFINED TEMPLATES

Project	Coefficient of Adoption of Templates (p-value)	
	Function Templates	Class Templates
TortoiseSVN	0.16 (0.31)	0.25 (0.12)
FileZilla	-	0.78 (0.00)
KeePass	-0.60 (0.02)	0.85 (0.00)
MeshLab	0.71 (0.00)	0.85 (0.00)
SAGA GIS	0.77 (0.00)	-0.33 (0.06)
Bitcoin	0.72 (0.00)	-0.18 (0.45)
Console	0.31 (0.13)	0.14 (0.50)
Cool Reader	0.81 (0.00)	0.17 (0.38)
Warzone 2100	0.49 (0.00)	0.68 (0.00)
VLC	0.45 (0.00)	0.71 (0.00)

In the function templates group, seven of ten projects show significant results (p-value < 0.05). The only outlier is FileZilla, whose use number of user defined function templates is zero, thus the correlation value fails to be produced. Specifically, six of ten projects show positive relationships, including four high positive correlations (above 0.7) and two mild positive relationships (0.4 to 0.7). Only one project shows a moderate negative correlation (-0.60). To summarize, the data for function template uses generally shows that the tendencies of adopting library function templates and user defined function templates follow the similar pattern, indicating that an increase of library template uses does not lead to a corresponding decrease of user template uses when the templates are applied to create specialized functions. Regarding the adoption of class templates, the data basically does not validate strong relationships. To be specific, among the ten projects, only five

show a significant correlation ( $p < 0.05$ ). The five significant results, however, all indicate relatively positive relevance, with four of them showing high positive relationships (above 0.7) and one showing mild positive correlation (0.4 to 0.7). Due to the low ratio of significances, this group of data cannot testify bound relevance between use density of library class

templates and use density of user defined templates. Nevertheless, the consistent positive relationships shown by five valid samples can still demonstrate that with number of library class template uses rising, the number of adopting user template uses does not reduce.

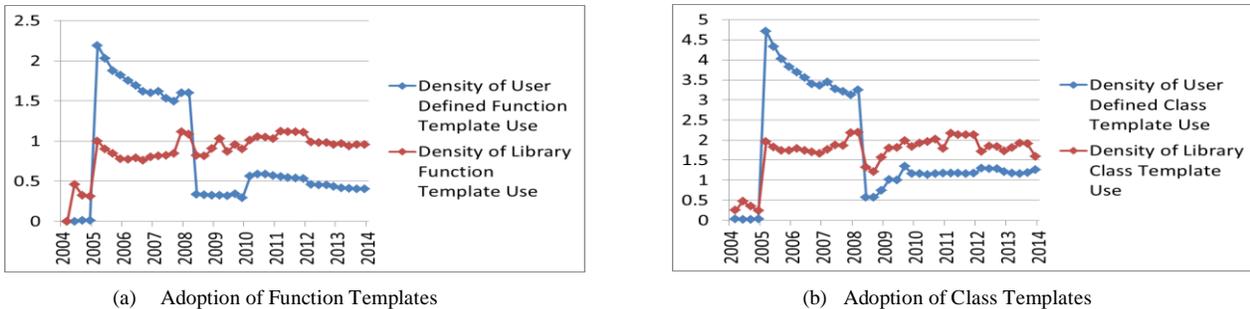


Figure 1. Density of Adopting User Defined Templates vs. Density of Adopting Library Templates (*the case in TortoiseSVN*)

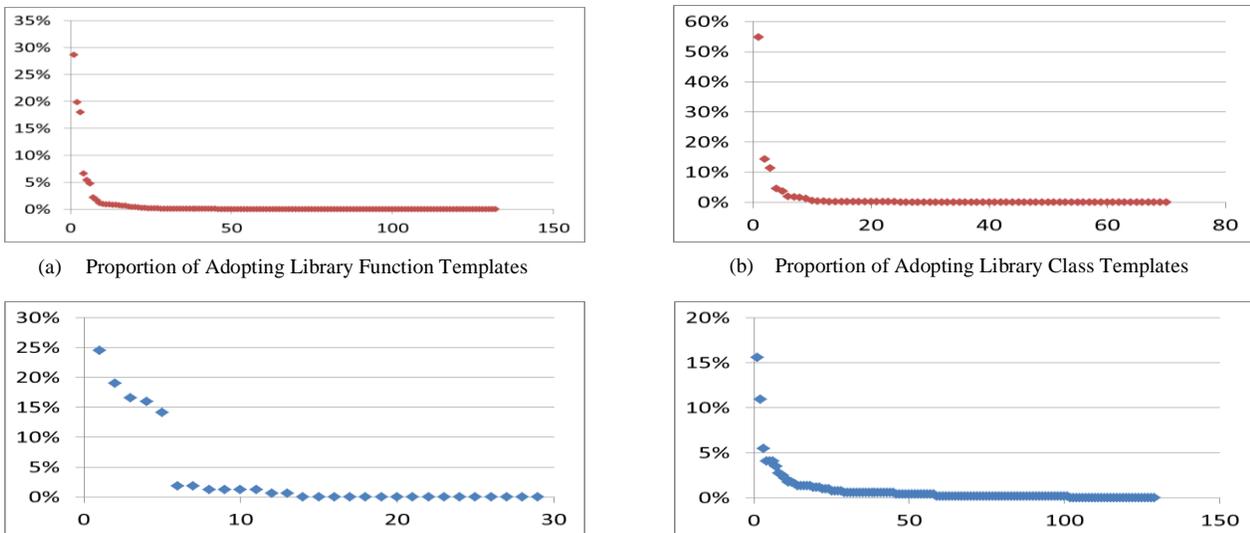


Figure 2. Proportion of Adopting Different Templates (*Figure c and Figure d reflect adoption of user defined templates in TortoiseSVN*)

Overall, the function templates data group strongly indicates that **the increase of library template uses does not lead to the reduction of user defined template uses**, and the class templates data group validates this result slightly weaker. We deduce the reason for this fact lies in the effects of some irreplaceable user defined templates. That is, even though library templates (such as STL, Boost, and ATL) play an important role in programming, they are only elementary components to construct compound data structures but cannot replace user defined templates. Meanwhile, user defined templates are always needed in specific scenarios and sometimes more frequently used than library templates.

**B. RQ2: Are library/user-defined templates biasedly used? If so, which templates are the most commonly-used ones?**

Figure 2 shows the result for RQ 2. In the sub-figures, each red dot represents a library template, while each blue dot represents a user defined template. For every dot, the x-value

is its rank among all templates in its group sorted by the numbers of their uses in descending order, while its y-value is the percentage calculated as the number of its uses divided by the total template uses. In our initial inspection, the dots basically distribute as a power curve. To precisely test our observation, we apply power functions to fit the dots and finally obtain good fitness (a large  $R^2$  value) for all sub-graphs. The fitting result demonstrates the use rates of templates appear in power distributions, showing that **both library templates and user defined templates are high biasedly used**.

The result for RQ 2 signifies the adoption of a small proportion of templates accounts for a majority of overall template uses. Based on this knowledge, we pick out the most popularly-used library templates from the experimental data and manually query which libraries they belong to. Table III lists the names of these libraries and their proportions in use.

Here, we list eight libraries whose proportions are larger than 1%. These libraries totally account for over 95% of all library template uses.

TABLE III. MOST COMMONLY-USED TEMPLATE LIBRARIES

Library Name	Number	Proportion
Standard Container: Vector	1586	54.80%
Standard Container: List	416	14.37%
Standard Container: Map	328	11.33%
Standard Utility: Smart Pointers	152	5.25%
Standard Container: Set	132	4.56%
Standard Utility: Pair	58	2.00%
Language Support Library: Numeric Limits	49	1.69%
Standard Container: Deque	45	1.55%
<b>Total</b>	<b>2766</b>	<b>95.58%</b>

On account of this result, we recommend C++ novices to focus on learning how to use the template libraries listed in Table III. The reason is that these libraries are statistically proved to be frequently used by practitioners in developing open source projects and maybe helpful in their understanding of templates. For library developers, we advise them to pay their attention on the safety and compatibility of these most frequently-used template libraries when they update C++ libraries. The reason is that the changes or defects in these libraries may result in severe impacts on programs adopting these library templates.

C. RQ3: Are library class templates less often derived than user defined class templates?

As stated in Section 3, class templates are mainly used in three different ways: Instantiation, Specialization, and Derivation. Since RQ 3 focuses on derivation, we calculate the proportion of derivations in class template uses. Table IV reports the proportion of derivations among class template use. Among three kinds of usages of class templates, derivations are more likely to appear in the uses of user defined class templates. This is true for eight of ten projects. The average data also support this conclusion, with a proportion of 7.64% (derivations of user defined class templates) over 0.54% (derivations of library class templates). The result indicates that compared with library templates, user defined templates are more frequently to be inherited to define child templates.

Due to the poor ratio of derivations of library class templates, we recommend library designers to develop more library templates that can be easily inherited to declare new class templates, especially some elementary components which are easily to be used as base classes. In this manner, library templates could play more important role in object oriented programming and better facilitate programmers.

D. Threats to Validity

**External Threats.** Due to well-known accessibility reasons, our study only investigates open source projects. The research questions investigated in this paper may not compatible with regularities of template uses in industrial software, as different

ways of managing software development probably make a difference in adoption of templates. Moreover, even though our study is performed on the historical revisions of 10 different kinds of projects, all of them are application software. Since no system software is included, the results may be skewed to some extent. Finally, since covering all historical revisions for a project is a hard work, we select representative revisions for each project. However, in this way, we probably miss several revisions that are milestones in the process of project development.

**Internal Threats.** The internal threat lies in the inclusion of test code in the open source projects. The test code may influence the accuracy of code size calculation. However, test code only accounts for a small proportion of the total code size. Therefore, we believe that our results are reliable.

TABLE IV. PROPORTION OF DERIVATIONS AMONG CLASS TEMPLATE USE

Project	Library	User Defined
TortoiseSVN	1.23%	5.46%
FileZilla	0.27%	0%
KeePass	0.99%	16.72%
MeshLab	1.12%	5.16%
SAGA GIS	0.97%	10.39%
Bitcoin	0.315%	0%
Console	0%	0.84%
Cool Reader	0%	2.74%
Warzone 2100	0%	0.12%
VLC	0.53%	35%
<b>Average</b>	0.54%	7.64%

V. RELATED WORK

Existing studies about C++ templates mainly focused on STL. For example, Josuttis [5] comprehensively summarized the adoption of STL in implementing an amount of useful libraries. Austern [6] studied in depth on the design ideas and infrastructures of STL and discussed the detailed specifications of STL. Other studies on C++ templates include how to refine design of built-in templates among the C++ standard [13], how to identify idiom usages of generic libraries [14], how to effectively compile templates at runtime [15], and how to compare C++ templates with the generic features in other languages [16]. The only empirical study about adoption of C++ template was undertaken by Basit *et al.* [8]. They conducted a case study to observe whether templates can reduce clone code in STL. As a result, they found out many code duplications that cannot be eliminated by templates.

With more and more open source codes available through the Internet, researchers have been gradually starting to make use of empirical techniques such as data mining to access useful information about language adoption. These techniques can not only help to understand adoption of different languages, but also supply objective evidence to evaluate features in various languages. Existing studies in this area include predicting future trends of language evolution by comparative study among languages [17], mining software repositories to investigate language features [11] and language adoption [18] [19] [20], and understanding language elements

with data mining techniques [21] and artificial experiments [22].

Among all previous studies, the work done by Parnin *et al.* [9] [10] is most related to our work. They deeply investigated the adoption of Java generics in open source projects. The papers report that: (1) Use of Java generics sometimes shows negative relationship with the number of type casts; (2) Java generics can prevent some code duplications; (3) Generics are usually adopted by several contributors in a project rather than all committers; (4) Developers do not always convert raw types to generic types; (5) Different abilities of IDE in supporting generics do not influence the adoption of generics.

Due to distinct characteristics between C++ and Java, it is improper to investigate C++ templates by following the same way in studying how Java generics are adopted. The reasons are two-fold. First, Java has a super-type "Object", which is always applied to create generic data structures before generics are introduced. However, C++ does not have such a super-type. Therefore, we cannot study conversions from using such language feature to using templates. Second, since most C++ open source projects are not as well logged as Java projects, it is difficult to obtain enough information about the behaviors of committers.

## VI. CONCLUSION AND FUTURE WORK

In this paper, we explore how C++ templates are adopted in open source projects by comparing library templates to user defined templates. The whole study is undertaken by investigating the uses of both function templates and class templates in open source systems. As a result, we find that an increasing use of library templates does not reduce the use of user defined templates, both library and user defined templates are biasedly adopted, and user defined templates are more likely to be derived compared to library templates. Based on these results, we give practical suggestions to both library developers and common programmers. For example, programmers are advised to pay attention on learning those commonly-used libraries which we pick out through empirical study, and we recommend library developers to develop more extensible library templates which may increase libraries' ability to support object oriented programming.

In the future work, we will replicate the study to validate the above-mentioned findings on more systems. Further, an investigation is being done by us to uncover factors that influence the adoption of templates and practical benefits of applying templates. In addition, new features introduced into C++ 11 are also our concerns.

## ACKNOWLEDGMENT

This work is supported by the National Key Basic Research and Development Program of China (2014CB340702), the National Natural Science Foundation of China (91318301,

61321491, 61170071), and the Natural Science Foundation of Jiangsu Province (BK2011190). We especially thank Yibiao Yang from ISE group at Nanjing University for his valuable suggestions on our experiments.

## REFERENCES

- [1] ISO. Information Technology—Programming Languages—C++. ISO/IEC 14882-1998. *ISO/IEC*. 1998.
- [2] ISO. Information Technology—Programming Languages—C++, Second Edition. ISO/IEC 14882-2003. *ISO/IEC*. 2003.
- [3] ISO. Information Technology—Programming Languages—C++, Third Edition. ISO/IEC 14882-2011. *ISO/IEC*. 2011.
- [4] D. Vandevoorde, N. Josuttis. C++ Templates: The Complete Guide. *Addison-Wesley*. 2002.
- [5] N. Josuttis. The C++ Standard Library: A Tutorial and Reference - Second Edition. *Addison-Wesley*. 2012.
- [6] M. Austern. Generic Programming and the STL. *Addison-Wesley*. 1999.
- [7] A. Stepanov, M. Lee. The Standard Template Library. Presentation to the C++ standards committee, March 7, 1994.
- [8] H. Basit, D. Rajapakse, and Jarzabek, S. An Empirical Study on Limits of Clone Unification Using Generics. *SEKE'05*, 109-114, 2005.
- [9] C. Parnin, C. Bird, and E. Murphy-Hill. Java Generics Adoption: How New Features are Introduced, Championed, or Ignored. *MSR'11*, 3-12, 2011.
- [10] C. Parnin, C. Bird, and E. Murphy-Hill. Adoption and use of Java generics. *Empir Software Eng*, 18(6):1047-1089. 2013.
- [11] M. Hoppe, S. Hanenberg. Do Developers Benefit from Generic Types? An Empirical Comparison of Generic and Raw Types in Java. *OOPSLA'13*, 457-474, 2013.
- [12] J. G. Siek, A. Lumsdaine. A language for generic program-ming in the large. *Science of Computer Programming*, 76 (5): 423-465, 2007.
- [13] D. Groger, J. Järvi, J. Siek, B. Stroustrup. G. Dos Reis, and A. Lumsdaine. Concepts: Linguistic Support for Generic Programming in C++. *OOPSLA'06*, 291-310, 2006.
- [14] A. Sutton, R. Holeman, and J. I. Maletic. Identification of Idiom Usage in C++ Generic Libraries. *ICPC'10*, 160 - 169, 2010.
- [15] M. J. Cole, S. G. Parker. Dynamic compilation of C++ template code. *Journal Scientific Programming*, 11(4): 321 - 327. 2003.
- [16] R. Garcia, J. Järvi, A. Lumsdaine, J. Siek, and J. Willcock. An Extended Comparative Study of Language Support for Generic Programming. *Journal of Functional Programming*, 17(2): 145 - 205. 2007.
- [17] Y. Chen, R. Dios, A. Mili, and L. Wu. An Empirical Study of Programming Language Trends. *Software, IEEE*. 22(3): 72-79, 2005.
- [18] R. Dyer, H. Rajan, H. N. Nguyen, and T. N. Nguyen. Mining Billions of AST Nodes to Study Actual and Potential Usage of Java Language Features, *ICSE'14*. 2014.
- [19] S. Karus, H. Gall. A Study of Language Usage Evolution in Open Source Software. *MSR'11*, 13-22, 2011.
- [20] L. A. Meyerovich, A. Rabkin. Empirical Analysis of Programming Language Adoption. *OOPSLA'13*, 1-18, 2013.
- [21] O. Callaú, R. Robbes, and D. Rählisberger. How Developers Use the Dynamic Features of Programming Languages: The Case of Smalltalk. *MSR'11*, 23-32, 2011.
- [22] S. Kleinschmager, S. Hanenberg, R. Robbes, É. Tanter, and A. Stefik. Do Static Type Systems Improve the Maintainability of Software Systems? An Empirical Study. *ICPC'12*. 153 - 162, 2012.
- [23] Boost C++ Library. <http://www.boost.org>.
- [24] Loki Library. <http://loki-lib.sourceforge.net>.
- [25] Introduction to ATL. <http://msdn.microsoft.com/en-us/library/hdf7fy18.aspx>.

# TyS – A Framework to Facilitate the Implementation of Object-Oriented Type Checkers

Francisco Ortin

Computer Science Dept.  
University of Oviedo  
ortin@lsi.uniovi.es

Daniel Zapico

Computer Science Dept.  
Capgemini Consulting  
danzapico@gmail.com

Jose Quiroga

Computer Science Dept.  
University of Oviedo  
quirogajose@uniovi.es

Miguel Garcia

Computer Science Dept.  
University of Oviedo  
garciamiguel@uniovi.es

**Abstract**— Type systems are mainly aimed at providing the absence of erroneous behaviors. Their formalization is commonly used to prove specific safety properties, but those formal specifications are not usually used to implement the language. Once a type system has been proven to be sound, a type checker must be developed in the implementation of the language processor. In this article we present TyS, a framework to facilitate the implementation of type checkers, following widely known object-oriented design patterns. TyS processes a type system specification file, generating the implementation of the corresponding type checker. The generated code relies on an API provided by the framework, and it can be reutilized by the rest of components and tools used in the language implementation. TyS has been used with different lexer and parser tools, developing both dynamic and static type systems.

**Keywords**— *type checker; type system; design patterns; language implementation*

## I. INTRODUCTION

*Type* is defined as a collection of objects having similar structure [1], and the branch of mathematics and logic that concerns with classifying objects into types is called *type theory*. With the appearance of computers and programming languages, type theory found practical application in the construction of type systems. A *type system* is defined as a tractable syntactic method for proving the absence of certain program behaviors, by classifying phrases according to the kinds of values they compute [2]. Therefore, a type system classifies program values into types, considering certain program constructions illegal on the basis of its type. For instance, a type system may prohibit passing a string value as an argument to a function that receives an integer parameter.

*Typechecking* is the analysis that detects semantic inconsistencies and anomalies, guarantying that entities match their declaration and preventing unsafe and ill-behaved programs from ever running. The algorithm that performs type checking is called *type checker* [3]. A language processor (e.g., compiler or interpreter) typically implements a *type checker* in its semantic (contextual) analysis phase [4].

Static (compile-time) type checking provides many benefits [2]. The most obvious one is *error detection*, which allows the early detection of programming errors, making possible to fix them immediately rather than discovering the type errors at runtime. Another common benefit is runtime performance,

because many type operations are checked by the compiler, reducing the number of type checks performed at runtime [5] [6]. Types also enforce disciplined programming, defining a kind of partial contract between implementers and users. Additionally, type systems may involve better *safety*, as programs accepted by a safe type system will behave with the absence of runtime type errors [7]. Finally, types also constitute a form of *documentation*, specifying up-to-date information about the behavior of entities [2].

These are the most common benefits of the traditional use of type systems for language design and implementation. However, type systems have also been applied in multiple different scenarios, such as reverse engineering [8], testing [9], web metadata [10], and even security [11].

### A. Type System Formalization

Type systems are formalized with precise notations and definitions. The detailed proof of formal properties (e.g., type safety) give confidence in the appropriateness of their definition [7]. Typical landmarks in type system formalization are the Russell’s original ramified theory of types [1], Ramsey’s simple theory of types [12], Church’s typed lambda-calculus [13], Löf’s constructive type theory [14], and Berardi’s pure type systems [15]. Type checkers are sometimes specified by means of attribute grammars [16]. Based on these formal methods, a range of tools have been developed, achieving different goals:

- *Proof assistants* or interactive theorem provers are software tools to assist with the development of formal systems and their proofs, by means of human-machine collaboration. A formal system such as the type system and the semantics of a programming language can be defined with these tools. Theorems such as type safety can then be proved. Examples of two powerful proof assistants are Isabelle [17] and Coq [18].
- *Lightweight analysis and verification of programs*. These tools are sometimes referred to as *lightweight formal methods* because they do not prove the correctness of type systems. They commonly perform model checking in order to detect programming errors that are not ordinarily detected until runtime. Different examples of these tools are ESC/Java [19], SLAM [20] or SPIN [21]. PLT Redex is another lightweight mechanization tool designed for specifying, debugging and testing operational semantics and type systems of programming languages [22].

- *Language processor development tools.* Most of the existing compiler and interpreter construction tools are centered in the development of scanners and parsers. Others offer the development of *type checkers* and interpreters, employing different mechanisms. For instance, the Synthesizer Generator [23] supports attribute grammars with incremental attribute evaluation, whereas the ASF+SDF Meta-Environment is based on conditional term rewriting rules [24]. There are also works based on translating to code the inference rules that formalize a type system, such as TCG [25] and Tinker-Type [26].

### B. Implementation of Type Checkers

The code generated by the tools mentioned above commonly follows the execution flow of the formalization used (e.g., a deductive system based on a set of inference rules), and it usually involves lower runtime performance than an *ad hoc* implementation [27]. Besides, the generated code is frequently difficult to debug and trace, because it comes from a general translation of mathematical specifications. In occasions, the formalizations used by the language designer (and hence the generated code) are unknown to the compiler programmer.

In this paper we present TyS, a framework to facilitate the implementation of object-oriented type checkers whose type soundness might be previously proven. The generated code follows widespread object-oriented design patterns to facilitate the integration with the available reusable code, components, tools and frameworks. The language independent design patterns used are easily understandable and maintainable by the language processor developer [28], and they model the reusable dynamic and static type checking features of existing type systems. Moreover, they offer a straightforward way to extend and adapt type checkers from the specific requirements of different programming language type systems.

## II. ARCHITECTURE

Fig. 1 shows the architecture of the TyS framework. A type checker specification text file identifies all the types in the type system, the subtyping relation, and all the operations (rules) supported by each type. This file is processed by TyCC, the type checker constructor. TyCC generates a specific implementation of the type checker described in the specification file, depending on the input parameters passed. These parameters include the exception class used to report type errors (e.g., `SemanticException` and `ParserException` in ANTLR [29]), the selected output language used to implement the generated type checkers, the class implementing the `TypeFactory` interface (Section IV), and the directory where the API is placed. Regardless the parameters passed to TyCC, the type checker implementation follows the same object-oriented design patterns. If there is any error in the specification file, TyCC shows the appropriate message and aborts the generation of the output type checker implementation.

The API provides the general services used by any type checker generated with TyCC. These services include a mechanism to represent type expressions, a type table and a type factory to reuse objects representing the same type, and the runtime type exceptions used in the framework. The library has one implementation for each different output language

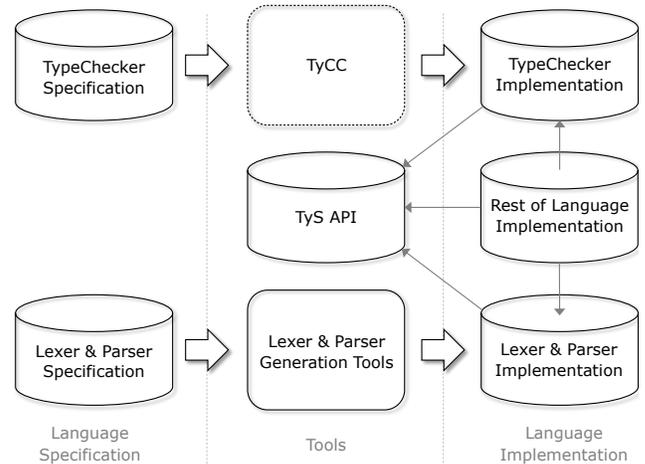


Figure 1. Architecture of the TyS framework.

supported. The functionality provided by the API can be used to implement both static and dynamic type checking (e.g., in compilers and interpreters).

Both the API and the generated type checker are used to implement the language processor, together with the rest of the language implementation. TyS has been designed to be independent of the rest of phases and tools used in the language implementation. We have used TyS with both yacc/bison and ANTLR parser generators, and in single- and multi-pass language processors [30].

## III. SPECIFICATION OF TYPES

The framework will be described by specifying an example type checker of an imperative language, quite similar to the C programming language –the source code is available for download at [30].

### A. Type Identification

As mentioned, the types and its operations are described in a text file. We first identify the types and their subtyping relation. In our example we specify 9 different types:

```
Types = {
    Char < Int < Real
    Char < TString
    Bool < Int
    Int < Bool
    Array
    Pointer
    Function
    Void
}
```

All the types must be declared in the `Types` block, and they can appear more than once. The subtyping relation is identified with the `<` symbol, indicating that the type on the left is a subtype (promotion, coercion or implicit cast) of the type on the right. In our example, `Char` is a subtype of `Int`, which in turn is a subtype of `Real`. Notice that the set of types can be partially ordered, that is, not every pair of types need be related (e.g., `Function` or `Void`).

The subtyping relation generates an `implicitCast(Type):Type` polymorphic method used to

perform the implicit type coercions in the generated type checker. If the type represented by the object passed as the implicit parameter (*this*) promotes to the type represented by the argument, *implicitCast* returns the argument; null is returned otherwise. The behavior of the *implicitCast* operation can be overridden in particular types. Therefore, it is not mandatory to define all the subtyping rules with the < symbol. More complex type convention rules, such as covariant, invariant, and contravariant subtyping relations (e.g., for *Array* and *Function*), can be later defined as type operations (rules) – an example is presented in the following subsection.

### B. Type Operations

After identifying the types and their basic subtyping relations, the specific operations (rules) of each type should be defined. The code generated by TyCC follows the *Composite* design pattern (Fig. 2) [31]. Any type is generalized with the *Type* interface, where all the available operations are declared. Two kinds of types can be defined: primitive types (*leaf* nodes in the *Composite* pattern) that represent a type by themselves; and *composite* types (intermediate nodes) that are constructed using other (primitive or composite) types.

The default implementations of the operations declared in *Type* are provided by the *BaseType* abstract class. These default implementations are inherited by all the derived classes. The particular behavior for a specific type can be modified by implementing the corresponding method in the derived classes, using dynamic binding (method overriding).

A primitive type is simply declared by specifying the type identifier and, between curly braces, the list of the operations it provides. The following example is an excerpt of the *Int* type specification:

```
class Int {
  Type comparison(Type t) {
    if ( t.implicitCast(TypeTable.getInstance()
      .getType("Real")) == null )
      throw new TypeException("Int and "+t.getName()+
        " cannot be compared");
    return TypeTable.getInstance().getType("Bool");
  }
}
```

The *comparison* operation defines the type rule for the six comparison operators (*==*, *!=*, *<*, *<=*, *>* and *>=*), defined over integers in our example language. If the type of the second operand is a subtype of *Real* (included), the comparison is correct and the *Bool* type is returned. Otherwise, a *TypeException* is thrown holding an error message (*getName* returns the type identifier). Objects representing types in TyS are obtained through a *TypeTable* class (detailed in Section IV), included in the framework API.

When an operation is defined in a type (e.g., *comparison*), it is included in the *Type* interface. The default implementation (in *BaseType*) throws an exception indicating that the operation is not applicable for that type. Therefore, the corresponding method could be called over any type, but only those types defining that operation will infer the resulting type – as we will see later on, the default implementation of the operations can also be changed.

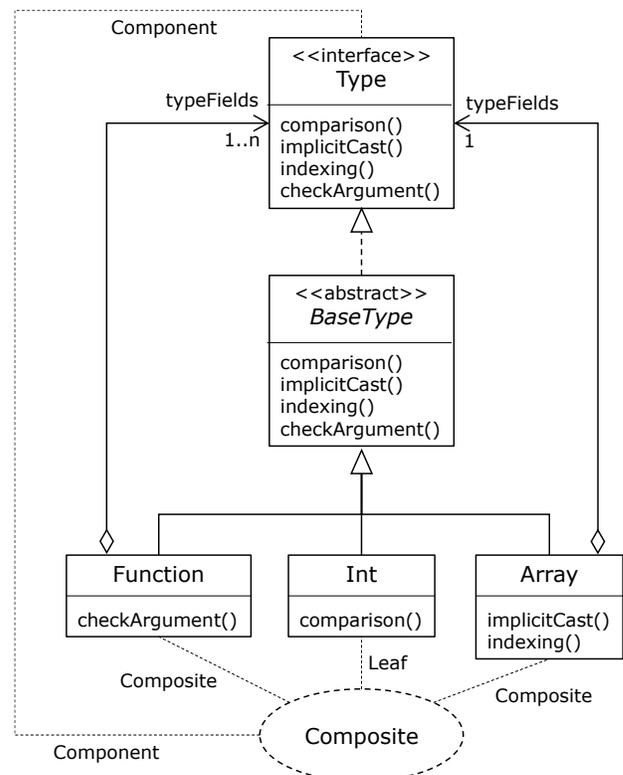


Figure 2. The *Composite* design, used to represent types.

Composite types are constructed using other (primitive or composite) types. This recursive composition is modeled in the *Composite* design pattern (Fig. 2) with an association from the composite type to the child ones (*typeFields*). The following excerpt defines the *Array* composite type in TyS:

```
class Array(Type) {
  String getName() {
    return getChildType().getName() + "[";
  }
  Type getChildType() {
    return getTypeField(0);
  }
  Type implicitCast(Type t) {
    if (t instanceof Array && getChildType().
      .implicitCast(t.getChildType())!=null )
      throw new TypeException(getName()+
        "cannot be converted to "+t.getName());
    return t;
  }
  Type indexing(Type t) {
    if ( t.implicitCast(TypeTable.getInstance()
      .getType("Int")) == null )
      throw new TypeException(t.getName() +
        " cannot be used as an index");
    return getChildType();
  }
}
```

The *Array* type declaration indicates that another child type is required by writing “(Type)” after its type identifier. More types can be specified using a separator (comma or blank space); and *Type+* represents a variable length collection of at least one type – scalar values can also be used [32]. In the generated *Array*

class, a vector of `Types` (`typeFields`) is added as a private attribute, and so is the corresponding `getTypeField(int):Type` public method. In the `Array` example, `getTypeField(0)` returns the type of the elements of the array (its child type).

Two methods generated by TyCC are overridden in the `Array` type. The default behavior of `getName` (returning the "Array" string) is overridden; `[]` is added as a suffix to the type name of the element. Similarly, the behavior of `implicitCast` is also changed. In the example language, we define covariant subtyping for arrays: an array  $A_1$  promotes to another array  $A_2$  when the elements of  $A_1$  promote to the elements of  $A_2$  [2].

The indexing operation represents the inference rule of the `[]` operator. Since the first operand (`this`) is an `Array`, the only condition to be checked is that the second one is a subtype of `Int`. If so, the type of the array element is returned; an exception is thrown otherwise. In the example code provided [30], the indexing operation is also defined by the `Pointer` type. As in the C programming language, the `[]` operator can be applied to both arrays and pointers [30].

A fragment of the `Function` type specification is [30]:

```
class Function(Type+) {
  Type getReturnType() {
    return getTypeField(0);
  }
  int getNumberOfArguments() {
    return typeFields.size()-1;
  }
  Type checkArgument(int i, Type t) {
    if (i>getNumberOfArguments())
      throw new TypeException("Function " +getName()+
        " takes "+getNumberOfArguments()+" args");
    if (t.implicitCast(getTypeField(i)) == null)
      throw new TypeException("The " +i+ "th arg "+
        " is not a subtype of the parameter");
    return getTypeField(i);
  }
}
```

A function type comprises at least a return type plus a possibly empty collection of the types of its arguments (`Type+`). Therefore, we consider the first mandatory type in `typeFields` as the return type (`getReturnType`). Similarly, the `numberOfArguments` is the number of child types minus one, the return type.

The `checkArgument` operation checks whether an argument of type `t` can be passed as the  $i^{th}$  parameter. First, the function is tested to have at least `i` parameters. Second, the type of the argument (`t`) must be a subtype of the  $i^{th}$  parameter. If so, the type of the parameter is returned.

As shown in Fig. 2, `BaseType` holds the default behavior of all the operations defined for any type. The developer of the type checker can also use TyS to modify the default implementations given for any operation. The following specification defines two example default implementations, the `assignment` and `cast` type operations, for every type:

```
class BaseType {
  Type assignment(Type t) {
    if (t.implicitCast(this) == null)
      throw new TypeException(t.getName() +
        " cannot be assigned to "+getName());
    return this;
  }
  Type cast (Type t) {
    String t1 = getName(), t2=t.getName();
    if (t1.equals("Void")||t1.equals("Function")||
      t2.equals("Void")||t2.equals("Function"))
      throw new TypeException("Cannot cast from " +
        getName() + " to " + t.getName());
    return t;
  }
}
```

In an assignment, the type of the expression on the right must promote to the operand on the left, and the resulting type is the type of the operand on the left. The example default behavior of the `cast` operation is that any type but `Void` and `Function` can be cast to another type. This default behavior is then modified in `Array` (not shown for the sake of brevity), so that two arrays can be cast when the types of their corresponding elements can also be cast [30].

#### IV. FRAMEWORK API

We have seen how types are specified by defining the operations they provide. We now describe how TyS can be used by the rest of components comprising a compiler implementation (Fig. 1). For this purpose, the TyS API provides services for building and managing the objects that represent the types specified by the programmer.

##### A. Obtaining Types

The `TypeTable` class in Fig. 3 is in charge of managing the objects representing types. A single instance of this class is created, following the *Singleton* design pattern (`getInstance():TypeTable`) [31]. The main method of this class is `getType(String):Type` that receives the type identifier as a parameter, and returns the object representing that type.

In our example [30], we have used the ANTLR tool for language recognition [29]. The following piece of code shows an excerpt of our type checker implementation:

```
type returns [Type t] { t= null;} :
( "int" {t=TypeTable.getInstance().getType("Int");}
| "real" {t=TypeTable.getInstance().getType("Real");}
| "bool" {t=TypeTable.getInstance().getType("Bool");}
| "char" {t=TypeTable.getInstance().getType("Char");}
| "string" {t=TypeTable.getInstance()
              .getType("TString");}
)
( "[" "]" {t=TypeTable.getInstance()
              .getType("Array("+t.getName()+")"); }
| "*" {t=TypeTable.getInstance()
              .getType("Pointer("+t.getName()+")");}
)*
;
```

The previous production defines the syntax of types, excluding functions, in our imperative language. The production returns the corresponding `Type` object. The first primitive type

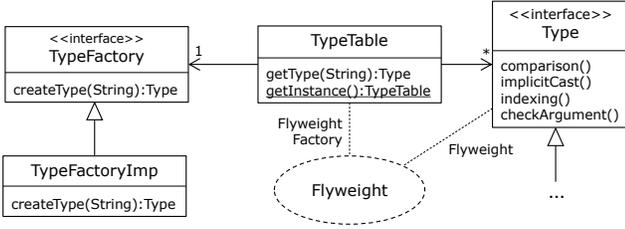


Figure 3. Type table in the TyS framework API.

(int, real, real, bool, char or string) is retrieved from the type table, and stored in the  $t$  local variable. Afterwards, a repetition of zero or more  $[]$  or  $*$  symbols may appear, defining a composite type. The type identifier is built by concatenating the type constructor (Array or Pointer) with the child type ( $t$ ) between brackets. For example, the type  $\text{int}^* []$  is represented with the string "Array(Pointer(Int))". The returned object representing that type is an Array whose child type is a Pointer whose child type is an Int.

The design of the type table follows the *Flyweight* pattern [31]. As shown in Fig. 3, a `TypeTable` instance holds a collection of types. If the requested object type (`getType`) is in that collection, it is simply returned; otherwise, a new object representing the demanded type is created (calling the `createType` method of `TypeFactory`), added to the collection, and returned. The main objective of this design is the reutilization of objects representing the same type, avoiding high memory consumption [28].

Objects representing types cannot be directly created by the programmer. The constructors of all the type classes generated by TyCC are declared with the Java default information hiding level (package). Therefore, new instances of these classes cannot be created from outside their package. Type factories (classes derived from `TypeFactory` in Fig. 3) are in charge of creating types with their `createType` method. They are in the same package as the type classes generated by TyCC. Type factories are not public, and they are used by the public class `TypeTable`. Therefore, the `getType` method of `TypeTable` is the only mechanism to create instances of types from outside their package, as commanded by the *Flyweight* pattern [31].

The `createType` method of `TypeFactory` must parse the string representing a type passed as a parameter, and return the appropriate `Type` object. For this purpose, the default implementation of this functionality (`TypeFactoryImp`) obeys the *Little Language* design pattern [33]. The language these strings follow has the next syntax:

$$\text{type} \rightarrow \text{compositeType} ( \text{type} ( , \text{type} )^* ) \\ | \text{primitiveType}$$

The *primitiveType* and *compositeType* nonterminal symbols depend on the language been specified. In our example, *primitiveTypes* are Char, Int, Real, TString, Bool, and Void; while *compositeTypes* are Array, Pointer, and Function. For instance, if we pass the "Function(Int,Array(Int),Char)" string to the `createType` method of `TypeFactory`, the object graph shown in Fig. 4 is returned. The `ft` object represents the type of a

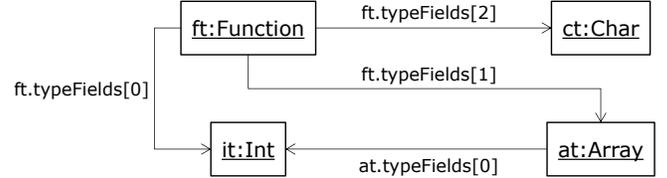


Figure 4. Object diagram representing the "Function (Int, Array (Int), Char)" type.

function that returns an integer and receives two parameters: an array of int, and a char.

It is worth noting that `TypeTable:getType` uses `TypeFactory:createType` and vice versa. The `getType` methods calls `createType` when the requested object is not in the collection of types stored in the `TypeTable`. Similarly, when `createType` is building a new composite type, it calls `getType` to retrieve its child types.

### B. Invoking Type Operations

Once we have an object representing the appropriate type, we only need to call the required operations to perform type checking. Recall that all the operations in all the types are included as methods in the `Type` interface. If the corresponding operation is not applicable, a runtime exception is thrown.

The following fragment of our ANTLR type checker example, shows how the  $[]$  and  $*$  operators are checked:

```

rightOperator [Type arg] returns [Type t]
{
    t = arg;
    Type typeOfIndex = null;
} :
( "[" typeOfIndex = expression "]"
  { t = t.indexing(typeOfIndex); }
| "*"
  { t = t.getPointedType(); }
)*
;
  
```

When the *rightOperator* is an expression between square brackets, the inferred type ( $t$ ) is the result of calling the `indexing` operation against the type received as an argument ( $arg$ ). If that type is an array, its child type is inferred; otherwise, an exception is raised. If the *rightOperator* is  $*$ , the `getPointedType` operation is called instead. That operation simply returns the child type of Pointers, throwing an exception otherwise. This algorithm is executed in a loop, while a  $[]$  or  $*$  operator exists.

## V. IMPLEMENTATION

The TyS framework has been developed in C++ (TyCC) and Java (the framework API). We have implemented different type checkers using the flex, bison/yacc and ANTLR compiler construction tools. These implementations comprise both single- and multi-pass language processors [34].

TyS has been used to implement different examples such as a simple arithmetical expression interpreter (Calc), an object-oriented programming language (Drill), a multi-pass imperative language compiler (Frog), and the example shown in this article (SubC) [30]. It has been used to implement the first prototype of the nitrO virtual machine [35], and its design principles have

been applied in the development of the *Stadyn* programming language [36] [37]. It has also been used for educational purposes, in a compiler construction course [28].

The four examples mentioned in the previous paragraph (including the one presented in this article), the binaries and source code of TyS, and its documentation are available for download at <http://www.reflection.uniovi.es/tys>

## VI. CONCLUSIONS

The TyS framework proposes a collection of object-oriented design patterns, a type system construction tool, and an API to facilitate the implementation of imperative object-oriented type checkers. TyS has been used in the implementation of different kinds of language processors including those developed as single- and multi-pass compilers, and processors that support both static and dynamic typing [30]. These implementations have used TyS in combination with different tools such as yacc/bison and ANTLR. The generated code is highly understandable, following widespread design patterns instead of translating code from a high-level formalization. Memory consumption has been considered, implementing a type table in the API that reutilizes the existing objects representing types.

Currently, TyCC only generates Java source code. We plan to generate other object-oriented languages such as C++ and C#. We also plan to include support for polymorphic types, including type variables and the implementation of a unification algorithm in the framework API [28].

## ACKNOWLEDGMENTS

This work was partially funded by the Department of Science and Innovation (Spain) under the National Program for Research, Development and Innovation: project TIN2011-25978.

## REFERENCES

- [1] A. N. Whitehead, and B. Russell. *Principia Mathematica*, Cambridge University Press, 1910.
- [2] B. C. Pierce. *Types and Programming Languages*, MIT Press, 2002.
- [3] L. Cardelli, "Type Systems," *The Computer Science and Engineering Handbook*, CRC Press, 2004.
- [4] A. V. Aho, J.D. Ullman, and R. Sethi. *Compilers: Principles, Techniques, and Tools*, Addison Wesley, 1985.
- [5] F. Ortin, M. A. Labrador, and J.M. Redondo, "A hybrid class- and prototype-based object model to support language-neutral structural intercession," *Information and Software Technology*, vol. 56, no. 2, pp. 199-219, February 2014.
- [6] J. M. Redondo, F. Ortin, and J. M. Cueva, "Optimizing Reflective Primitives of Dynamic Languages," *International Journal of Software Engineering and Knowledge Engineering*, vol. 18, no. 6, pp. 759-783, September 2008.
- [7] A. K. Wright, and M. Felleisen, "A Syntactic Approach to Type Soundness," *Information and Computation*, vol. 115, no. 1, pp. 38-94, November 1994.
- [8] C. Grothoff, J. Palsberg, and J. Vitek, "Encapsulating objects with confined types," *ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA)*, October 2001.
- [9] D. J. Richardson, "TAOS: Testing with analysis and Oracle support," *International Symposium on Software Testing and Analysis*, pp. 138-153, August 1994.
- [10] H. Hosoya, and B. C. Pierce. "XDuce: A statically typed XML processing language," *ACM Transactions on Internet Technology*, vol. 3, no. 2, pp. 117-148, May 2003.

- [11] M. Abadi, "Security Protocols and Specifications," *Foundations of Software Science and Computation Structures (FOSSACS)*, Lecture Notes in Computer Science, vol. 1578, pp. 1-13, March 1999.
- [12] F. P. Ramsey, "The foundations of mathematics," *Proceedings of the London Mathematical Society*, vol. 25, 1925.
- [13] A. Church, "A formulation of the simple theory of types," *Journal of Symbolic Verification and Synthesis*, Kluwer Academic Publishers, vol. 5, no. 2, pp. 56-68, June 1940.
- [14] P. Martin-Löf. *Intuitionistic Type Theory*, Bibliopolis, 1984.
- [15] S. Berardi, "Towards a mathematical analysis of the Coquand-Huet Calculus of Constructions and the other systems in Barendregt's cube," Technical report, Department of Computer Science, Carnegie-Mellon University, 1988.
- [16] D. E. Knuth, "Semantics of context-free languages," *Mathematical Systems Theory*, vol. 2, no. 2, pp. 127-145, 1968.
- [17] T. Nipkow. (December 2013). *Programming and Proving in Isabelle/HOL*. [Online] Available: <http://www.cl.cam.ac.uk/research/hvg/Isabelle/dist/Isabelle2013-2/doc/prog-prove.pdf>
- [18] A. Chlipala. *Certified Programming with Dependent Types: A Pragmatic Introduction to the Coq Proof Assistant*, The MIT Press, November 2013.
- [19] K. Rustan, M. Leino, G. Nelson, and J. B. Saxe. *ESC/Java User's Manual*, Compaq Systems Research Center, 2000.
- [20] T. Ball, and S.K. Rajamani, "The SLAM Project: Debugging System Software via Static Analysis," *Principles of Programming Languages (POPL)*, pp. 1-3, January 2002.
- [21] G. J. Holzmann. *The Spin Model Checker*, Addison Wesley, 2003.
- [22] M. Felleisen, R. B. Findler, and M. Flatt. *Semantics Engineering with PLT Redex*, The MIT Press, August 2009.
- [23] T. W. Reps, and T. Teitelbaum, "The Synthesizer Generator," *Software Engineering Symposium on Practical Software Development Environments (SDE)*, pp. 42-48, May 1984.
- [24] M. Brand, A. Deursen, J. Heering, H. A. Jong, M. Jonge, T. Kuipers, P. Klint, L. Moonen, P. A. Olivier, J. Scheerder, J. J. Vinju, E. Visser, and J. Visser, "The ASF+SDF Meta-environment: A Component-Based Language Development Environment," *Compiler Construction (CC)*, European Joint Conference on Theory and Practice of Software (ETAPS), pp. 365-370, April 2001.
- [25] H. Gast, "A Generator for Type Checkers," Ph.D. Thesis, Eberhard-Karls-Universität at Tübingen, 2004.
- [26] M. Y. Levin., and B. C. Pierce, "TinkerType: a language for playing with formal systems," *Journal of Functional Programming*, vol. 13, no. 2, pp. 295-316, March 2003.
- [27] M. L. Scott. *Programming Language Pragmatics*, Morgan Kaufmann Publishers, 2000.
- [28] F. Ortin, J. M., Cueva, and D. Zapico, "Patterns for Teaching Type Checking in a Compiler Construction Course," *IEEE Transactions on Education*, vol. 50, no. 3, pp. 273-283, August 2007.
- [29] T. Parr. *The Definitive ANTLR 4 Reference*, 2nd ed., Pragmatic Bookshelf, January 2013.
- [30] F. Ortin, and D. Zapico. (December 2013). *TyS, a Framework to Facilitate the Development of Object-Oriented Type Checkers*. [Online] Available: <http://www.reflection.uniovi.es/tys>
- [31] E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley, October 1994.
- [32] D. Zapico, "Design and implementation of a tool for the automatic generation of object-oriented type systems," Master Thesis, Polytechnic School of Engineering, University of Oviedo, March 2003.
- [33] M. Grand. *Patterns in Java*, vol. I, A Catalog of Reusable Design Patterns Illustrated With UML, Wiley, 1999.
- [34] F. Ortin, B. Lopez, and J. B. G. Perez-Schofield, "Separating Adaptable Persistence Attributes through Computational Reflection," *IEEE Software*, vol. 21, no. 6, pp. 41-49, November 2004.
- [35] F. Ortin, and D. Diez, "Designing an Adaptable Heterogeneous Abstract Machine by means of Reflection," *Information and Software Technology*, vol. 47, no. 2, pp. 81-94, February 2005.
- [36] F. Ortin, D. Zapico, J. B. B. Perez-Schofield, and M. Garcia, "Including both Static and Dynamic Typing in the same Programming Language," *IET Software*, vol. 4, no. 4, pp. 268-282, August 2010.
- [37] F. Ortin, M. Garcia, "Union and intersection types to support both dynamic and static typing," *Information Processing Letters*, vol. 111, no. 6, pp. 278-286, February 2011.

# Software Architecture Rationale Capture through Intelligent Argumentation

Xiaoqing (Frank) Liu, NagaPrashanth Chanda and Eric Christopher Barnes

Computer Science Department  
Missouri University of Science and Technology  
Rolla, MO, USA  
{fliu,nc9r6,ecbyt7}@mst.edu

**Abstract**—A growing model for software architecture defines it as a set of principal design decisions which describe the system. These design decisions need to be made by resolving design issues in a collaborative environment that helps software architects to design the architecture of a system. The architecture design decisions are usually made based on experiences since there aren't defined methods and models for architecture design. Each design decision yields a set of outcomes which impacts both the system architecture and the final product. As software product systems tend to be large in size, one need to understand the rationale behind decision of each architectural element. This is to justify the system's design and to avoid critical architectural problems. Often during these design decision making process the rationale is not fully captured. This paper identifies and addresses the above mentioned research challenge. It presents a method for software stakeholders to use intelligent argumentation system for collaborative rationale capture. The argumentation will be recorded in an online system to document the rationale behind the design decisions resulting in product architecture. Finally, the proposed method is evaluated using a case study. It demonstrates feasibility of capturing software architecture rationale using intelligent on-line argumentation.

**Index Terms**— *Collaborative Software Architecture Design, Collaborative Decision Support, Intelligent Argumentation, Collaborative Knowledge Management, Web-Based Knowledge Management*

## I. INTRODUCTION

During the design of the architecture for a system, there are many design issues that needs to be addressed. The stakeholders of an organization have their own rationale from various perspectives to resolve design issues. Most of the time their views to solve a design issue conflict with each other. In order to collaboratively examine and resolve an issue, stakeholders must participate in argumentation to exchange their rationale with respect to their design choice.

Argumentation is a process by which stakeholders choose from multiple alternatives to resolve a design issue, giving arguments explaining their support or lack of support for a given design alternative[1]. This provides stakeholders with an equal platform for exchanging their knowledge and viewpoints, allowing them to select an alternative which best address the concerns of the whole. If properly organized and recorded, argumentation provides a format for rationale which gives a comprehensive explanation of the motivations behind design decisions.

We developed an argumentation system that allows stakeholders to post their arguments in support or attack of an alternative that addresses the issue [1]. In order to strengthen the arguments, stakeholders can also post evidences along with their arguments. During this process, as the number of arguments increase, a huge argumentation tree is produced. To compute the design alternative that is favored by most of the stakeholders, the entire argumentation tree is analyzed using argumentation reduction inference engine. The computed result helps the decision makers to select an appropriate alternative to resolve the architecture design issue.

As the architecture of the system evolves, there are many architectural elements and their respective design decisions. So, there must be proper mapping between the elements of the architecture and design decisions. This allows the users to trace from architectural elements to their associated design issues, design alternatives, arguments, evidences, and decisions in an argumentation tree.

This paper is organized as follows. Section II presents a review about the related work, Section III briefs about the intelligent argumentation system that is used for argumentation process, Section IV explains in detail about the framework to capture software architecture rationale using intelligent argumentation and Section V validates the proposed method using a case study.

## II. RELATED WORK

This section presents the literature review of related research. Sub section A presents the work done in field of architectural knowledge management and Sub section B discusses the related work of other argumentation systems.

### A. Architecture Knowledge Management:

Much work is done in field of architectural knowledge management. In one of the approach proposed by Fabian et al [2], design decisions are concrete bindings between requirements and from requirements to their manifestations as model elements in architectural models. The architectural knowledge rationale is maintained as documentation linked to architectural significant requirements. Another approach PAKME [3] captures design alternatives as cases from literature. A design case consists of problem and solution, patterns and tactics used, rationale, and related design options. ADDSS [4] is another system focused on storing the rationale behind design decisions. It captures rationale by linking

motivating factor to design decision made, also stores design pattern knowledge.

Cui et al [5] proposed a design centric architectural design in which stakeholders determine the architectural issues from requirements as well as their solutions. The system explores all feasible combinations of the issue solutions and combines the feasible combinations to generate architectural solution. The rationale is automatically collected from each solution involved in the final architecture. While the rationale includes both pros and cons of a particular solution, it is not presented in great detail. Architecture as design decisions was presented by Lytra et al [6]. The system automates component and constraint generation based on design decisions. Each design decision has a set of outcomes, which are mapped directly to the architectural elements they generate in the component model, allowing for each component to be traced back to its corresponding design decision. Savolainen and Männistö [7] proposed a method of creating architectural views that more prominently communicate the conflicts from multiple perspectives between key stakeholders' concerns. It helps capturing architecture rationale based on interactions among stakeholders. Bratthall et al [8] experiment to verify the importance of design rationale when predicting change impact is mostly related to how to document the architectural rationale and how important is the availability of architectural knowledge to assess the change impact.

### B. Argumentation Systems

There is large amount of work done in collaborative decision making through argumentation. Most of the argumentation systems proposed follow Stephen Toulmin's model of argumentation [9]. The first method was gIBIS [10] which represents design dialog as graph. The method displays arguments, issues and positions in form of graph. HERMES [11] is a computer based decision support tool which organizes arguments and evidences in hierarchy. Chenn-Junn Huang's argumentation system assesses the quality of the arguments by parsing the arguments [12].

The above methods related to architectural knowledge management are not widely adopted in practice. The knowledge management methods discussed above captures only static knowledge of the architecture, not the dynamic exchange of rationale of stakeholders on design issues. The methods fail to capture the knowledge from multiple perspectives. The methods related to argumentation systems are not specific to architectural knowledge management and are difficult to use.

### III. THE INTELLIGENT ARGUMENTATION SYSTEM

We developed Web based Intelligent Argumentation system for collaborative decision support [1]. The stakeholders can participate in the argumentation process using web browser and the server manages different clients simultaneously along with business calculations to determine the stakeholders' favorability for a design issue. Within the argumentation system, the design issue is referred to as an *Issue* and various design alternatives that addresses the issues are called *Positions*. The stakeholders can post their views in terms of

arguments during the argumentation process. The stakeholders should enter a degree of strength explicitly for an argument in range of -1 to 1. An argument with negative degree signifies that it is attacking another argument and argument with positive degree signifies that it is supporting another argument. An argument with 0 degree strength signifies indecisiveness [13]. The priority of the stakeholder is also one of the factors in decision making. So, the stakeholders are assigned priorities based on their role during software design process [15]. Figure 1 shows a snapshot of the intelligent argumentation system

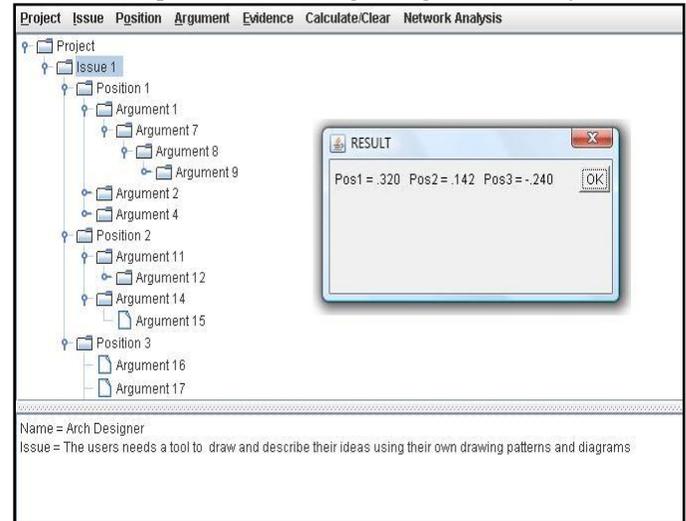


Figure 1 Snapshot of The Intelligent Argumentation System

The entire argumentation tree built during the argumentation process is reduced to a single level such that all arguments posted refer directly to the position. The overall favorability of the design alternative is computed by weighted summation of the argument strengths. In order to assess the impact of the indirect arguments on a position, we have four general argumentation reduction heuristic rules and 25 fuzzy rules are derived from these 4 rules. Please refer to references [1, 14] for more discussion of the intelligent argumentation system. The four heuristic rules are formulated as,

**Argumentation Reduction Rule 1:** If argument B supports argument A and argument A supports position P, then argument B supports position P.

**Argumentation Reduction Rule 2:** If argument B attacks argument A and argument A supports position P, then argument B attacks position P.

**Argumentation Reduction Rule 3:** If argument B supports argument A and argument A attacks position P, then argument B attacks position P.

**Argumentation Reduction Rule 4:** If argument attacks argument A and argument A attacks position P, then argument B supports position P.

Figure 2 shows a sample argumentation tree which represents positions, arguments and their associations. There are two design alternatives for a design issue, Position 1 and Position 2. Arguments A1, A2, A3, and A4 are posted under position 1. The argument A3 attacks position 1 based on an argumentation heuristic rule.

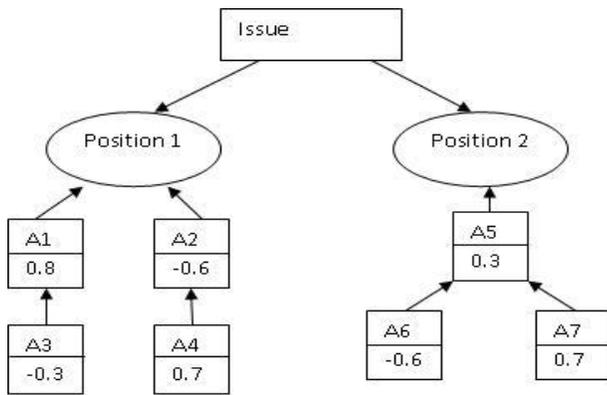


Figure 2 Sample Argumentation tree

#### IV. FRAMEWORK TO CAPTURE SOFTWARE ARCHITECTURE RATIONALE THROUGH ARGUMENTATION

The proposed method captures the software architecture decision rationale through intelligent argumentation. Unlike other methods discussed earlier, where they capture static knowledge of the architecture, our method is dialog based which captures dynamic interactions of the stakeholders who are responsible for the architecture design. It is built on an existing intelligent argumentation system which has its applications in various domains and displayed promising results.

The requirements for a software product are the basis for the development of a system. The product needs to be developed such that it satisfies the requirements of its customers. The software architect design team has to design the architecture of the system before implementation can begin. The architecture design is mainly dependent on the software requirements captured during requirements gathering stage. The problems that the system faces to ensure the satisfaction of the requirements are termed as design issues. These design issues need to be resolved through one or more design decisions. Figure 3 shows the architecture of the present method to capture software architecture rationale knowledge through intelligent argumentation.

The architecture of the proposed method consists of three high level components. The first is the issue based architecture design rationale capture, which is responsible for documenting the rationale behind the design decisions in terms of arguments from various stakeholders for different design alternatives. The argumentation based software architecture design rationale knowledge base shows the linkage between the software architecture, its design issues and their respective rationales. Lastly the inference engine determines design decision by performing inference process on the argumentation tree.

The internal structure of one of the components, the argumentation based software architecture knowledge base is shown in Figure 4. The structure includes software architectural elements and rationale elements such as design issues, design decision, arguments, evidences and design alternatives. It clearly shows the dynamic interactions between the stakeholders to determine the design decision. There exists a complex relationship among various elements. Some of such

relationships include: the design issue has its origin from requirements and architectural elements, the requirements determine the architecture of the system, arguments are posted for a design issue and design alternatives, evidences are attached to arguments, design alternatives solve a design issue and design decision affects the architecture of the system.

Here, the software architecture is considered to be designed using four elements namely components, connectors, configuration and constraints. A component is like a service that provides some required functionality and is derived from functional requirements. A connector is a link between two components of the system to depict the interactions among them. For example: a broadcast connector that can transmit the required information between two components. Constraints determine the qualities that the components or connectors

Argumentation based Software Architecture Rationale Knowledge Management

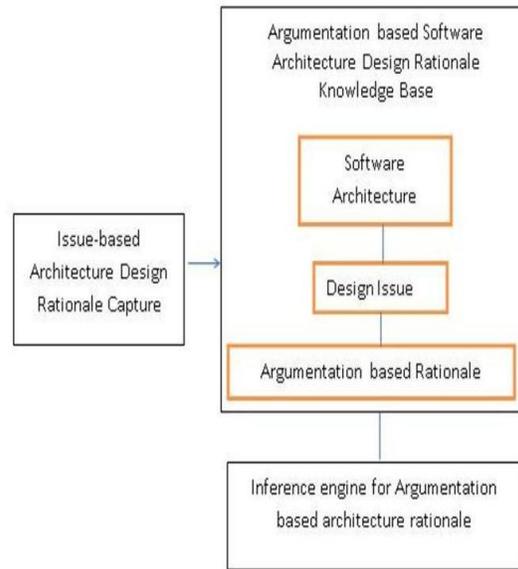


Figure 3 Architecture of Software Architecture Rationale Knowledge Management System

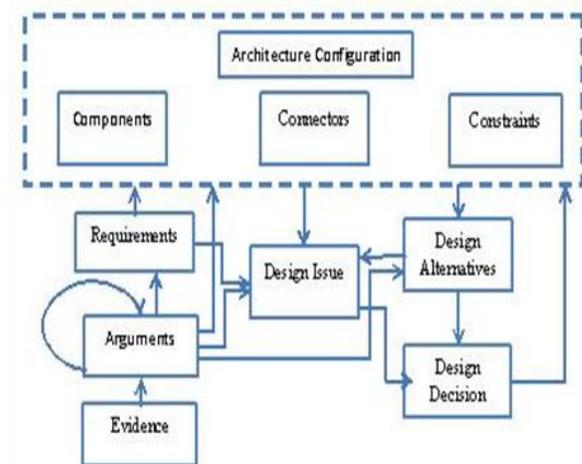


Figure 4 Structure of Architecture Knowledge Base

should satisfy. They are generally attached along with the architectural elements that need to satisfy particular quality Configuration is a collection of components and connectors.

As stated earlier, design issues are derived directly from software requirements. The design decision that addresses the design issue may have its impact on multiple architectural elements. Therefore, once the design decision is taken, the architecture of the system needs to be modified accordingly. There are potentially multiple design alternatives to solve a given design issue. A design alternative consists of a design solution along with the set of architectural elements it affects. Deciding which alternative should be implemented may be a contentious issue. If the conflict among the stakeholders is properly managed, the issues can be resolved to achieve a design decision that best addresses the concern of the whole group and also will have a structured rationale behind the decision. Once multiple design alternatives are suggested the stakeholders debate the pros and cons of each in terms of their ability to satisfy the requirements.

After entire argumentation tree is built, the inference engine determines which design alternative that is favored by most of the stakeholders. This favorability factor helps the stakeholders to decide which design alternative to implement in the architecture design. Each architectural element such as component, connector and constraint maintains a link with the design decision which resulted in their origin and also to the design decision which might have modified afterwards. The captured rationale enables the stakeholders to understand the architecture of the system from requirements to its design.

During software architecture design, the functional requirements define what the system should do and they contribute mainly to the design decisions to be taken. The non-functional requirements (NFR's) are equally important as they define how the system should be in order to perform desired functionality. Each architectural element in system's architecture contributes in some way to the overall satisfaction of the systems NFRs [16]. Often, majority of the time is allocated to resolve design issues pertaining to functional requirements without considering NFR's quoted. The quality of a software system is measured with respect to how well the system architecture is designed in order to satisfy the NFR's. For instance, a system that is designed to meet all the functional requirements but is inefficient and insecure and therefore fails to satisfy the needs of its customers despite meeting their functional requirements. Hence, the system's software architecture must be designed in such way that it should equally satisfy the system level NFR's. However, this task is complicated by the presence of NFR conflicts, where design decisions made to support the resolution of one NFR has a detrimental effect on other NFRs. These NFR conflicts are analyzed with respect to their impact on each architectural element and the conflicts are resolved using the intelligent argumentation system Please refer to [16] for in detail explanation of NFR conflict analysis and detection in case of product lines. Please refer to [17] to understand more about NFR decomposition with respect to functional requirements.

## V. CASE STUDY

In order to demonstrate the effectiveness of proposed method, a case study is performed on the architecture of our own intelligent argumentation system. This case study is performed in a simulated environment where we enacted ourselves as having different roles in software organization and participated in the design decision making process by posting arguments from various perspectives. The purpose of this case study is to capture the rationale behind the design decisions taken to design the architecture of the intelligent argumentation system. During the design of the system's architecture, there were many design decisions made to address various design issues related to requirements which affected multiple architectural elements. One such issue is related to a requirement where users of the system need a tool to draw and describe their ideas using their own drawing patterns. This design issue can be resolved considering various design alternatives. The rationale based on arguments for the aforementioned design issue is captured by nine different stakeholders, each having various roles in software development. A decision is made based on the result provided by the intelligent argumentation system after the inference process.

The intelligent argumentation system is built on Client - Server architecture with distinction among presentation layer, business layer and Data persistence layer. The presentation layer handles user interface (UI) of the system that allows stakeholders to interact with the system and post their arguments. The business logic is handled by business layer where the inference process is performed. The data persistence layer deals with reading and writing the data into the database. This architecture consists of many components which are connected and communicating, through connectors. The architectural elements have their own rationale with respect to the design issues and the system is designed by resolving the issues with the help of intelligent argumentation system.

The stakeholders of the system, each having their own role, had provided their arguments to support or attack their choice of design alternative to address the design issue. Table I shows the various roles of the stakeholders in the organization who participated in the decision making process.

TABLE I. ROLES OF STAKEHOLDERS

S.No	Role ID	Roles	Stakeholder Priority
1	D1	Developer	0.8
2	D2	Developer	0.8
3	U1	Participants/Users of System	0.5
4	U2	Participants/Users of System	0.5
5	U3	Participants/Users of System	0.5
6	AD1	Architecture Designer	0.9
7	T1	Maintainer/Tester	0.7
8	T2	Maintainer/Tester	0.7
9	S1	Sales Representative	0.8

For the design issue mentioned above, there are three design alternatives which can address the issue in their own way, canvas integrated in system, standalone drawing tool, and

desktop sharing. By integrating the canvas into the system, users can draw and share the same with other users without leaving the current session. The standalone drawing tool can also be used to draw patterns but it is not integrated in the current system forces users to toggle between multiple applications. Using desktop sharing, users can expose their drawings to others and share their ideas. With these three design alternatives, the stakeholders participated in the argumentation either supporting or attacking the design alternative or arguments posted by others. One of the components, issue based architecture design rationale capture as explained in the above framework is responsible to capture the stakeholder’s rationale. Table II shows the arguments posted by the stakeholders for one of the winning design alternative, Canvas Integrated in the system. Argument Id A1.1 depicts that this argument is posted under another argument having id A1.

Similarly the stakeholders posted their arguments for the other two design alternatives. After the inference process is carried out based on the stakeholder’s arguments, the system computed that the integrated canvas is the design alternative that is most favored by stakeholders. Based on the computed

decision, a new component canvas is introduced into the architecture of the Intelligent Argumentation System. This new component also interacts with other components in the system. But, there is no direct impact on other architectural elements with the introduction of canvas. For this component to interact with other dependent components, a connector is introduced. Hence, the rationale for the architectural component, Canvas is captured and recorded in the knowledge base. In similar manner, all other design issues are resolved during architecture design and their rationale is recorded within a permanent repository of architectural knowledge. The documented architectural knowledge consists of the business requirement, the design issue, design alternatives, the entire stakeholder’s argumentation and the design decision. The component, software architecture design rationale knowledge base of the framework maintains the link between the architectural elements and their respective knowledge. Figure 5 shows the resulting architecture of the Intelligent Argumentation System. The argumentation system architecture shows three high level components: Client, Server and Database system. These high level components are further classified into lower level components to resolve lower level design issues.

TABLE II. DESIGN ALTERNATIVE (POSITION 1): CANVAS INTEGRATED IN THE SYSTEM

Argument ID	Stakeholder	Argument	Strength of the Argument
A1	D1	An implementation of a simple canvas with basic drawing capability involves less development and design	0.6
A1.1	D2	A simple canvas cannot give users more options to represent their ideas	-0.5
A1.1.1	U2	The users has to draw everything free hand. If we have already developed drawing patterns to draw a square, rectangle or cycle will help the user to spend less time on drawing	0.4
A1.1.1.1	D1	Since, the basis of our system is on argumentation and not on drawing, simple hand drawn images will suffice the requirement to represent the ideas pictorially.	-0.8
A1.1.1.1.1	D2	The sophisticated drawing tool will enhance the representation of ideas and also decreases the human effort to hand draw all the images	-0.4
A2	U1	The canvas is simple to use and very easy for a user to understand and use the same	0.5
A3	D1	Since this canvas can be implemented in line with existing system, there is no extra development needed for integration as well as for sharing among the users	0.9
A3.1	AD1	This also decreases the complexity of the architecture and also it is a light weight application to be implemented	0.4
A4	S1	The use of canvas does not involve any licensing issue and also the existing computer’s hardware is suffice to meet the new requirement	0.5
A5	T2	This can be tested easily as it is part of our existing system and developed from scratch	0.6
A5.1	T1	This involves lot of test cases to be considered and increases the project duration	-0.5

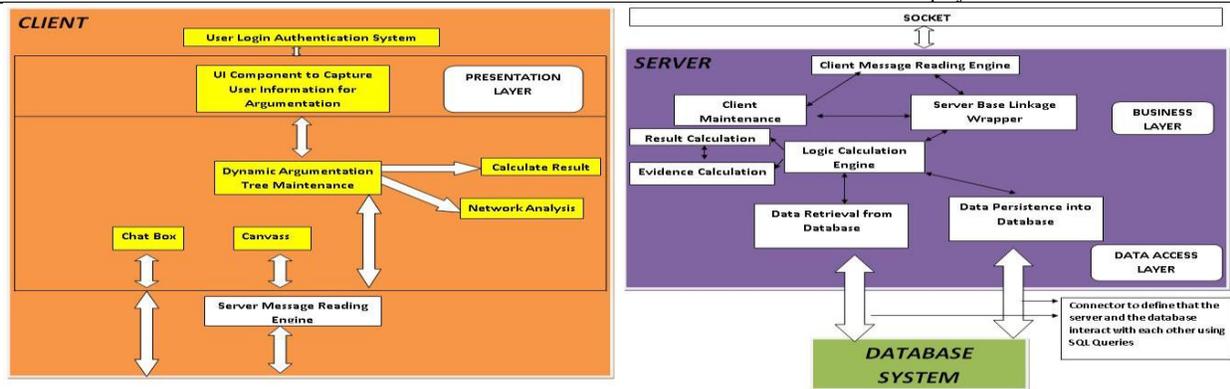


Figure 5 Architecture of Intelligent Argumentation System

## VI. CONCLUSION AND FUTURE WORK

In this paper, we presented a method to resolve the architectural design issues and capture design rationale in the design of software architecture through intelligent argumentation. The design issues are resolved with the help of dynamic interactions among the stakeholders in terms of design issues, design alternatives, arguments, and evidences. The collaborative design decisions aid the system architects to design and modify the architecture of the system. This method captures the rationale in a structured manner and maintains the record of the architectural knowledge. The system is evaluated using a case study to validate the effectiveness of the method. This method is especially valuable in collaborative decision making environment where stakeholders are located geographically across different locations. If multiple products encounter same design issues, then the design knowledge captured for similar products can be retrieved and reused so that significant effort and cost can be saved in software design process. In future, we will develop a method to reuse captured architectural rationale knowledge across products to resolve similar design issues.

## VII. ACKNOWLEDGMENT

We thank the Intelligent Systems Center at Missouri University of Science and Technology for supporting this project.

## REFERENCES

- [1] X. F. Liu, S. Raorane, and M. Leu, "A Web-based Intelligent Collaborative System for Engineering Design," Collaborative product design and manufacturing methodologies and applications. W.D. Li, S.K. Ong, Andrew Y.C. Nee, Chris McMahon. Springer, 2007, pp. 37-58.
- [2] F. Gilson, V. Englebert., "Rationale, Decisions and Alternatives Traceability for Architecture Design" ECSA '11: Proceedings of the 5th European Conference on Software Architecture: Companion Volume.
- [3] M. Babar, I. Gorton., "A Tool for Managing Software Architecture Knowledge" SHARK-ADI '07 Proceedings of the Second Workshop on Sharing and Reusing architectural Knowledge Architecture, Rationale, and Design Intent.
- [4] R. Capilla, F. Nava, J. Montes, C. Carrillo "ADDSS: Architecture Design Decision Support System Tool" ASE '08: Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering.
- [5] X. Cui, Y. Sun, H. Mei, "Towards Automated Solution Synthesis and Rationale Capture in Decision-Centric Architecture Design", Software Architecture, 2008. WICSA 2008. Seventh Working IEEE/IFIP Conference.
- [6] I. Lytra, H. Tran, U. Zdun, "Constraint-Based Consistency Checking between Design Decisions and Component Models for Supporting Software Architecture Evolution" Software Maintenance and Reengineering (CSMR), 2012 16th European Conference.
- [7] J. Savolainen, T. Mannisto, "Conflict-Centric Software Architectural Views: Exposing Trade-Offs in Quality Requirements," Software, IEEE , vol.27, no.6, pp.33,37, Nov.-Dec.2010 doi: 10.1109/MS.2010.139.
- [8] L. Bratthall, E. Johansson, and B. Regnell. 2000. Is a Design Rationale Vital when Predicting Change Impact? A Controlled Experiment on Software Architecture Evolution. In Proceedings of the Second International Conference on Product Focused Software Process Improvement (PROFES '00), Frank Bomarius and Markku Oivo (Eds.). Springer-Verlag, London, UK, UK, 126-139.
- [9] S. E Toulmin, The Uses of Argument. Cambridge, UK, University Press, 1958.
- [10] J. Conklin, M.L. Begeman, gIBIS: A Hypertext Tool for Exploratory Policy Discussion, ACM Transactions on Information Systems, Vol.6, No. 4, (1998) pp. 303-331.
- [11] N. Papadias, HERMES: Supporting Argumentative Discourse in Multi Agent Decision Making". Proc. 15th National Conference on Artificial Intelligence (AAAI-98) (1998) pp. 827-832.
- [12] C. J. Huang et al, Implementation and Performance of an Intelligent Online Argumentation Assessment System. International Conference on Electrical and Control Engineering, (2010) pp. 25-27.
- [13] R.S Arvapally, X. F. Liu, "Collective assessment of arguments in an online intelligent argumentation system for collaborative decision support," Collaboration Technologies and Systems (CTS), 2013 International Conference on , vol., no., pp.411,418, 20-24 May 2013 doi: 10.1109/CTS.2013.6567263.
- [14] X. F. Liu, E. Khudkhudia, L. Wen, V. Sajja, and M. Leu, "An Intelligent Computational Argumentation System for Supporting Collaborative Software Development Decision Making," Artificial Intelligence Applications for Improved Software Engineering Development. F. Meziane and S. Vadera, Hershey, PA: IGI Global, pp. 167-180, 2009.
- [15] X. F. Liu, M. Satyavolu, M.C. Leu, "Contribution based priority assessment in a web-based intelligent argumentation network for collaborative software development," Collaborative Technologies and Systems, 2009. CTS '09. International Symposium on , vol., no., pp.147,154, 18-22 May 2009 doi: 10.1109/CTS.2009.5067475.
- [16] X. F. Liu, E.C Barnes, J. Savolainen: Conflict detection and resolution for product line design in a collaborative decision making environment. CSCW 2012: 1327-1336.
- [17] V. Sadanam, X. F. Liu, Analysis of Conflicts among Nonfunctional Requirements using Integrated Analysis of Functional and Non-functional Requirements. 31<sup>st</sup> Annual International Computer Software and Applications Conference, 24-27 (2007), pp. 215-218.
- [18] N. Niu, L.D. Xu, J.-R.C Cheng, Z. Niu, "Analysis of Architecturally Significant Requirements for Enterprise Systems," *Systems Journal, IEEE* , vol.99, no.99, pp.1,1, 0 doi: 10.1109/JSYST.2013.2249892

# An Approach for Capturing and Documenting Architectural Decisions of Reference Architectures

Milena Guessi  
Dept. of Computer Systems  
University of São Paulo - USP  
São Carlos, Brazil  
Email: milena@icmc.usp.br

Flavio Oquendo  
IRISA Research Institute  
University of South Brittany  
Vannes, France  
Email: flavio.oquendo@irisa.fr

Elisa Yumi Nakagawa  
Dept. of Computer Systems  
University of São Paulo - USP  
São Carlos, Brazil  
Email: elisa@icmc.usp.br

**Abstract**—During the design of software architectures of software systems, it is widely known the relevance of capturing and documenting architectural decisions, i.e., reasons, implications, justification, and trade-offs related to choices made in these architectures. Therefore, it is possible to achieve a more complete documentation of software architectures that also contributes to the success of the software systems. In parallel, reference architectures have become a quite relevant element to the successful construction, evolution, and standardization of software systems. Although architectural decisions are also important for reference architectures, they have not been sufficiently addressed in the context of such architectures. In particular, the use and dissemination of reference architectures may be jeopardized due to lack of appropriate architectural decisions. The main contribution of this study is to present a systematic approach for capturing and documenting architectural decisions of reference architectures. In order to assess the feasibility of our proposal, a case study is presented showing how our approach can be used for enhancing the architecture description of such architectures. With this, we intend to bring attention to the importance of architectural decisions for reference architectures and, as a consequence, to contribute to a more complete documentation, dissemination, and effective use of such architectures.

**Keywords**—Reference Architecture, Architecture Decision, Architectural Decision, Architecture Description.

## I. INTRODUCTION

Software architectures play a determinant role for software systems qualities such as maintainability, dependability, and interoperability, as previously stated by Wasserman [1]. But, most of the success of software architectures relies on its architecture description, which encompasses the set of artifacts expressing the architecture and has several applications, e.g., as basis for system design and development activities, as input to verification and validation automated tools, and as documentation [2].

The documentation of architectural decisions is an important part of architecture descriptions. For example, architecture decisions support the architectural description by recording reasons, implications, justifications, and trade-offs for the most relevant architectural choices. As a result, documenting architectural decisions can add a great value to the software architecture in a medium and long term by preventing knowledge vaporization. Different approaches exist for managing architectural knowledge, e.g., pattern-centric (which aims at establishing a shared vocabulary of reusable, abstract solutions); dynamism-centric (which uses ADLs to codify explicit

architectural knowledge into models that can be accessed during runtime); requirements-centric (which intends to bridge the gap between architecture and requirements); and decision-centric (which prevents knowledge vaporization by capturing the reasoning leading to the software architecture) [3].

In the same perspective, we also observe the proposal of reference architectures, which encompass the *knowledge* about how to design software architectures of a given application domain. To do so, reference architectures address business rules, architectural styles, best practices of software development (e.g., architectural decisions, domain constraints, legislation, and standards), and software elements that support the development of software systems in a given domain [4]. From this, we observe that reference architectures integrate most approaches for architectural knowledge management. Therefore, the documentation of architectural decisions for reference architectures is certainly a prerequisite for their effective role as communication vehicles throughout software systems life cycle. Good examples of reference architectures can be found in the literature, such as AUTOSAR (Automotive Open System ARchitecture) [5] for the automotive domain, and UniversAAL [6] and Continua [7] for Ambient Assistent Living (AAL) domain.

Even though AUTOSAR is a very complete architecture resulting from several years of research and effort, architectural decisions about the design of the reference architecture are only partially addressed in its documentation, since it lacks rationale and considered trade-offs for example [8]. In fact, to our best knowledge, there is no commonly used approach addressing architectural decisions in reference architectures. If at least the most important decisions were reported, it would be easier to understand why things are as they are in the reference architecture. Moreover, since reference architectures differ from concrete software architectures — e.g., they are more generic and often described at a higher abstraction level, have no clear stakeholders, involve more architectural qualities, and have a larger scope — not all approaches used to describe software architectures are also adequate for reference architectures. In this scenario, several approaches have been proposed for building reference architectures. For example, Muller (2008) [9] provides guidelines for establishing reference architectures. Nakagawa et al. (2014) [10] presents a process to establish, represent, and evaluate reference architectures with focus on improving the separation of concerns in such architectures. Angelov et al. (2012) [11] proposed a classification frame-

work for reference architectures that aims at promoting the analysis and design of reference architectures by assessing their adherence to one of these reference architectures types. Nonetheless, Nakagawa et al. (2012) [12] were the first to introduce the necessity of documenting architecture decisions in the development of reference architectures. However, it is still necessary to determine how this knowledge should be codified and stored in reference architectures, which must be defined by specific methods and architecture frameworks.

In this scenario, the main contribution of this paper is to present an approach for capturing and documenting architectural decisions during the design of reference architectures. We also present a case study in order to show how our approach can contribute to the description of such architectures. As a result, we have observed the importance of capturing architectural decisions during the design of reference architectures as a means to achieve a more complete documentation of such architectures that certainly contributes to their dissemination and effective use. Thus, we also intend this paper motivates further investigations on incorporating an architectural decisions documentation activity throughout reference architectures' life cycle.

The remainder of this paper is organized as follows. In Section II, we introduce the concept of reference architectures and discuss current approaches for documenting software architectures. In Section III, we present our approach for integrating architectural decisions in the description of reference architectures. In Section IV, we discuss the results from a case study regarding the architectural description of reference architectures. In Section V, we discuss future research perspectives. Finally, we present our final remarks in Section VI.

## II. BACKGROUND

Several terms are used to designate software architectures in different abstraction levels. In particular, the term reference model is frequently misused as a synonym for reference architecture. A reference model designates a structure that promotes the understanding on a given domain by sharing a common vocabulary and the parts and its interrelationships without considering implementation details [13]. The OASIS Reference Model for Service Oriented Architectures<sup>1</sup> is an example of reference model. Conversely, reference architectures are less abstract than reference models as they provide concrete guidelines for the design of software architectures, such as architectural styles, best practices for software development, and software elements supporting the development of systems. In particular, reference architectures can be derived from the combination of reference models and architecture patterns (see Figure 1). Therefore, different concrete software architectures can be derived from reference architectures.

The architecture description is the main artifact for accessing software architectures. One of the main goals of an architectural description is to present sufficient detail about the software architecture in order to enable its analysis against architectural requirements. Given the relevance of architecture descriptions, the ISO/IEC/IEEE 42010 standard [2] disseminates best practices for creating such artifact. As proposed by

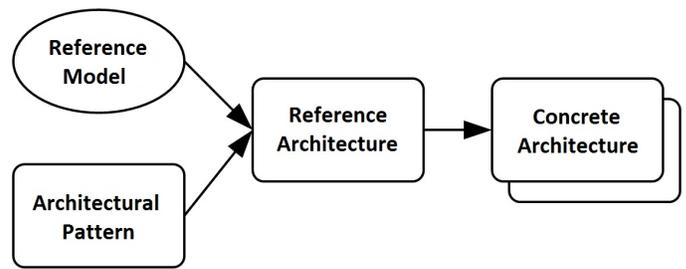


Fig. 1. Relationship among reference model, architectural pattern, reference architecture, and concrete architecture [13]

this standard, the usage of multiple architectural viewpoints is a common practice for documenting architectures of software systems. Architectural viewpoints establish the conventions for the construction, interpretation, and use of architectural views framing a particular set of concerns from system stakeholders [2]. As each architectural view conforms to a specific viewpoint, each view provides a particular representation for the system elements and the relationships existing among them [2]. Several viewpoints, such as logical, runtime, data, and physical viewpoints, have been proposed for the description of software architectures [14]. In this context, we can also observe the proposal of architecture frameworks, i.e., a set of viewpoints which establish a common practice for describing the software architecture of a particular domain or stakeholder community, such as “Views and Beyond” [15] and “4+1 Views” [16]. Motivated by the success of architectural viewpoints in the representation of software architectures, architectural viewpoints have also been investigated for the representation of reference architectures [17], [18].

Architectural Description Languages (ADLs) have been proposed for describing software architectures, such as Wright [19] and Rapide [20] which are based on formally defined syntax and semantics. But, even though formal ADLs provide automatic mechanisms for validation and verification of architectural models, they can be difficult for non-technical stakeholders to understand. Therefore, semi-formal languages have been frequently used for describing software architectures as they combine the rigor of formal languages to the understandability of natural languages. For example, UML (Unified Modeling Language) [21] and SysML (System Modeling Language) [22] are semi-formal languages that can be used for describing software architectures. In particular, SysML introduces a Requirements Diagram which captures the hierarchy among requirements and can be useful for guaranteeing traceability among them, and a Parametric Diagram which describes constraints to system property values, such as performance, confidence, or weight, and can be useful for integrating specifications and models of the system into engineering analysis models.

In another context, the main goal of architectural knowledge management is to prevent knowledge vaporization in software architectures. To do so, architecture rationale for significant architectural decisions made in the software architecture should also be included in the architecture description. Significant architectural decisions could be for example the selection of a particular concern or viewpoint, the definition of the most adequate abstraction level for a particular view-

<sup>1</sup><https://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf>

point, or the reason for adopting a particular design pattern. Furthermore, several aspects of an architectural decision can be relevant, such as their implications to the software architecture design, constraints and rules imposed by them, and also the reasoning that lead to them [23]. Therefore, architectural decisions play an important role for education, reuse, and evolution of software architectures as they are used for sharing expertise and best practices. In the context of reference architectures, significant architectural decisions encompass guidelines for deriving the reference architecture into concrete software architectures besides documenting the architectural knowledge of concrete software architectures of a given domain. Thus, reference architectures certainly need to address architectural decisions in their architectural description.

### III. INTEGRATING ARCHITECTURAL DECISIONS IN THE DESCRIPTION OF REFERENCE ARCHITECTURES

We included an activity for documenting architectural decisions into the design process of reference architectures defined by Nakagawa et al. (2014) [10]. The first step for designing the reference architecture is to outline its scope which consists in the identification of relevant stakeholders of the reference architecture and their concerns. The scope of the reference architecture guides the selection of a set of viewpoints that frame those concerns. The next activity of this process comprises the creation of selected viewpoints. We propose the documentation of the most relevant architectural decisions in parallel to all design activities. In particular, we propose the creation of a specific repository for architectural decisions which may be directly mapped to other architectural description artifacts (e.g., viewpoints, architecture models). The architecture decisions repository can also be understood as another viewpoint in the reference architecture description. We defined these steps inside a loop to indicate that the reference architecture should be built iteratively. At the end of each iteration, there is an evaluation activity for validating the completeness and/or effectiveness of the architectural description related to the set of concerns defined in the beginning of the loop. This evaluation should be developed by relevant stakeholders and developers of the reference architecture. Moreover, the evaluation approach used depends on the chosen ADL. If this evaluation is not successful, all previous activities should be revised in order to address identified flaws. Finally, a consensus must be reached among the most important stakeholders regarding the necessity of more iterations. Subsequent iterations of this process can include more details, concerns, viewpoints, or stakeholders to the architectural description. This process is described using SPEM in Figure 2.

In our approach, we used the architecture framework for architectural decisions proposed by Heesch et al. (2012) [24]. This framework is composed by four viewpoints, namely: a Decision Detail viewpoint showing information about individual decisions; a Decision Relationship viewpoint showing the relationship between architectural design decisions and their current state in a particular moment in time (e.g., idea, decided, rejected, approved); a Decision Chronology viewpoint showing all versions of an architectural decision, i.e., all states that it assumes over time; and a Decision Stakeholder Involvement viewpoint showing stakeholders' responsibilities in the decision-making process, which is important to personalize architectural knowledge, i.e., documenting who knows what.

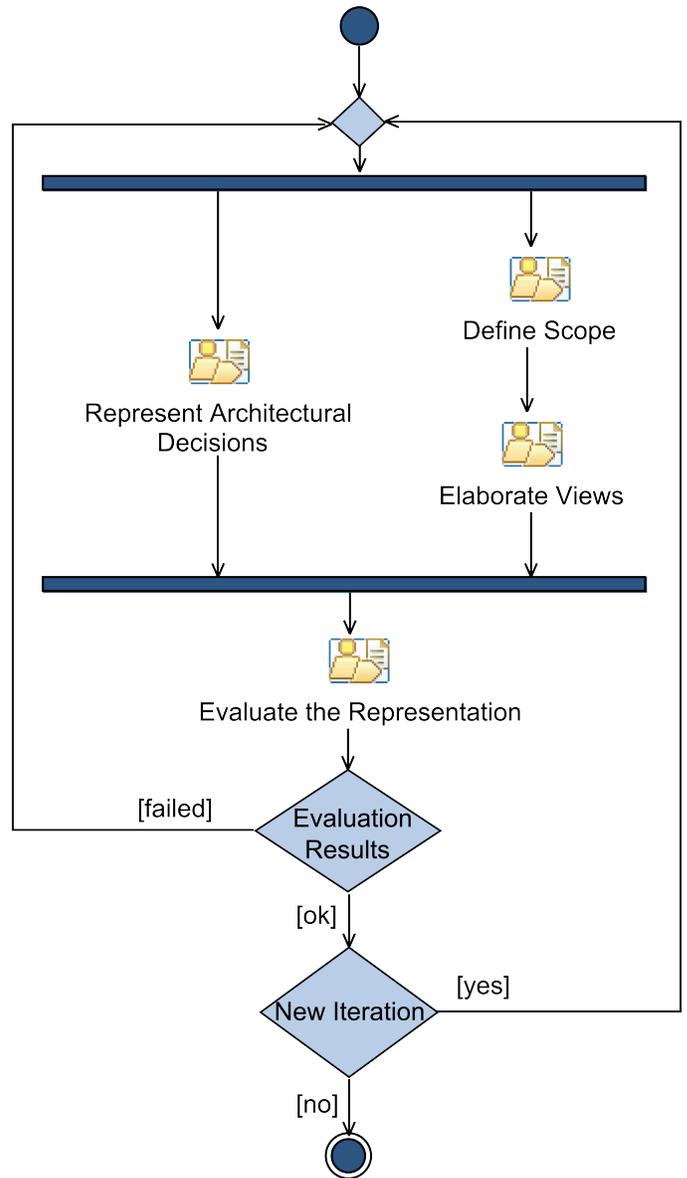


Fig. 2. Approach for documenting reference architectures. In this approach, architectural decisions are documented in parallel to the creation of architecture views

In their study, the authors use UML for creating most of these viewpoints: the decision detail viewpoint only presents a textual representation (detailed in Table I), the design decision relationship and stakeholders involvement apply the use case diagram, and the chronology viewpoint shows design decisions in a state machine diagram using special stereotypes for states and associations. In our approach, we experimented the use of SysML techniques as it is an OMG standard which has also been suggested as an evolution of UML. In particular, SysML has a requirements diagram that can be useful for describing architecture decisions. Moreover, SysML has already been used for describing reference architectures [25]. Therefore, we propose the use of the requirements diagram for complementing the textual representation in the decision detail viewpoint, the block definition diagram and the parametric diagram in the design decisions relationship viewpoint, the state machine

TABLE I. TEMPLATE FOR THE DECISION DETAIL VIEWPOINT [24]

<b>Name</b>	A short name of the decision that serves as key in the other viewpoints.
<b>Current state</b>	Indicates the current state of the decision.
<b>Decisions groups</b>	A decision can be associated to one or more groups, which share specific characteristics. Decisions could for instance be grouped by subsystem, architecture team who made the decision, or quality attribute requirements.
<b>Problem/issue</b>	The circumstances under which the architect felt the need to make a decision among one or more alternatives. In other words, the issue addressed by the decision.
<b>Decision</b>	The outcome of the decision. In other templates this element is called solution.
<b>Alternatives</b>	The alternative solutions considered when making the decision.
<b>Related decisions</b>	All decisions that have a relationship to the decision.
<b>Related systems concerns</b>	List of all related concerns tackled in this decision.
<b>History</b>	The history of the described decision. The history contains all state changes, e.g., when the decision was proposed, decided, approved, and so on.

diagram in the decision chronology viewpoint, and the use case diagram in the decision stakeholders involvement viewpoint.

#### IV. CASE STUDY

A case study was conducted to evaluate an architectural description for a reference architecture built with our approach. Our purpose with this study was to access the capability of the new architectural description in transmitting knowledge about the reference architecture design. In order to do so, we created an alternative description for the Situated Multiagent Reference Architecture proposed by Weyns (2006) [26]. This reference architecture is composed by:

- one or more Agents, which are autonomous problem solving entities that can dynamically control their own behavior (i.e., perceive and actuate in the environment);
- an Application Environment, which is the medium for agents interaction, such as sharing information and coordinating behavior, that has to be designed for each concrete multiagent system; and
- a Deployment Context, comprised by the hardware, software, and other external resources with which the system interacts (e.g., sensors, actuators, printer, network, database, web service).

In the original architecture description, architecture decisions were presented throughout the reference architecture documentation. In our new architecture description for this reference architecture, we centralized this information in a separate viewpoint, named architecture decisions detail view. We precisely followed the original architectural description in order to guarantee that no errand information was introduced in the new architectural description. As a result, absent information in the original description, such as history, decisions groups, and related decisions, could not be included in the new architecture description. Table II presents a fragment of the architecture decisions detail view in the new architectural description.

Overall, four subjects participated in our case study. The subjects were separated in two groups and each group an-

TABLE II. EXAMPLE OF AN ARCHITECTURAL DECISION DETAIL IN OUR CASE STUDY

<b>Name</b>	Abstract from Deployment Context
<b>Problem/issue</b>	Depending on specific application requirements, distribution can take different forms. For a distributed application, the deployment context consists of multiple processors deployed on different nodes that are connected through a network.
<b>Decision</b>	Abstract from Deployment Context
<b>Alternatives</b>	The same instance of the Application Environment subsystem could be deployed on each node, or specific instances are designated to different nodes, e.g., when different types of agents are deployed on different nodes
<b>Related systems concerns</b>	Variability

alyzed one of the versions of the architecture description. To avoid biasing the evaluation, subjects were not aware of the existence of a second description. The original version of the architecture description is referred to as “original” while the new architecture description created for this case study is referred to as “new”. All subjects received a link to an on-line questionnaire about the reference architecture design including their perception on the architecture description (e.g. facility for accessing a given information, positive and negative aspects of the documentation) and their background on the topics covered in the reference architecture. It is relevant to mention that none of the subjects had prior knowledge on this particular reference architecture and they all spent a couple of hours in their review. In this case study, we focus on the subjects’ overall comprehension about architectural decisions in the reference architecture. In particular, we inquired our subjects about the reasons that motivated the creation of a current knowledge repository. We also asked our subjects to grade from 1 (easy) to 4 (difficult) their difficulty level for answering this question. Table III shows the subjects answers and their perceived difficulty level. Next, we compare subjects’ performance in both architectural descriptions versions. From this table, it is possible to observe that both groups experienced some difficulty in answering this question although there was a small advantage for the original architecture description.

TABLE III. SUBJECTS PERFORMANCE ON THE TEST

Version	New		Original	
Subject	#1	#2	#3	#4
Answer	Correct	Don't know	Wrong	Wrong
Difficulty	2	4	2	1

Subjects that analyzed the new architecture description also informed their impressions regarding three aspects of the architecture decision detail view: (i) usefulness, related to the contribution of the view to understand the reference architecture design; (ii) effort, related to the amount of time required to analyze or scan the view for a particular information; and (iii) clarity, related to the amount of time required to understand a particular information in the view. Subjects could grade each aspect from 1 (low) to 4 (high). Table IV presents the subjects answers. We observe that the subjects perception on the usefulness of this view were opposite, but only Subject 1 answered the question correctly. In addition, Subject 1 considered the architectural decisions detail view useful but time consuming, as the whole section needed to be linearly scanned. Since the clarity of this view was poorly evaluated by both subjects, we intend to further investigate the use of SysML and other ADLs in the construction of this view.

TABLE IV. SUBJECTS PERCEPTION ON THE ARCHITECTURE DECISION DETAIL VIEW

Subject	#1	#2
Usefulness	4	1
Effort	4	3
Clarity	1	2

To better understand the factors that could have played an important role in this case study, we also asked the subjects about their personal experience on the topics covered in the reference architecture description such as UML, SysML, and Embedded Systems. Table V summarizes subject’s background. We observe that all four subjects present solid knowledge on Software Engineering as three of them are doctorate students and one of them is a post-doctorate student in the field. We also observe that both groups present similar knowledge on most of the topics. In particular, the lack of a solid SysML did not play a determinant role in the analysis of the architecture decision detail view since it presented textual descriptions. However, this factor should certainly be considered in a future evaluation of this view that also uses SysML techniques.

TABLE V. SUBJECTS BACKGROUND ON THE COVERED TOPICS IN THE ARCHITECTURE DESCRIPTION

Version	New		Original	
	#1	#2	#3	#4
Subject				
UML	4	4	4	3
SysML	2	1	3	2
Embedded System	4	4	4	4
Software Architecture	4	2	3	3
Reference Architecture	4	2	3	3

We observed that the group evaluating the new architecture description was more successful since none of the subjects in the other group answered this question correctly. The fact that architecture decisions were scattered in the other views of the original description might explain why subjects evaluating it did not find the correct answer. Therefore, our approach of presenting architecture decisions in a particular repository of the reference architecture seems interesting. Nonetheless, our results also point out that more investigation is certainly needed in order to improve the clarity of architecture decisions. To do so, we will investigate ways for mapping architecture decisions to other elements of the description. Finally, it is important to highlight that this study was supervised by experts who also validated the conclusions drawn from our results.

## V. BRIEF DISCUSSION AND PERSPECTIVES OF RESEARCH

Reference architectures have increasingly importance in the development of a set of software systems while architectural decisions provide the necessary information for deriving the reference architecture. Therefore, architecture decisions should be addressed in parallel to the reference architectures’ creation, maintenance, and evolution for effectively documenting and using the knowledge contained in reference architectures. In spite of the positive results achieved in this work, it is still necessary to conduct additional case studies and/or experiments in order to achieve more evidences on the relevance of our approach. It is also interesting to use our approach in reference architectures of the industry context.

Motivated by the results of our case study and by previous experience in the reference architecture research area, we out-

line some important research opportunities involving reference architectures and architectural decisions, such as:

- Systematizing the documentation of architectural decisions in reference architectures processes. By taking architecture decisions as an intrinsic part of the development of reference architectures, it will be possible to prevent knowledge vaporization in reference architectures and to create concrete ways for accessing this knowledge in the future. Motivated by Jansen and Bosch (2005) [27] and Kruchten et al. (2009) [28] current processes for creating reference architectures should certainly be revised to add specific guidance in this direction;
- Enabling traceability among architecture decisions and other viewpoints in the architecture description. Architecture decisions may crosscut several viewpoints as they can be related to logical elements (e.g., classes, packages, systems), physical elements (e.g., machines, software installed on these machines, network connections), or functional properties of the reference architecture. In this sense, enabling traceability from architecture decisions to other artifacts in the architectural description provides interesting insights about the reference architecture internals (e.g., dependencies, considered trade-offs, alternatives). To do so, it is necessary to extend architecture decisions with dependencies, constraints, and relationships;
- Adapting ADLs for documenting architecture decisions in reference architectures. Currently, most architecture decisions have been informally described as only a textual explanation is presented and the relationship to other artifacts in the architecture description is indirect. In this sense, the use of formal or semi-formal ADLs can standardize and automatize the documentation of architecture decisions. The automatic support provided by formal ADLs ultimately facilitates the recovery of architecture decisions from knowledge repositories, the reuse of architecture decisions in concrete instances of the reference architecture, and the automatic analysis of their dependencies. Nonetheless, UML and SysML still do not consider architectural decisions as first class entities. Although we used SysML in our case study, it is still necessary to investigate which are the best techniques for codifying architectural decisions into architecture description elements;
- Introducing architecture decisions variability in reference architectures. The variability in reference architectures concerns the ability of a software artifact built from such architectures to be adapted for a specific context in a preplanned manner [30]. In this sense, adding variability to architecture decisions would help to create an even larger knowledge repository as all alternatives, dependencies, and options would be registered in the reference architecture. Moreover, deriving the reference architecture into concrete software architectures could be resumed to the selection of architecture decisions. To do so, it is necessary to investigate in which ways architecture decisions can

be tailored from reference architectures to concrete software architectures. In this sense, an approach that adds variability to architectural decisions in software architectures, such as the one proposed by Alebrahim and Heisel (2012) [31], could be adapted for reference architectures.

Thus, several interesting opportunities exist regarding architectural decisions in the context of reference architectures. However, each aspect of these topics needs to be investigated in broader and deeper ways; for instance, through qualitative and quantitative evaluations and experimental studies in industry.

## VI. CONCLUSIONS

Only through a careful documentation of architectural decisions, it will be possible to have well-documented reference architectures. The main contribution of this study is to propose an approach for documenting architectural decisions in reference architectures. We motivated our proposal with results from a case study. We observed that addressing architectural decisions in the description of reference architectures increases their potential for better communicating and disseminating knowledge contained in these architectures. It is important to say that since reference architectures differ from concrete software architectures, it is still necessary to continue the investigation on this topic. As several research lines need to be addressed yet, we intend to collaborate to the reference architecture community, as well as to the software engineering community, by providing means for better capturing and documenting architecture decisions in reference architectures.

## ACKNOWLEDGMENT

This work is supported by São Paulo Research Foundation (FAPESP) (Grant: 2012/24290-5 and 2011/23316-8), CNPq, and Capes.

## REFERENCES

- [1] A. I. Wasserman, "Towards a discipline of software engineering," *IEEE Software*, vol. 13, no. 6, pp. 23–31, 1996.
- [2] ISO/IEC/IEEE 42010:2011, *Systems and software Engineering — Architecture Description*, Std., 2011.
- [3] R. Farenhorst and R. C. de Boer, "Knowledge Management in Software Architecture: State of the Art," in *Software Architecture Knowledge Management Theory and Practice*, M. A. Babar, T. Dingsyr, P. Lago, and H. van Vliet, Eds. Springer, 2009, pp. 21–38.
- [4] E. Y. Nakagawa, P. O. Antonino, and M. Becker, "Reference Architecture and Product Line Architecture: A Subtle But Critical Difference," in *ECSA'2011*, Essen, Germany, 2011, pp. 207–211 (LNCS v. 6903).
- [5] AUTOSAR. (2013) AUTOSAR (AUTomotive Open System ARchitecture). [On-line]. <http://www.autosar.org/> (01/21/2013).
- [6] UniversAAL Project. (2011) The UniversAAL Reference Architecture. [On-line]. <http://www.universaal.org/images/stories/deliverables/D1.3-B.pdf> (01/21/2013).
- [7] Continua Health Alliance, "Continua Health Alliance," [On-line], *World Wide Web*, 2014, in <http://www.continuaalliance.org/> (Access in 02/10/2014).
- [8] E. Y. Nakagawa, M. Becker, and J. C. Maldonado, "A Knowledge-based Framework for Reference Architectures," in *SAC'2012*, Riva del Garda, Italy, 2012, pp. 1197–1202.
- [9] G. Muller. (2008) A Reference Architecture Primer. [On-line]. The Netherlands. <http://www.gaudisite.nl/ReferenceArchitecturePrimerPaper.pdf> (09/30/2013).
- [10] E. Y. Nakagawa, M. Guessi, J. C. Maldonado, D. Feitosa, and F. Oquendo, "Consolidating a process for the design, representation, and evaluation of reference architectures," in *WICSA'2014*, Sydney, Australia, 2014, pp. 1–10.
- [11] S. Angelov, P. Grefen, and D. Greefhorst, "A framework for analysis and design of software reference architectures," *Information and Software Technology*, vol. 54, pp. 417–431, 2012.
- [12] E. Y. Nakagawa, F. Oquendo, and M. Becker, "RAModel: A Reference Model for Reference Architectures," in *WICSA & ECSA'2012*, Helsinki, Finland, 2012, pp. 297–301.
- [13] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*. Addison-Wesley, 2003.
- [14] P. Kruchten, *The Rational Unified Process: An Introduction*, 2nd ed. Addison, 2000.
- [15] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, P. Merson, R. Nord, and J. Stafford, *Documenting Software Architectures: Views and Beyond*, 2nd ed. Addison-Wesley, 2011.
- [16] P. Kruchten, "Architectural Blueprints - The "4+1" View Model of Software Architecture," *IEEE Software*, vol. 12, no. 6, pp. 42–50, 1995.
- [17] E. Y. Nakagawa and J. C. Maldonado, "Reference Architecture Knowledge Representation: An Experience," in *SHARK'2008*, Leipzig, Germany, 2008, pp. 51–54.
- [18] M. Guessi, L. B. R. Oliveira, and E. Y. Nakagawa, "Representation of Reference Architectures and Reference Models: A Systematic Review," in *SEKE 2011*, Miami, EUA, 2011, pp. 1–4.
- [19] R. J. Allen, "A Formal Approach to Software Architecture," Ph.D. dissertation, Carnegie Mellon University, 1997.
- [20] D. C. Luckham, "Rapide: A Language and Toolset for Simulation of Distributed Systems by Partial Orderings of Events," in *POMIV'1996*, ser. DIMACS, vol. 29, New Jersey, USA, 1996, pp. 1–25.
- [21] OMG, *Unified Modeling Language v. 2.3*, [On-line], Std., Rev. 2.3, 2010, <http://www.omg.org/spec/UML/2.3/> (01/21/2013).
- [22] —, *Systems Modeling Language v 1.2*, [On-line], Std., Rev. 1.2, 2010, <http://www.omgsysml.org/> (01/21/2013).
- [23] J. Bosch, "Software architecture: the next step," in *EWSA'2004*, 2004, pp. 194–199 (LNCS v. 3047).
- [24] U. van Heesch, P. Avgeriou, and R. Hilliard, "A documentation framework for architecture decisions," *Journal of Systems*, vol. 85, pp. 796–820, 2012.
- [25] M. Guessi, "Subsidies for representation of embedded systems reference architectures," Master's thesis, University of So Paulo, 2013.
- [26] D. Weyns, "An Architecture-Centric Approach for Software Engineering with Situated Multiagent Systems," Ph.D. dissertation, Katholieke Universiteit Leuven, Belgium, 2006.
- [27] A. Jansen and J. Bosch, "Software Architecture as a Set of Architectural Design Decisions," in *WICSA'2005*, Pittsburgh, USA, 2005, pp. 1–10.
- [28] P. Kruchten, R. Capilla, and J. C. Dueas, "The Decision View's Role in Software Architecture Process," *IEEE Software*, pp. 36–42, 2009.
- [29] E. Y. Nakagawa, E. F. Barbosa, M. L. Fioravanti, and J. C. Maldonado, "ProSA-RA: A process for the design, representation, and evaluation of aspect-oriented reference architectures," *Journal of Systems and Software*, pp. 1–40, 2011, (in review).
- [30] M. Galster, P. Avgeriou, D. Weyns, and T. Mnnist, "Variability in software architecture: current practice and challenges," *SIGSOFT Software Engineering Notes*, vol. 36, no. 5, pp. 30–32, 2011.
- [31] A. Alebrahim and M. Heisel, "Supporting Quality-driven Design Decision by Modeling Variability," in *QoSA 2012*, Bertinoro, Italy, 2012, pp. 1–6.

# Towards a Tactic-Based Evaluation of Self-Adaptive Software Architecture Availability

Alireza Parvizi-Mosaed<sup>1</sup>, Shahrouz Moaven<sup>2</sup>, Jafar Habibi<sup>3</sup>, Abbas Heydarnoori<sup>4</sup>  
Sharif University Of Technology

Tehran, Iran

{aparvizi<sup>1</sup>, moaven<sup>2</sup>}@ce.sharif.edu, {jhabibi<sup>3</sup>, heydarnoori<sup>4</sup>}@sharif.edu

**Abstract**— nowadays, several non-automatic or semi-automatic software architecture evaluation methods have been proposed to evaluate their quality attributes as availability. In spite of their applicability, they are not effective in self-adaptive software architectures due to their off-line properties; e.g., scenario-based methods. Since the architectural tactics provide a bridge between architectural designs and quality attributes, they have sufficient potential to resolve this problem. In this paper, we assume that the software architecture is completely composed of some architectural patterns. Then we propose an automated evaluation method which composes the architectural tactics and the patterns to measure the availability of software architectures. In this method, the composition of a few availability tactics and patterns are simulated with appropriate probability distribution functions. To predict the availability of patterns, a data mining approach is applied to these simulated models to generate training models for each combination of tactics and patterns. Furthermore, a utility function is defined to compute the availability of systems by these models in  $O(n)$  where  $n$  is the number of patterns of systems. This method improves the data gathering and analysis activities of the SASSY (Self-Architecting Software SYstems) framework. To validate our method, we have applied it to the Rapidminer case study.

**Keywords**- Availability, Self-Adaptive Architecture, Architectural Tactic, Architectural Pattern, Data Mining.

## I. INTRODUCTION

Quality attributes are the best criteria for evaluating the quality of software architectures [1]. Even though quality management is an umbrella activity in the software development process, its cost is different from one level of modeling to another. In other words, the cost of the quality management activity will be increased whenever the models become more detailed (e.g., moving from architectural models to design models). Therefore, architectural models enable us to evaluate quality attributes with lower costs [2].

The architecture evaluation methods are categorized as early or late methods to measure the quality of systems at the architectural level. In early methods, architectures are evaluated before the implementation step in the software development process, whereas in late methods, this process is postponed to test or execution times [3]. Architectural tactic composition is a useful evaluation method as it provides a bridge between the architectural design and quality attributes to predict, control, and satisfy the quality of software architectures [4]. This method has sufficient capability to provide an early or late method when it is merged with scenario-based, experience-based, or simulation-based

methods; e.g., ATAM (Architecture Tradeoff Analysis Method) is a scenario-based evaluation method which is improved by architectural tactics [2].

The best advantage of the tactic composition methods is highlighted in self-adaptive software architectures due to their dynamic and automatic properties. These systems are usually mapped to a composition of architectural patterns; e.g., SASSY (Self-Architecting Software SYstems) is a self-architecting framework which applies an appropriate pattern composition to the software architecture in order to maintain the quality of SOA (Service Oriented Architecture) [5, 6]. Hence, the pattern and tactic composition methods are appropriate methods for evaluating the quality of self-adaptive software architectures.

Although various architecture evaluation methods have been proposed recently [3], no tactic-based automated methods have been presented to predict the availability of self-adaptive architectures. In this paper, the composition of architectural tactics and patterns is simulated by taking advantage of Probability Distribution Functions (PDFs) and the queuing theory [7] to resolve the aforementioned problem. Due to the complexity of these simulations, there is no mathematical formula to compute its availability. Thus, numerous scenarios are applied to these simulations to create a dataset. This dataset is then used to predict the availability of patterns by employing a data mining technique.

It is supposed that components send or respond messages with the Gaussian Probability Distribution Function (GPDF); e.g., while clients send requests to a server, it responds them with a GPDF rate. Results show that the relation between PDFs of components of patterns and the availability metrics can be modeled as declared previously by Kazman [8]. Therefore, this paper provides a utility function to represent the relation between the availability of patterns and their components. This utility function evaluates the quality of self-adaptive software architectures when their structures are imagined as a hierarchy of architectural patterns.

Our previous works [9, 10, 29] have proposed Fuzzy logic, AHP (Analytic Hierarchy Process) and Genetic algorithms to select the best composition of architectural patterns and a prototype have implemented. As they improve the planning activity of the SASSY framework, this paper enhances its data gathering and analysis activities. The remainder of this paper is organized as follows. Section 2 provides a more detailed explanation of architectural patterns, availability tactics, and their compositions. Section 3 describes related work. Section 4

presents our proposed approach. Section 5 provides our evaluations. Finally, Section 6 concludes the paper.

## II. ARCHITECTURAL PATTERNS AND TACTICS

### A. Architectural Patterns

Architectural patterns are practical solutions for a specific problem in a certain context [11]. The quality measurement is one of these problems which addressed in self-adaptive software architectures when they monitor the context of systems to analyze their quality in run time [12]. To this aim, pattern composition methods have been proposed recently to quantify the quality attributes. More specially, patterns influence certain quality attributes according to some criteria such as cohesion or coupling of interactions [13, 14, 15].

### B. Availability Tactics

Tactics are design decisions which control the quality of the architecture. They generally support the following three activities: 1) measuring certain quality attributes, 2) preventing systems from quality damages, and 3) recovering quality attributes [2]. Although they support several activities, this paper focuses on the quality measurement activity. Moreover, specific tactics are proposed for certain quality attributes. As mentioned in some literatures like [2] and [4], the availability attribute involves prevention, recovery and fault detection tactics. Fault detection tactics, such as Ping-Echo, Heartbeat, Exception and Voting [16, 2, 4], are just measurement tactics to quantify the availability of the software architecture. The functionality of both components and connectors are affected when tactics are applied to the software architecture [17]. RBML is a UML-based modeling language to describe these manipulations [18]. More specifically, RBML describes tactics as components and connectors with a specific functionality. Hence, availability tactics have been modeled in the RBML-PI add-in component by Kim [19].

### C. Composing Architectural Tactics and Patterns

The combination of tactics and patterns provides a basis for assessing the quality of self-adaptive software architectures. Various approaches have been offered recently to formalize this combination. For example, formal architectural map has been introduced in [20, 21] to transparently exhibit collaborations among tactics and patterns.

Some methods have been proposed to customize the architectural patterns with availability tactics due to their component-based structures. Moreover, the relationship among tactics, patterns, and quality attributes has been diagnosed in [17, 22]. They show the major operations to customize patterns according to the tactics. In this regard, six operations for implementing, replicating, adding (out of pattern), adding (in the pattern), modifying, and deleting are introduced for components or connectors. Moreover, they measure the difficulty of implementing tactics in architectural patterns.

## III. RELATED WORK

Although researchers are proposing many software architecture evaluation methods, they are not usable for a few software architecture domains such as real-time applications. Therefore, a self-adaptive evaluation method is required to measure the quality of applications in these domains. This section overviews the related works and compares their benefits and defects against our method.

The early evaluation methods such as scenario-based methods cannot support self-adaptive systems due to their offline process. Shanmugapriya and Suresh [23] have surveyed various early evaluation methods. Although, various methods evaluate different aspects of self-adaptive systems, none of them quantify the availability of these systems. Zhu et al. [24] have presented a mining approach which extracts the architectural tactics from the architectural patterns for each quality attribute. Although it measures the quality attributes of patterns, it does not present any prediction methods. Moreover, pattern comparison is a big challenge due to the dependency of tactics to patterns. Paakki et al [25] have proposed a pattern mining approach to detect the architecture patterns from the software architecture. Then, they collect some metrics, such as number of messages, to predict the quality of an architecture. Immonen [26] has provided a reliability and availability approach to predict these quality attributes. This approach maps the reliability and availability requirements into architectural models. Even though, it uses architectural patterns, and provides analytical models such as state-based models to predict the availability and reliability of architectures, it is a case base method. Moreover, it requires more time to predict the availability of software architectures.

## IV. PROPOSED AVAILABILITY EVALUATION APPROACH

In this section, a tactic-based method is introduced to evaluate the availability of self-adaptive software architectures. The proposed approach takes advantage of RBML modeling language to describe the composition of tactics and patterns. While RBML explains the major operations of tactics, numerous scenarios are applied to tactics to generate a huge dataset of availability samples. The generated results are enough to make a training model for predicting the availability of patterns.

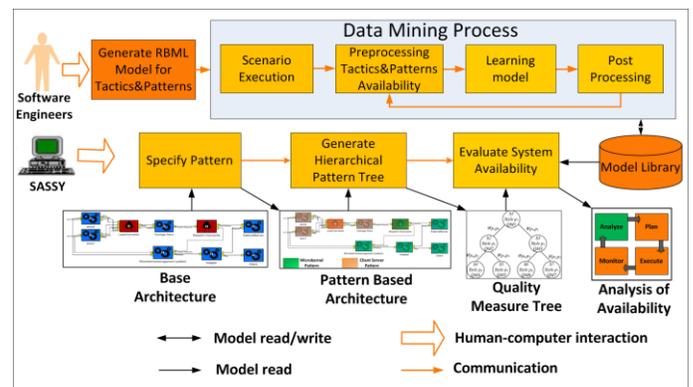


Figure 1. High level structure of tactic based evaluation method

Fig.1 depicts a high level structure of the tactic-based evaluation method. While software engineers are constructing the availability models with a repetitive mining process, the SASSY starts the evaluation process to measure the availability of a software architecture by using training models. SASSY follows the MAPE (Monitor, Analyze, Plan and Execute) automation model to re-architect the software architecture based on quality attributes. It can start the proposed method to measure the availability of a self-adaptive architecture before re-architecting. In the following, the proposed method is explained in more details.

### A. Modeling the Composition of Tactics and Patterns

The corresponding relationships among the components of patterns and the tactics are recognized by software engineers. The patterns are customized with appropriate operations before their combination with architectural tactics. Then, the customized patterns are specifically described with the RBML modeling language. In this paper, the RBML models for composing the patterns of Pipes-and-Filters and Microkernel with the tactics of Ping-Echo and Heartbeat are provided. Moreover, these tactics and patterns are simulated according to the proposed approach in literatures [2, 4, 16, 19, 22].

Pipes-and-Filters is a distributed pattern which basically has at least three components involving two filters and one pipe where filters process the flow of data and the pipe links filters together. Since all Pipes-and-Filters patterns can be produced from a basic one, the Ping-Echo and Heartbeat tactics have been composed with basic Pipes-and-Filters pattern.

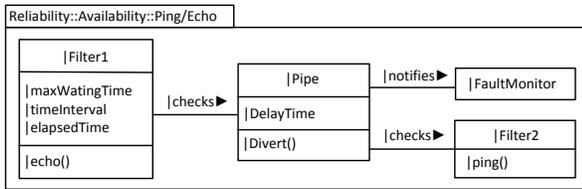


Figure 2. RBML model of composing Pipes-and-Filters and Ping-Echo

The RBML model of composing Pipes-and-Filters and Ping-Echo is represented in Fig.2. Filter1 sends packets in timeinterval periods and waits to receive the corresponding response from the Pipe component. Pipe buffers the packets and diverts them to Filter2. Finally, Pipe routes answers from Filter2 to Filter1. The packet will be dropped whenever this process takes more than the defined threshold.

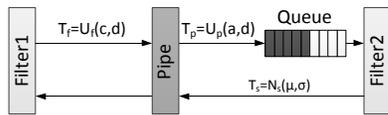


Figure 3. Simulation of composing Pipes-and-Filters and Ping-Echo

To simulate this composition, it is supposed that Filter1 generates asynchronous packets with a uniform distribution rate while the Pipe component makes some delays and forwards packets to Filter2. Since systems imitate the queuing theory, packets are buffered in a finite queue as Fig.3 displays. In other words, Filter2 returns packets in periods with a GPDF

rate due to the normal behavior of real systems. The Pseudo code of this simulation is given below. This algorithm generates numerous automated scenarios to collect an appropriate dataset for the data mining activity. In order to normalize the simulation results and cover all possible scenarios, scenarios have been limited to distinct ranges.

```

Compose Pipe-and-Filter and Ping-Echo (CPPFE)
1: Set Iteration number and appropriate ranges for Threshold, Queue size,  $U_f$ ,  $U_p$ ,  $N_s$ 's parameters.
2: For  $i=1$  to Iteration
3: Initialization: generate random parameters for  $U_f$ ,  $U_p$ ,  $N_s$  and random number for Queue size and Threshold with uniform distribution.
4:  $T_f \leftarrow U_f$ ,  $T_p \leftarrow U_p$ 
5: while {stable dropped and received packet curves} do
6:  $T_s \leftarrow N_s$ 
   execute one of the following statements with minimum time
7: Drop arrived packets to the full Queue.
8: Drop timed out packets from the Queue.
9: Filter1 sends a packet with  $T_f$  time interval.
10: Pipe inserts a packet to the Queue with  $T_p$  time interval.
11: Filter2 responds to arrived packets with  $T_s$  time period.
12: For each packet If  $T_f + T_p + T_s < \text{Threshold}$ 
13: Pipe increases received packet numbers.
14: else
15: Pipe increases dropped packet numbers.
16: End Program

```

Let packets be produced with  $T_f$  and  $T_p$  constant delay times in each iteration. Filter2 services queued packets by different  $T_s$  while the buffer is receiving packets in a  $T_f + T_p$  time period. In fact, Filter2 frequently services packets by a GPDF with constant mean and variance till the simulation result is stable.

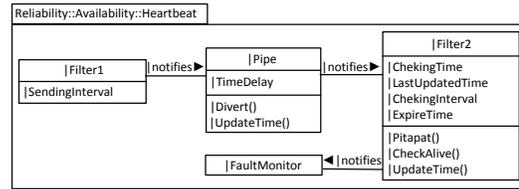


Figure 4. RBML model of composing Pipes-and-Filters and Heartbeat

Fig.4 depicts the RBML model of composing Pipes-and-Filters and Heartbeat. Filter1 sends packets toward the Pipe component periodically in durations of SendingInterval. Before the Pipe routes packets to Filter2, it updates the heartbeat time by the operation UpdateTime. Based on the Heartbeat definition, Filter2 compares the received time of packet with the previous one to check whether it is alive or not.

This composition has been simulated with the uniform and Gaussian probability distributions. Although it provides the same structure of Pipes-and-Filters and Ping-Echo compositions, it refuses to use the queuing theory due to the Heartbeat behavior. In fact, as Filter2 takes advantage of a single entry buffer to service packets with a GPDF rate, when it receives two packets simultaneously, it just services one packet and drops other.

As the below pseudo code demonstrates, packets are received when the absolute difference between the total delay and the previous receive time is less than the defined threshold. To model the composition of the microkernel pattern and the Ping-Echo tactics, it has been supposed that both client and adapter components are integrated in the adapter component as Fig.5 depicts.

### Compose Pipe-and-Filter and Heartbeat

- 1: Set Iteration number appropriate ranges for Threshold,  $U_f$ ,  $U_p$ ,  $N_s$ 's parameters.
- 2: For  $i=1$  to Iteration
- 3: Initialization: generate random parameters for  $U_f$ ,  $U_p$ ,  $N_s$  and random number for Threshold with uniform distribution.
- 4:  $T_f \leftarrow U_f$ ,  $T_p \leftarrow U_p$
- 5: while {stable dropped and received packet curves} do
- 6:  $T_s \leftarrow N_s$   
Execute one of the following statements with minimum time
- 7: Filter1 sends a packet with  $T_f$  time interval.
- 8: Pipe forwards a packet to Filter2 with  $T_p$  time interval.
- 9: For each packet If  $|T_f + T_p + T_s - \text{previous received time}| < \text{Threshold}$
- 10: Filter2 increases received packet numbers.
- 11: else
- 12: Filter2 increases dropped packet numbers.
- 13: End Program

Adapter sends packets to External Server and Microkernel components directly while Internal Server receives packets indirectly. Microkernel spends some time to divert packets to Internal Server from the Microkernel.

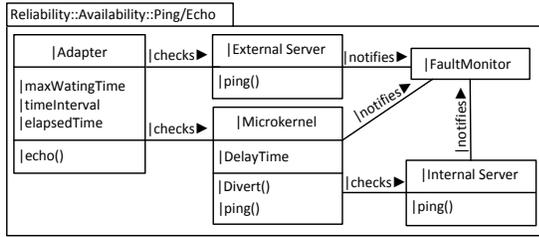


Figure 5. RBML model of composing microkernel and Ping-Echo

In addition, as depicted in Fig.6, Microkernel and Ping-Echo composition is simulated with three queues. This model utilizes the queuing theory while queues work independently. In other words, while the Adapter is sending packets, router decides to dispatch packets among queues with specific probabilities. Then, Microkernel, Internal, and External Server will respond to packets by a GPDF rate. Although Microkernel answers packets rapidly, it takes a little time for Internal and External Servers to respond packets due to their physical distance in real networks.

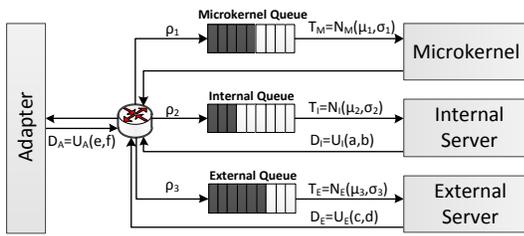


Figure 6. Simulation of composing microkernel and Ping-Echo

To predict the availability of the Microkernel pattern, the received and dropped packets are computed for Microkernel, Internal, and External Services separately as depicted in Fig.6. Finally, they are integrated to measure the availability of the Microkernel pattern.

Moreover, the composition of Heartbeat and Microkernel has been simulated with two independent scenarios. As represented in Fig.7, Adapter sends packets with the uniform distribution where Microkernel either responds to packets with a GPDF rate as previous simulations, or forwards them to the Internal Server.

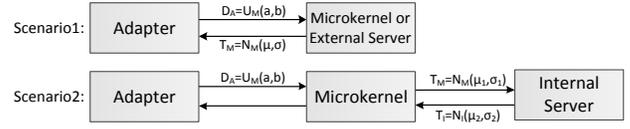


Figure 7. Simulation of composing microkernel and Heartbeat

### B. Data Mining Process

Data mining process is a repetitive activity which gathers datasets, prepares them, generates training models, and reanalyzes results. Accuracy of learning relies on several conditions such as the way datasets have been prepared, and the learning algorithms applied to training datasets. Therefore, this process improves training models based on the previous learning experiences. In the following, we describe how the data mining process has made highly accurate models for the aforementioned compositions of patterns and tactics. The following steps are followed in this process:

- **Scenario Execution.** This activity applies different simulation scenarios to generate training data. The input and output simulation parameters, including Threshold, Queue Size,  $U_A$ ,  $U_B$ ,  $U_E$ ,  $N_M$ ,  $N_E$ ,  $N_I$ , Received Time, and Drop Time make up the dataset features. We have divided input parameters into inner (which are set in the nested loop) and outer (other inputs) parameters in the aforementioned algorithms. Number of dropped and received packets will be stabilized when numerous scenarios with fixed inner parameters are applied to simulation models. As Table 1 shows, the maximum fluctuation of received and dropped packets is less than  $10^{-6}$  when they are stable.
- **Preprocessing the Availability of Tactics & Patterns.** To predict future events, the data mining process analyzes datasets to learn models. Besides, anomalies, null values, correlations, and outliers are common events in datasets which reduce the training accuracy. We have cleaned the simulation datasets by some preprocessing methods, like duplicate removal, anomaly reduction and type conversion methods. Moreover, we have labeled ReceivedPacket feature to make a classification model in the next step. Thus, we have converted this feature to polynomial values.
- **Learning Model.** This activity learns a training model when it provides a learning algorithm to analyze data relationships. Since the data mining process has been implemented in the Rapidminer (<http://rapidminer.com>) application, we have chosen the classification algorithm of this application to learn the simulation models. Although different classification models have been examined in next iterations, we explored that Neural Network algorithms have highest accuracy in comparison with other algorithms as table 1 shows. Moreover, recall, precision, and f-measure are other criteria that we use in our evaluations of models.
- **Post Processing.** The results of the evaluation show that both Ping-Echo on Microkernel and Heartbeat on Microkernel (Internal Component) have the lowest precisions against other simulations. By analyzing results with visualization methods, some classes consisting of a few records were

found. Although these classes could be removed by sampling, we would like to propose an appropriate algorithm to handle this challenge without dropping scarce scenarios in the future work.

TABLE 1. EVALUATING SIMULATIONS

	Received Fluctuation	Dropped Fluctuation	Algorithm	Recall	Precision	F-measure
	$\times 10^{-7}$					
Ping-Echo on Pipe-Filter	1.72	4.06	Neural Network	0.85	0.8	0.82
Ping-Echo on Microkernel	16.26	18.20	Bagging Neural Network	0.72	0.73	0.72
Heartbeat on Pipe-Filter	5.00	3.85	Auto MLP	0.75	0.74	0.74
Heartbeat on Microkernel (External Component)	2.38	2.63	Neural Network	0.91	0.92	0.91
Heartbeat on Microkernel (Internal Component)	3.53	5.14	Neural Network	0.81	0.72	0.76

## V. EVALUATION OF THE SYSTEM AVAILABILITY

As supposed that architects take advantage of pattern-based designing approaches, each subsystem will be designed by a distinct pattern where its components completely develop functionalities of the corresponding subsystems. While architects are thinking about system of systems, architectures will be produced by a hierarchical structure of patterns.

As Fig.1 depicts, SASSY analyzes the base architecture to map its components into appropriate patterns. By the previous assumption, software architectures are designed by a hierarchical structure of patterns where the root pattern distributes subsystems among its components.

As Fig.8 depicts, patterns are decomposed into several patterns except those that occur in the leaves. In fact, the decomposition of leaf patterns generates design patterns whereas the design models are out of the scope of architectural models.

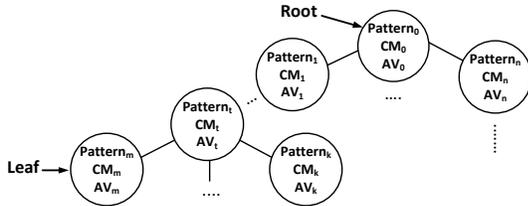


Figure 8. Hierarchical structure of patterns

$$AV_{\beta} = \begin{cases} T_{\beta} \times \prod_{\alpha \in \{\text{child of } \beta\}} AV_{\alpha} & , \beta \notin \text{Leaf} \\ T_{\beta} & , \beta \in \text{Leaf} \end{cases} \quad (1)$$

Where  $\{ \forall i \in \{\text{subsystems}\} AV_i \leq 1, T_i \leq 1 \}$

Let  $CM_i$  and  $AV_i$  represent the  $i^{\text{th}}$  component and its availability value respectively. Also,  $T_{\beta}$  determines the availability of pattern  $\beta$  which is earned by running architectural tactics. By this assumption, equation (1) formulates the availability of pattern  $\beta$ .

While availability is defined as the probability of access to services whenever authorized users request them,  $AV_i$  and  $T_i$  variables are stochastic variables. Moreover, when patterns

and their components have independent distributions, the probability of access to all components is equal to the production of probability of access to each component separately. The below algorithm represents evaluation steps:

### Availability Evaluation Algorithm

- 1: Explore PDF parameters of components
- 2: Post order search hierarchical structure of patterns
- 3: For each pattern do
- 4: Fetch corresponding model from Model Library
- 5: Set model parameters
- 6: Predict availability of pattern(T variable) from the model
- 7: Measure total availability of pattern(AV) from components and patterns availability.
- 8: End Program

While we have supposed that components send requests or respond them by specific PDFs, the self-adaptive systems analyze components to explore basic parameters of their PDFs. In fact, they recognize the average and variance of components where they imitate the GPDF. Besides, they explore the uniform distribution function value where components either send or respond to packets with this function.

To measure the total availability, the hierarchical structure of patterns is traced with a post order search. Therefore, the availability of subsystems is measured before their parent. While supposed that each subsystem is designed by an architectural pattern, its corresponding training model is fetched from the library model. To predict the availability of a pattern, the PDF of that pattern and its parameters are required. The PDF of aforementioned patterns is GPDF because the Ping-Echo and Heartbeat messages go through the independent components of patterns. Therefore, if  $N_i(\mu_i, \sigma_i)$  is the GPDF of  $i^{\text{th}}$  component then  $N(\mu, \sigma)$  is the GPDF of pattern with the following parameters [27]:

$$\mu = \frac{\sum_{i \in \{\text{Components of pattern}\}} \mu_i}{\sum_{i \in \{\text{Components of pattern}\}} 1}, \sigma^2 = \frac{\sum_{i \in \{\text{Components of pattern}\}} \sigma_i^2}{\sum_{i \in \{\text{Components of pattern}\}} 1}$$

Finally, self-adaptive systems make use of (1) to measure the total quality of patterns with regard to their components. This process continues to compute the availability of the root pattern which represents the quality of the system.

### A. Case Study

Rapidminer is a platform that provides an environment for data mining [28]. In this study, we reverse engineered this application with the Enterprise Architect to extract its class diagram. Then, we selected the main operator classes. As Fig.9 depicts, this subsystem is produced with the Microkernel and Pipes-Filters patterns. To explore the GPDF parameters of these components, we ran sample data mining projects on a five-core system with 2.66 GHz CPU and 4.00 GB of RAM and stored the execution time of components. The average results are summarized in Table2. Moreover, we enhanced the components with 3 threads to implement a queue with 3 entries. To explore the values of  $D_f$ ,  $D_E$ ,  $D_A$ ,  $T_P$  and  $T_F$  we have computed the delay between components. In addition, we have supposed that ping request must be receive lower than 50000 $\mu$ s time. The comparison between the results of the Ping-Echo tactic on the entire subsystem and our method illustrated that our method can predict the availability of this case study with a precision of more than 67%.

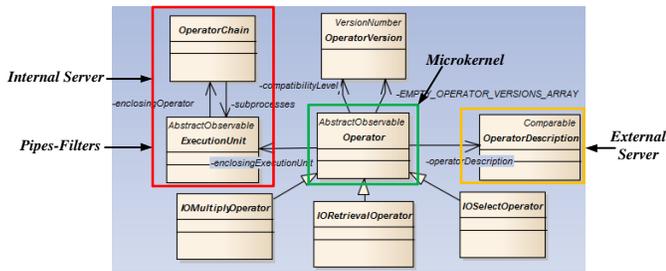


Figure 9. The main frame class diagram of Rapidminer

TABLE2. GPDF PARAMETERS OF MAIN COMPONENTS BASED ON MILLISECOND( $\mu$ S)

	Operator	OperatorChain	OperatorDescription	ExecutionUnit
$\mu$	20000	25320	32010	45040
$\sigma$	123	102	89	141
<b>Uniform Distribution Values</b>				
	$D_I$	$D_E$	$D_A$	$T_P, T_F$
	50	48	65	59

## VI. CONCLUSIONS

Self-adaptive architectures demand on evaluating the quality of the system in a short period of time. According to the SASSY framework, software architecture can be completely designed by a collection of patterns. This paper introduced an automated quality evaluation method that composes availability tactics and patterns with the RBML modeling language. Simulating RBML models with PDFs will result in useful datasets that can be applied in a data mining process to create a library of training models. The results illustrate that composing Ping-Echo and Heartbeat tactics with Microkernel and Pipes-and-Filters patterns make highly accurate models. Moreover, a mathematical formula to estimate the availability of an architecture by using training models was suggested. Applying the proposed method on a subsystem of the Rapidminer application shows that it can predict its availability with a permissible precision. In the future work, we want to expand the aforementioned method for other availability tactic and pattern compositions, quality attributes, and PDFs. Moreover, our future work purpose is development of a self-adaptive tool that endures the proposed method.

## REFERENCES

- [1] D. G. Firesmith, P. Capell, and et al, *The method framework for engineering system architectures*, CRC Press, 2008, pp. 39-70.
- [2] L. Bass, *Software Architecture in Practice*, 3rd ed., Addison-Wesley Professional, 2003, pp. 87-250.
- [3] B. Roy and T. Graham, *Methods for evaluating software architecture: A survey*. Tech. Rep., Queen's University at Kingston, 2008.
- [4] F. Bachmann, L. Bass, and M. Klein, *Deriving architectural tactics: A step toward methodical architectural design*. Tech. Rep., CMU/SEI-2003-TR-004, 2003.
- [5] E. D. Nitto, C. Ghezzi, and et al, "A Journey to highly dynamic, self-adaptive service-based applications," *Automated Software Engineering*, Vol. 15, pp. 313-341, December 2008.
- [6] D. Menasce, H. Gomma, and et al, "SASSY: A Framework for Self-Architecting Service-Oriented Systems," *Software, IEEE*, Vol. 28, pp. 78-85, December 2011.
- [7] S. Kim, D. Kim, and et al, "Quality-driven architecture development using architectural tactics," *Journal of Systems and Software*, Vol. 82, pp. 1211-1231, August 2009.
- [8] R. Kazman, S. J. Carrière, and S. G. Woods, "Toward a discipline of scenario-based architectural engineering," *Annals of Software Engineering*, Vol. 9, pp. 5-33, January 2000.
- [9] S. Moaven, J. Habibi, and et al, *A Fuzzy Model for Solving Architecture Styles Selection Multi-Criteria Problem*. In Proc. Computer Modeling and Simulation. Liverpool, United Kingdom, pp. 388-393, 2008.
- [10] S. Moaven, A. Kamandi, and et al, *Toward a Framework for Evaluating Heterogeneous Architecture Styles*. In Proc. Intelligent Information and Database Systems. Dong Hoi, Vietnam, pp. 155-160, 2009.
- [11] P. Coad, "Object-oriented patterns," *Communications of the ACM*, Vol. 35, pp. 152-159, september 1992.
- [12] M. D. P. Romay, L. Fernández-Sanz, and D. Rodríguez, *A Systematic Review of Self-adaptation in Service-oriented Architectures*. In Proc. The Sixth International Conference on Software Engineering Advances. Barcelona, Spain, pp. 331-337, 2011.
- [13] R. T. Fielding, *Architectural Styles and the Design of Network-Based Software Architectures*, PhD dissertation, Dept. of Computer Science, Univ. of California, Irvine, Calif., 2000.
- [14] R. Cloutier, *Applicability of Patterns to Architecting Complex Systems*, PhD dissertation, Stevens Institute of Technology, 2006.
- [15] N. Harrison, and P. Avgeriou, *Leveraging architecture patterns to satisfy quality attributes*. In Proc. The First European Conference on Software Architecture. Berlin, Germany, pp. 263-270, 2007.
- [16] P. Trivedi, A. k. Dubey, and S. Pachori, *Reliability tactics*. In Proc. The Electronics Computer Technology. Kanyakumari, pp. 167-169, 2007.
- [17] S. Malek, and et al, *Self-Architecting Software Systems (SASSY) from Qos-Annotated Activity Models*. In Proc. Principles of Engineering Service Oriented Systems. Vancouver, Canada, pp. 62-69, 2009.
- [18] R. B. France, D. Kim, and et al, "A UML-based pattern specification technique," *IEEE Transaction on Software Engineering*, Vol. 30, pp. 193-206, March 2004.
- [19] S. Kim, D. Kim, and S. Park, *Tool Support for Quality-Driven Development of Software Architecture*. In Proc. The IEEE/ACM international conference on Automated software engineering. Antwerp, Belgium, pp. 127-130, 2010.
- [20] H. Bagheri, Y. Song, and K. Sullivan, *Architectural style as an independent variable*. In Proc. The 25<sup>th</sup> IEEE/ACM International Conference on Automated Software Engineering. Belgium, 2010.
- [21] H. Bagheri, and J. S. Kevin, *A Formal Approach for Incorporating Architectural Tactics into the Software Architecture*. In Proc. The 23th International Conference on Software Engineering and Knowledge Engineering. Miami, USA, pp. 770-775, 2011.
- [22] N. B. Harrison, and P. Avgeriou, "How do architecture patterns and tactics interact? A model and annotation," *Journal of Systems and Software*, Vol. 83, pp. 1735-1758, October 2010.
- [23] P. Shanmugapriya, and R. M. Suresh, "Software Architecture Evaluation Methods – A survey," *International Journal of Computer Applications*, Vol. 50, pp. 19-26, July 2012.
- [24] L. Zhu, M. A. Babar, and R. Jeffery, *Mining Patterns to Support Software Architecture Evaluation*. In Proc. The Fourth Working IEEE/IFIP Conference on Software Architecture. 2004.
- [25] J. Paakki, A. Karhinen, J. Gustafsson, L. Nenonen, and A. I. Verkamo. *Software metrics by architectural pattern mining*. In Proc. The International Conference on Software: Theory and Practice. Beijing, China, pp. 325-332, 2000.
- [26] A. Immonen, *A method for predicting reliability and availability at the architectural level*. In Research Issues in Software Product-Lines - Engineering and Management. T. Kakola and J. C. Duenas, Eds. Berlin Heidelberg, pp. 373-422, 2006.
- [27] V. Capasso, D. Bakstein, *An Introduction to Continuous Time Stochastic Processes: theory, models, and applications to finance, biology, and medicine*, 2nd ed, Birkhauser, 2012, pp. 28-30.
- [28] M. Hofmann, R. Klinkenberg, *RapidMiner: Data Mining Use Case and Business Analytics Applications*, 1rd ed., CRC Press, 2013, pp. 3-19.
- [29] S., Moaven, J., Habibi, H., Ahmadi, and A., Kamandi, *A decision support system for software architecture-style selection*. In Proc. SEKE. Boston, USA, pp. 147-151. 2009

# Automated Software Architectural Synthesis using Patterns: A Cooperative Coevolution Approach

Yongrui Xu

State Key Lab of Software Engineering  
School of Computer, Wuhan University  
Wuhan, China  
xuyongrui@whu.edu.cn

Peng Liang\*

State Key Lab of Software Engineering  
School of Computer, Wuhan University  
Wuhan, China  
liangp@whu.edu.cn

**Abstract**—In software architecting process, architects use architectural patterns as reusable architectural knowledge for architectural synthesis. However, it has been observed that the resulting architecture does not always conform to the initial architectural patterns employed. Architectural synthesis using architectural patterns is also recognized as a challenging task, especially for novice architects due to lack of experience. In this paper, we propose a cooperative coevolution approach to automate architectural synthesis using architectural patterns. We first analyze several common architectural patterns and the constraints when using them. We then extend existing architectural synthesis with patterns based on the results of this analysis. We also describe the definition process for pattern metrics, which are used for automated architectural synthesis, from pattern constraints. Finally, we map the extended architectural synthesis to a cooperative coevolution model, which can optimize the resulting architectural solutions and avoid the violations to the pattern constraints automatically. Myx architectural pattern is used as an example to illustrate our approach.

**Keywords**—*automated architectural synthesis; architectural patterns; cooperative coevolution*

## I. INTRODUCTION

Large software systems are composed of lots of components, and the interactions between them are very complex. To reduce the complexity when designing software architectures, architects rely on a set of idiomatic architectural patterns, which are packages of architectural design decisions and are identified and used repeatedly in practice [1], such as MVC, pipe and filter, blackboard, and layer patterns.

Using architectural patterns for architectural synthesis gets lots of benefits, and the software architecture of large systems is increasingly designed by composing architectural patterns [1][2]. Therefore, many existing work focuses on how to select appropriate patterns from a pattern repository in specific design context by considering quality requirements in architectural synthesis [3][4]. However, many researchers observed that the resulting architecture of a system does not always conform to the initial patterns employed which guide the design at the beginning [5]. It is mainly due to the reasons that (1) existing work focuses on pattern recommendation and selection, but pays less attention to the conceptual gap between the abstract elements and the implementation units in the employed patterns; (2) each pattern has a set of design

constraints when using it, and architects may use the pattern being unaware of the constraints or misinterpreting the constraints due to lack of experience (especially for novice architects). If the pattern constraints are not satisfied, architects may have to redesign the architecture in order to avoid negative impact to the quality of the system. In summary, most existing work focuses on “architectural patterns recommendation and selection” instead of “architectural patterns implementation” which is part of architectural synthesis [6], and they did not address how to arrange components and connectors elegantly in a pattern to avoid the violations to the pattern constraints.

On the other hand, architectural synthesis heavily depends on the experience of architects, especially when the design space is increased exponentially with increasing system scale. Many approaches have been proposed to support exploring and exploiting architecture design space automatically [7]. Most of them use the Search-Based Software Engineering (SBSE) or Search-Based Software Design (SBSD) techniques [8]. However, as mentioned in [8], although many aspects of Software Engineering problems lend themselves to a coevolution model of optimization, surprisingly, there is little work that has been done on using this coevolution model to address Software Engineering problems. Automated architectural synthesis is one of these problems.

To this end, we propose a cooperative coevolution approach that aims at synthesizing pattern-based architecture solutions automatically. This approach tries to avoid the violations to the pattern constraints while considering the responsibility assignment in the resulting architecture solutions. In our approach, we first extend the classical architectural synthesis activity which essentially links the problem space to the solution space of architecture design with two parts: manual steps by architects and automated steps by tools. We then investigate on the constraints of existing architectural patterns, and define pattern metrics based on these constraints to construct the fitness function for automated architectural synthesis by tools. As we mentioned before, when the candidate architecture solutions are synthesized, there are two main objectives (i.e., avoid violations to pattern constraints and assign responsibility to architectural elements). Each objective may correspond to one population (a set of solutions). When the two objectives are not strongly correlated, their two populations can be coevolved to work better together, which offers great potential for

---

\* Corresponding author

This work is partially sponsored by the NSFC under Grant No. 61170025.

architectural synthesis [8]. Hence the problem of automated architectural synthesis using patterns is translated into a cooperative coevolution optimization problem. We demonstrate how the proposed approach can help architects arrange components and connectors with minimum constraint violations to implement architectural patterns in specific design context. The contributions of this work are: (1) defining and using pattern metrics to measure the violations to pattern constraints in architecture design; and (2) translating the pattern-based architectural synthesis to a cooperative coevolution problem, which can be automated.

The rest of this paper is organized as follows. Section II introduces the automated architectural synthesis approach in detail. Section III uses Myx architectural pattern, which is used in ArchStudio [9], as an example, to explain the use of our approach. Related work is discussed in Section IV and we conclude and outline future directions in Section V.

## II. APPROACH

In this section, we first analyze several common architectural patterns and the constraints when using them. We then extend classical architectural synthesis activity presented in [10] to a pattern-based architectural synthesis. We also describe the definition process for pattern metrics in detail in order to improve the applicability of our approach when architects use their own patterns, which are not covered in this paper. With the definition process, architects can define the pattern metrics from pattern constraints. Finally, we map the extended architectural synthesis to a cooperative coevolution optimization model, which tries to avoid the violations to the pattern constraints while considering the responsibility assignment of architectural elements automatically. The proposed approach makes the resulting architecture conform to the initial patterns employed at the beginning, while allocating responsibility for architectural elements in architectural synthesis.

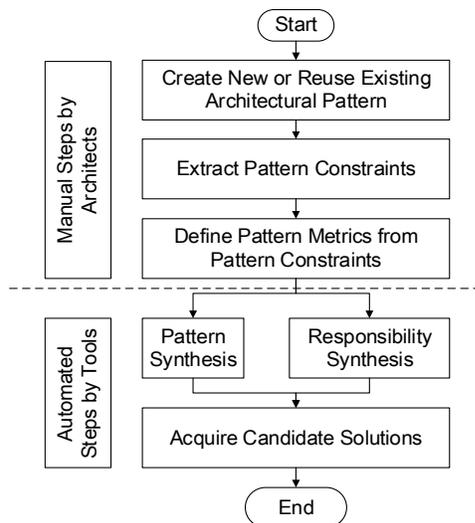


Figure 1. Extended architectural synthesis activity

### A. Extended Architectural Synthesis

General architecting process is composed of three activities: architectural analysis, synthesis, and evaluation [10], in which classical architectural synthesis (AS) activity

proposes a collection of candidate architectural solutions (e.g., architectural patterns) to address the architecturally significant requirements (ASRs) identified during architectural analysis. Architectural synthesis essentially links the problem to the solution space of architecture design. However, how to propose architecture solutions to a set of ASRs largely depends on the experience of architects in classical AS, and to make the matter worse, there is no available guidelines and steps to perform this activity for architects. In order to reduce the probability of making mistakes in AS (e.g., due to the lack of experience), we introduce the extended AS for architects.

As shown in Figure 1, the steps in the extended AS are divided into two parts: manual steps by architects and automated steps by tools. The output of architectural analysis (AA) is a set of function requirements and ASRs, and architects need to choose or create appropriate patterns to address them in architectural synthesis. Distinguished from other ways of using architectural patterns, our approach focuses on pattern constraints in pattern implementation. Architects can use pattern constraints which have been defined for common patterns or define the constraints for their own specific patterns. The next step of our approach is to define pattern metrics from pattern constraints by architects. In this paper, we provide the pattern metrics for an example architectural pattern - MVC, and we describe the definition process of pattern metrics from pattern constraints for architects who want to use other patterns. As shown in Figure 1, when the pattern metrics are defined, the automated part of the extended AS starts. The automated part is composed of two sub-processes: *responsibility synthesis* (RS) and *pattern synthesis* (PS), in which responsibility means functional requirements that should be implemented. In RS, the functional requirements are used as input, and this sub-process only considers the responsibility of the system by focusing on what the system should do. Unlike RS, PS sub-process is independent of business context and it only focuses on pattern implementation. It takes pattern metrics as input, and uses them to construct a fitness function which tries to minimize violations to the pattern constraints. PS and RS are automated and executed simultaneously, and they form the cooperative coevolution optimization model. The benefit of this partition between RS and PS is that PS is independent of business context, while RS focuses on the functional aspect of a system. Hence, in our approach, we follow the design concept of “divide-and-conquer” for architectural synthesis to implement the principle of “separation of concerns” [2].

### B. Constraints and Metrics of Architectural Patterns

An architectural pattern is composed of a triple  $\{context, problem, solution\}$ . In [1], Bass *et al.* further refine the solution of a pattern to five parts including *overview, elements, relations, constraints, and weaknesses*. The constraints of a pattern play an important role in limiting the possible pattern implementations. However, in practice, architects may choose to violate the constraints of the selected pattern in order to make a tradeoff among different factors, such as system quality attributes, implementation cost. This is the major reason why the resulting architecture of a system does not always conform to the initial patterns which guide the architecture design at the beginning [5].

The importance of design constraints in design has been recognized in [11] and design constraints are represented as a first-class entity in architecture design reasoning [12]. Hence we represent pattern constraints, a type of design constraint, as a first-class entity when using patterns in architectural synthesis, and we need to answer “*how to represent the pattern constraints when using architectural patterns?*” We choose several widely-used architectural patterns, and analyze their constraints based on the pattern descriptions in [1][2]. We summarize the constraints of several common architectural patterns in Table 1.

**Table 1.** Constraints of common architectural patterns

Pattern Name	Pattern Constraints
<b>Model-View-Controller (MVC)</b>	<ol style="list-style-type: none"> <li>1. There should be at least one instance of each pattern elements, i.e., model, view, and controller.</li> <li>2. The model element should not interact directly with the controller.</li> <li>3. There should be at least one view corresponding to an instance of model.</li> <li>4. Every view should be associated with at least one model.</li> <li>5. There is a one-to-one relationship between views and controllers.</li> </ol>
<b>Blackboard</b>	<ol style="list-style-type: none"> <li>1. There should be at least one instance of each pattern elements, i.e., blackboard, knowledge source, and controller.</li> <li>2. Control data and partial solutions are stored in blackboard, and knowledge sources access data through blackboard’s interfaces.</li> <li>3. Every knowledge source should not depend on other knowledge sources.</li> <li>4. Besides itself, every knowledge source should only access blackboard.</li> </ol>
<b>Pipe-and-Filter</b>	<ol style="list-style-type: none"> <li>1. Both pipe and filter can only connect the other type (i.e., filter and pipe) directly.</li> <li>2. In order to reduce the complexity, it restricts the association of components to an acyclic graph or in a linear sequence.</li> <li>3. The type of filter is either passive or active.</li> </ol>
<b>Reflection</b>	<ol style="list-style-type: none"> <li>1. There are at least two levels, including a meta level and a base level.</li> <li>2. Base level components may only communicate with each other via a metaobject at meta level.</li> <li>3. System aspects that are expected to stay stable should not be at meta-level.</li> <li>4. A metaobject does not allow the base level components to modify its internal state.</li> </ol>
<b>Layer</b>	<ol style="list-style-type: none"> <li>1. Every piece of a system is allocated to exactly one layer.</li> <li>2. There are at least two layers.</li> <li>3. The allowed-to-use relations should not be circular.</li> <li>4. The number of relations between components that travel through a subset of the layers should be insignificant compared to the number of adjacent dependencies between adjacent layers.</li> </ol>

In order to evaluate the quality of candidate architectural solutions generated in automated architectural synthesis, we need to define pattern metrics, which are used to measure the quality attributes of solutions, from pattern constraints. Due to space limitations, we only describe the pattern metrics from

the pattern constraints for one pattern: MVC, as an example, since this pattern is one of the most well-known patterns in architecture design. We will further introduce the definition process for pattern metrics in the next subsection.

In an interactive application, it is important to keep modifications to the user interface separate from the rest of the system. To address this design issue, MVC separates application’s functionality into three types of components: model, view, and controller, which are essential for an application of using MVC pattern. Hence, we define *LegalMVC* metric to judge whether the solution satisfies Constraint (1) of MVC pattern (as shown in Table 1). For Constraint (2), we define *ControllerUse(m)*, in which *m* is a given model, to count the number of relations that an element in model *m* depends on the element in the controller element. For Constraint (3), we define *CorrespondingViewUse(m)* to count the number of change notifications from *m* to all the views that correspond to it. Note that, for Constraint (2), if model elements depend on controller elements, it has an apparently negative impact on modifiability, portability, and reuse of architecture elements. Hence for some metrics like *ControllerUse(m)*, which have a great impact on quality attributes, we give them a high weight generally (i.e., the higher weight, the more the influence on the calculation of constraint violations). In Formula (1), we define model cost (MC) for a model element *i* to calculate pattern constraint violation cost for element *i*, where *i* belongs to model type in MVC.  $\alpha$  and  $\beta$  are the value of weight. Since *ControllerUse* has a higher weight than *CorrespondingViewUse*, we set  $\alpha \ll \beta$ . From Formula (1), the model cost for a given model element depends on the number of relations between this model element and its related view and controller elements.

$$MC(i) = \alpha \text{CorrespondngViewUse}(i) + \beta \text{ControllerUse}(i), \text{ where } \alpha \ll \beta \quad (1)$$

Similar to the pattern metrics definition of model element, for a given view *v*, there are also two metrics: for Constraint (4), we define *CorrespondingModelUse(v)*, in which *v* is a given view, to count the number of all the state-query relations from *v* to its corresponding models, while we define *ControllerUse(v)* to count the number of relations from *v* to its controllers. For Constraint (5), we define *CorrespondingControllerNumber(v)* for a given view *v* to check whether there is a one-to-one relationship between *v* and its controllers. For a view element *i*, we define view cost (VC) that is similar to model cost.

$$VC(i) = \alpha \text{CorrespondngModelUse}(i) + \beta \text{ControllerUse}(i) + \gamma \text{CorrespondngControllerNumber}(i), \text{ where } \alpha \ll \gamma, \beta \ll \gamma \quad (2)$$

In addition, for controller element *c*, we define *CorrespondingModelUse(c)* to count the state-change messages from *c* to its corresponding models. Similarly, two metrics *CorrespondingViewNumber(c)* and *ViewUse(c)* are defined for controller elements. Similar to MC and VC, we define controller cost (CC) for a given individual controller element *i* in MVC pattern:

$$CC(i) = \alpha \text{CorrespondngModelUse}(i) + \beta \text{ViewUse}(i) + \gamma \text{CorrespondngViewNumber}(i), \text{ where } \alpha \ll \gamma, \beta \ll \gamma \quad (3)$$

In order to evaluate the quality of different candidate solutions in automated pattern synthesis, we need to calculate the fitness score for a given MVC solution. If *LegalMVC* is *false* for one solution, it means that the solution is not a reasonable MVC solution, and we simply set the fitness score as  $\infty$ ; or if *LegalMVC* is *true*, the fitness score is calculated by summing the individual cost for every model, view, and controller elements (Here, we suppose that the solution is composed of  $r$  models,  $s$  views, and  $t$  controllers).

$$Fitness(s) = \begin{cases} \alpha \sum_{i=1}^r MC(i) + \beta \sum_{i=1}^s VC(i) + \gamma \sum_{i=1}^t CC(i) & \text{when } LegalMVC \text{ is True} \\ \infty & \text{when } LegalMVC \text{ is False} \end{cases} \quad (4)$$

As we can see from the MVC example, the evaluation for pattern constraint violations depends on the defined pattern metrics from pattern constraints. Since the quality of pattern metrics has a great impact on pattern synthesis, we describe the definition process for pattern metrics in detail in the next subsection so that architects can define the pattern metrics of specific architectural patterns of their own.

### C. Definition Process for Pattern Metrics

The definition process for pattern metrics is composed of three steps (*discover the roles of a pattern*, *discover the relations within a pattern*, and *discover the domain related metrics*), which are detailed below:

1) *Discover the roles of a pattern*. As a pattern provides a generic solution for a recurring problem: a solution that can be implemented in many ways without necessarily being ‘twice the same’ [13], and there is no configurable generic implementations for patterns that cover their whole design space. However, every pattern has its invariable roles, such as model, view, and controller in MVC pattern. Therefore, we can identify some metrics from pattern roles. This step may include the following sub-steps:

a) Define the metric of upper and lower limit about each role from pattern constraints. For example, in Layer pattern, we define *LegalLayers* to ensure that the number of layers is more than one.

b) Define the metric about the quantity relationship between different roles from pattern constraints. For example, in MVC pattern, we define the metrics *CorrespondingControllerNumber(v)* and *CorrespondingViewNumber(c)*.

c) Define the metric about type of roles from pattern constraints. For example, there is a passive filter or active filter in Pipe-and-Filter pattern, and different types of filter may influence the quality attributes of a system (e.g., performance). We define *PassiveFilterNumber* and *ActiveFilterNumber* for the two filter types in Pipe-and-Filter pattern.

d) Define the metric about responsibility of roles from pattern constraints. For example, according to the description of Constraint (2) in Blackboard pattern, the partial solutions acquired from each knowledge source and the control data should be stored in the role of blackboard. We define *ImproperDataNumber* to count the number of data which are improperly stored outside blackboard.

e) Define the metric about the mapping relations for components and pattern roles. In some patterns, one component may play multiple roles, or vice versa. For example, one component in Blackboard may play the role of blackboard and controller simultaneously, which leads to coupling between data and control logic. Hence we define *PureBlackboard* for every blackboard in Blackboard pattern.

2) *Discover the relations within a pattern*. Every pattern contains a set of interactions between the roles in the pattern. In this step, we identify the metrics from the interactions. It includes the following sub-steps:

a) Define the metric for direction of interaction. The interaction between different roles in a pattern is usually unidirectional, such as the lower layer should not access the higher layer in Layer pattern, and model elements cannot depend on controller elements in MVC pattern.

b) Define the metric for cycle interaction. This sub-step is similar to the previous sub-step, but it is more complex, since cycle interaction often includes more than two role elements.

c) Define the metrics about constraints of relations between different roles. For example, for Constraint (1) of Pipe-and-Filter (as shown in Table 1), the relations between different roles are often limited in pattern constraints, which should be defined in pattern metrics.

d) Define the metrics of relation types. There are many relation types between two elements in a pattern, such as inheritance, implementation, association, and so on. As different relation types may have different influence on quality attributes, we define specific metrics for the relation types.

e) Define the metrics for interaction mechanisms. A set of interaction mechanisms exist between elements in a pattern (e.g., events or messages), we should consider the interaction mechanisms in patterns, and define metrics for them.

3) *Discover the domain related metrics*. Every domain has its specific knowledge e.g., documented in literature and standards. Similarly, every software has its own application principles. In this step, architects define the metrics according to the domain and application principles.

The results of this definition process form a starting point for automated architectural synthesis. When this process finishes, the manual work by architects is completed as shown in Figure 1. The pattern metrics, which are acquired through this definition process, are used to construct the fitness function for automated pattern synthesis.

### D. Automated Architectural Synthesis

As discussed in Section II.A, with the extended pattern-based AS, architects can propose architectural solutions using patterns either by themselves (e.g., based on their experiences) or through pattern recommendation [14]. However, how to assign responsibilities to architecture design elements in RS sub-process and how to group elements to implement the architectural pattern in PS sub-process are challenging tasks in practice. Addressing both of them heavily depends on the experience of architects. Automation of AS activity is beneficial in that (1) it can evaluate the rationality of responsibility assignment to architectural elements; and (2)

avoid the violation to pattern constraints to a certain extent automatically. In addition, in the architecture design of large and complex systems, most of design problems may have a massive number of possible solutions, which is impossible to manually explore. To this end, we choose to employ a scalable meta-heuristic search technique to assist automated AS using patterns.

In our recent work [15], we formally defined the problem of automated pattern-based architectural synthesis. In our approach, the outcome of RS and PS sub-processes are regarded as two populations of solutions that evolve simultaneously, and the fitness of each individual in one population depends on the status of individuals in another population, i.e., the two populations have a cooperative coevolution relationship [8]. Therefore, it is feasible and reasonable to model the pattern-based AS as a cooperative coevolution optimization problem, which can be executed automatically to synthesize candidate architecture solutions.

The choice of the representations for collaboration problems and the definitions of the fitness functions for individuals in different populations are two ingredients for cooperative co-evolution. In RS (responsibility synthesis) sub-process, on one hand, the main decision is whether an architectural element should take certain responsibility. We use a binary string encoding scheme from the SBSE representation techniques [16] to represent a responsibility that is assigned to certain architectural elements. On the other hand, as there have been some mature metrics associated with the responsibility assignment problem that form good initial candidates for the RS fitness function (e.g., cohesion and coupling metrics), we directly use these metrics for RS in the cooperative coevolution. In PS sub-process, we consider what specific role an architectural element plays in that pattern, and we use one digit (0~9) to represent the role type. Then, we use the pattern metrics defined for each pattern constraint as the PS fitness function in the cooperative coevolution. Due to space limitation, the representations for RS and PS populations and the fitness functions for evaluating these two populations are detailed in [14].

### III. EXAMPLE

In this section, we use Myx architectural pattern, which is employed in ArchStudio [9], as an example, to explain the use of our proposed approach.

#### A. Architectural Pattern Constraints

We omit the details of Myx pattern, which can be found in [17], due to space limitations. We only describe the major constraints of this pattern: (1) Each component has two zones (i.e., top and bottom), and all interfaces belonging to a component should be assigned to either of these two zones. In any invocation or dependency relationships, if one interface is assigned to the top zone of one component, the other interface in the same relationship should be assigned to the bottom zone of another component, or the other way round (hence, in Myx pattern, the top and bottom zones of each component is the pattern roles); (2) Cycle invocations (i.e., one invocation from the top zone of a component to the bottom zone of same component) are not permitted (whether a cycle invocation

exists is determined by the grouping of architectural elements; (3) Only upward synchronous invocations, in which the invocation direction is from the top zone of one component to the bottom zone of another component, are permitted, while asynchronous invocations have no any limitations, i.e., either upward or downward asynchronous invocations are permitted (an invocation direction also depends on the grouping of architectural elements).

#### B. Metrics from Pattern Constraints

Existing cohesion and coupling metrics (e.g., [18][19]) provide initial candidates for fitness function in RS sub-process, and we focus on the metrics from pattern constraints for fitness function in PS sub-process in this subsection. To use these metrics, we should specify above or below relationships between two components. We propose a heuristic approach to decide which component between two components is an “above” component. A “counter” property is introduced for each component. When there is an invocation from the top zone of component  $A$  to the bottom zone of another component  $B$ , we increase the  $B$ 's counter with 1. Otherwise, if the invocation comes from the bottom zone of component  $A$  to the top zone of component  $B$ ,  $A$ 's counter is increased by 1. Finally, we compare the value of the counters in two components, and the component that has a bigger counter value is regarded as the “above” component. For example, the counter values of component  $A$  and  $B$  are 1 and 3 respectively as shown in Figure 2, and then component  $B$  is “above” component  $A$ .

According to the definition process described in Section II.C, we define a set of metrics specified below based on the constraints of Myx pattern presented in Section III.A:

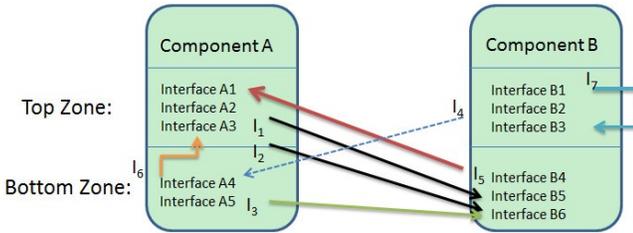
(1) *IntraUse(i)*: Since cycle invocations are not permitted, a component may never be above or below itself in invocation relationships. This metric measures the number of invocation relationships whose two connected interfaces are in the same component  $i$ , that violate Constraint (2).

(2) *IllegalUse(i,j)*: This metric denotes the number of invocation relationships between components  $i$  and  $j$  that violate Constraint (1).

(3) *Intra&IllegalUse(i)*: This metric is similar to *IntraUse(i)*. The difference between the two metrics is that this metric counts the invocation relationships that the two connected interfaces are not only in the same component  $i$ , but also in the same zone. It measures the number of invocation relationships that violate both Constraint (1) and Constraint (2).

(4) *ComplexUse(i,j)*: According to Constraint (3), downward invocation is only permitted for asynchronous invocations. In component-based software design, asynchronous invocations may complicate the system design, which is difficult to understand. The metric *ComplexUse(i,j)* denotes the number of asynchronous calls between component  $i$  and  $j$ . It indirectly measures the complexity of the design.

(5) *CycleUse(i,j)*: When component  $i$  is above component  $j$  in invocation relationships, the invocations from  $i$ 's top zone to  $j$ 's bottom zone lead to cycle invocations between the two components. Thus, we introduce this metric to count the number of calls that violate Constraint (2).



**Figure 2.** Example of different types of invocations in Myx Pattern

In Figure 2, there are 7 invocations between different interfaces (from  $I_1$  to  $I_7$ ), and we distinguish different kinds of invocations with different colors and lines. Invocation  $I_1$  and  $I_2$  are from component  $A$  to its above component  $B$ . These invocations satisfy all of the pattern constraints to be considered when using Myx pattern, since the two interfaces are in different components and only one of them is in top (or bottom) zone. According to the metrics definition in Section III.B, Invocation  $I_3$  can be counted in metric  $IllegalUse(A,B)$ ;  $I_4$  can be counted in metric  $CycleUse(A,B)$ ;  $I_5$  is a typical asynchronous invocation that can be counted in metric  $ComplexUse(A,B)$ ;  $I_6$  and  $I_7$  can be counted in metric  $IntraUse(A)$  and  $Intra\&IllegalUse(B)$  respectively.

We further define component violation cost ( $CVC$ ) for each component in the architecture design. In our definition,  $CVC$  accumulates the numbers of metrics for all the violated constraints to one component, while it can set weight for different types of constraint violations. Hence, we calculate  $CVC$  for a given component  $i$  using Formula (5):

$$CVC(i) = \alpha CycleUse(i, j) + \beta Intra\&IllegalUse(i) + \gamma IllegalUse(i, j) + \delta ComplexUse(i, j) + \epsilon IntraUse(i), \text{ where } i \neq j \quad (5)$$

Then, we define an objective function that calculates the total violation cost ( $TVC$ ) of a given architecture design solution using Formula (6). We try to acquire a candidate architecture solution with minimized  $TVC$ , which means the pattern constraint violations in the architecture design solution are minimized as well.

$$TVC = \sum_{i=1}^n CVC(i) \quad (6)$$

### C. Evaluation of Synthesis Results

The  $TVC$  metric defined in previous subsection can be used as an evaluation criterion for the PS sub-process. In the RS sub-process, most existing work focuses on the evaluation for the distribution of the responsibilities among components in the architecture design, such as the cohesion and coupling metrics [19]. In our proposed cooperative coevolution approach, one population evaluated by the cohesion and coupling metrics is used for RS, and another population evaluated by the  $TVC$  metric is used for PS. In every generation, the best individuals (i.e., the solutions that have high cohesion and low coupling) in RS population are combined with the individuals in population used by PS, and these hybrid individuals are used as input for the next generation in PS. Similarly, the best individuals (i.e., the solutions that have less pattern constraint violations) in PS are combined with the individuals in RS population as input for the next generation in RS. In each round of generation by PS and RS, we use a single fitness function for both populations

to evaluate the overall quality of the resulting architecture solutions. Therefore, we define the quality of a given solution  $s$  in Formula (7):

$$Quality(s) = \alpha Cohesion\&Coupling(s) + \beta TVC(s) \quad (7)$$

We use this formula to evaluate the candidate architecture solutions generated by the proposed approach using cooperative coevolution, and select and recommend the solution with highest (best) quality. It provides optimized architecture design solutions for architects.

## IV. RELATED WORK

We summarize and discuss relevant work on automated architectural synthesis and pattern constraints in this section.

Cui *et al.* [20] presented an automated decision-centric architectural synthesis approach, which transits from requirements to architectures through a solution exploiting and synthesizing process. In their approach, solution exploiting is accomplished by architects. For each elicited design issue, architects proposed solutions mainly based on their expertise and experience. Solution synthesizing in their approach is automated, which combines and evaluates all the feasible solutions from the solution exploiting results. Therefore, the quality of resulting solutions still heavily depends on the experience of architects.

Räihä [21] proposed to synthesize architecture using Genetic Algorithms (GA). In her approach, architectural styles and design patterns are used to transform the initial high-level architecture model to a detailed design. The architectural synthesis is based on an analysis model which contains information on functional requirements only. The differences between her approach and our proposed approach are that (1) design patterns and architectural styles are used as mutator for GA in Räihä's approach, and these patterns are inserted or deleted randomly in GA mutation. In our approach, we focus on the constraints of patterns, and which patterns are used is determined; (2) the criteria for evaluating candidate architecture solutions are different. Our approach considers the design quality (e.g., cohesion and coupling metrics) which is similar to Räihä's approach, while we also take pattern constraint violations into account.

Belle *et al.* [5] revisited the layer pattern to extract a minimum set of fundamental principles for using layer pattern, which are used to specify a series of constraints that a layered architecture should conform. They further made use of these constraints to guide the recovery of the layered architecture in a system, and model the architectural recovery as an optimization problem using automated heuristic search algorithm. However, their approach focuses on architecture recovery instead of architecture design, and their approach didn't consider the responsibility assignment of architectural elements in recovering layer pattern.

Bagheri and Sullivan [22] showed that it is possible to separate and combine formal representations of application properties (e.g., domain knowledge) and architectural styles. The key idea of their approach is to map the application which is independent of architecture styles to *models* (i.e., Platform Independent Model, PIM) in model-based development, and

map the architectural styles to *platforms* (i.e., Platform Definition Model, PDM). Similar to our proposed approach, their approach separates the application synthesis and architectural style synthesis during architectural synthesis, which also followed the design concept of “divide-and-conquer”. However, their approach is different from our approach in that (1) they used ADLs to formally define the specifications about application models and architectural styles, then a mapping engine is used to translate these specifications to architectural models in given architectural styles during architectural synthesis, while our approach uses a search-based technique which is more flexible to explore the whole design space for candidate architectural solutions; (2) the treatment of pattern constraints is different. They used a constraint solver to support incremental analysis and construction of models (solutions), but we use pattern constraints to define pattern metrics which are used to evaluate different solutions.

Maoz *et al.* [23] used component and connector views (C&C views) to investigate the architectural synthesis problem, and further extended this basic problem with support for architectural styles-based architectural synthesis. Similar to [22], they also used ADLs to formally define the C&C models and the architectural styles. Architectural synthesis with formal specifications using ADLs may end up with many satisfied solutions, and these satisfied solutions may have different qualities (e.g., both solutions *A* and *B* satisfy the performance requirements of an application, but the performance of solution *A* is better than solution *B*). It is difficult to recommend better solutions in candidate solutions with formal ADL techniques, which is the issue that our approach tried to address using cooperative coevolution.

## V. CONCLUSIONS AND FUTURE WORK

Architectural synthesis essentially links the problem to the solution space, and it plays a key role in architecting process from requirements to initial architecture design. However, due to its essential complexity, this architecting activity heavily depends on the experience of architects. In this paper, we extend the existing AS to a pattern-based AS, and propose a cooperative coevolution approach that synthesizes architecture automatically using architectural patterns. We first analyze several common architectural patterns, identify the pattern constraints, and represent them as a first-class entity for pattern implementation. We then present a process to define pattern metrics from pattern constraints, and acquire the pattern metrics to construct the fitness function for automated synthesis. The automated synthesis process is composed of two sub-processes: pattern synthesis (PS) and responsibility synthesis (RS). We further model these two sub-processes as a cooperative coevolution problem, which can be executed automatically to synthesize candidate architecture solutions. We use Myx architectural pattern as a concrete example to explain the use of the proposed approach.

We outline the future work in two points: (1) to conduct controlled experiments that compare the architecture design quality between the generated pattern-based AS solutions using our approach and the solutions by architects based on the same design problems in industry projects; (2) to develop a

tool that support the pattern-based automated AS using cooperative coevolution.

## REFERENCES

- [1] L. Bass, P. Clements, R. Kazman, *Software Architecture in Practice*, 3rd ed. Addison-Wesley Professional, 2012.
- [2] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal, *Pattern-Oriented Software Architecture: A System of Patterns*, 1st ed. Wiley, 1996.
- [3] K. D. Babu, P. Govindarajulu, A. R. Reddy, “ANP-GP approach for selection of software architecture styles,” *Int. J. Softw. Eng.*, 1(5):91-104, 2011.
- [4] M. Galster, A. Eberlein, M. Moussavi, “Systematic selection of software architecture styles,” *IET Softw.*, 4(5):349-360, 2010.
- [5] A. Belle, G. El Boussaidi, C. Desrosiers, H. Mili, “The layered architecture revisited: Is it an optimization problem?,” in *SEKE*, 2013, pp. 344-349.
- [6] A. Tang, P. Avgeriou, A. Jansen, R. Capilla, M. A. Babar, “A comparative study of architecture knowledge management tools,” *J. Syst. Softw.*, 83(3):352-370, 2010.
- [7] A. Aleti, B. Buhnova, L. Grunske, A. Koziolok, I. Meedeniya, “Software architecture optimization methods: A systematic literature review,” *IEEE Trans. Softw. Eng.*, 39(5):658-683, 2013.
- [8] M. Harman, S. A. Mansouri, Y. Zhang, “Search-based software engineering: Trends, techniques and applications,” *ACM Comput. Surv.*, 45(1):1-61, 2012.
- [9] J. Garcia, I. Krka, C. Mattmann, N. Medvidovic, “Obtaining ground-truth software architectures,” in *ICSE*, 2013, pp. 901-910.
- [10] C. Hofmeister, P. Kruchten, R. L. Nord, H. Obbink, A. Ran, P. America, “A general model of software architecture design derived from five industrial approaches,” *J. Syst. Softw.*, 80(1):106-126, 2007.
- [11] F. P. Brooks, *The Design of Design: Essays from a Computer Scientist*, 1st ed. Pearson Education, 2010.
- [12] A. Tang, H. van Vliet, “Modeling constraints improves software architecture design reasoning,” in *WICSA*, 2009, pp. 253-256.
- [13] F. Buschmann, K. Henney, D. C. Schmidt, *Pattern-Oriented Software Architecture: On Patterns and Pattern Languages*, 1st ed. Wiley, 2007.
- [14] M. Kassab, G. El-Boussaidi, H. Mili, “A quantitative evaluation of the impact of architectural patterns on quality requirements,” in *SERA*, 2011, pp. 173-184.
- [15] Y. Xu, P. Liang, “Co-evolving pattern synthesis and class responsibility assignment in architectural synthesis,” in *ECSA*, 2014. (under review).
- [16] M. Harman, P. Meminn, J. T. De Souza, S. Yoo, “Search Based Software Engineering: Techniques, Taxonomy, Tutorial,” in *Empirical Software Engineering and Verification*, 2012, pp. 1-59.
- [17] “The Myx Architectural Style.” Available at: <http://isr.uci.edu/projects/archstudio/myx.html>, accessed on 2013-11-22.
- [18] M. Bowman, L. C. Briand, Y. Labiche, “Solving the class responsibility assignment problem in object-oriented analysis with multi-objective genetic algorithms,” *IEEE Trans. Softw. Eng.*, 36(6):817-837, 2010.
- [19] C. L. Simons, I. C. Parmee, R. Gwynllwy, “Interactive, evolutionary search in upstream object-oriented class design,” *IEEE Trans. Softw. Eng.*, 36(6):798-816, 2010.
- [20] X. Cui, Y. Sun, H. Mei, “Towards automated solution synthesis and rationale capture in decision-centric architecture design,” in *WICSA*, 2008, pp. 221-230.
- [21] O. Räihä, “Genetic Algorithms in Software Architecture Synthesis,” Ph.D Thesis, School of Information Sciences, Tampere University, 2011.
- [22] H. Bagheri and K. Sullivan, “Monarch: Model-based development of software architectures,” in *MODELS*, 2010, pp. 376-390.
- [23] S. Maoz, J. O. Ringert, and B. Rumpe, “Synthesis of component and connector models from crosscutting structural views,” in *ESEC/FSE*, 2013, pp. 444-454.

# Towards Reusing Architectural Knowledge as Design Guides

## Functional Requirements, Tool Analysis and Research Roadmap

Mohsen Anvaari

Department of Computer and Information Science  
Norwegian University of Science and Technology  
(NTNU)

Trondheim, Norway  
mohsena@idi.ntnu.no

Olaf Zimmermann

Institute for Software  
University of Applied Sciences of Eastern Switzerland  
(HSR FHO)  
Rapperswil, Switzerland  
ozimmerm@hsr.ch

**Abstract**— In recent years, architectural knowledge management has demonstrated its potential to improve software development and evolution practices; various tools and research prototypes now exist for documenting architectural knowledge. However, capturing such knowledge is not enough: according to practitioners’ feedback, a certain amount of knowledge post-processing is required to make the captured knowledge consumable and stimulate reuse. In our previous work, we created a method for enhancing knowledge about the past (decisions made) into architectural guidance for the future (decisions required). However, additional concepts are required to let our method benefits from recent advances in architectural knowledge management tool engineering. In this paper we establish requirements for post-processing architectural knowledge captured on projects and enhancing the knowledge into architectural guidance. The requirements are derived from literature and industrial experiences. Next, we analyze existing tools with respect to these requirements. Finally, we establish a vision for an integrated method and tooling for architectural guidance modeling and outline a roadmap for future research and tool development towards this vision.

**Keywords**— *Architectural knowledge; decision reuse; architectural synthesis; design guide; knowledge management tool*

### I. INTRODUCTION

Architectural decisions are considered a first class entity in software engineering now [7]; researchers define software architecture as a set of architectural design decisions [1]. Various tools and research prototypes exist (or are under development) for documenting the architectural knowledge. Although most practitioners are still reluctant to use formal templates and tools that academic researchers have developed [9], our observations show that many organizations have started to capture their architectural decisions [13]. This is often done in light and pragmatic ways, e.g., using simple wikis or chronological meeting minutes [13].

According to studies on inhibitors for knowledge reuse, documenting the knowledge is not enough; post-processing is required to stimulate the reuse and make the knowledge consumable [20]. Hence, we created a method for enhancing knowledge about the past (decisions made) into architectural

guidance for the future (decisions required) [19]. However, additional concepts are required to let our method benefits from recent advances in architectural knowledge management tool engineering. Therefore, the goals of this paper are:

1. To specify the requirements for tools that facilitate the post-processing of captured architectural knowledge from projects and enhancing such raw knowledge into design guides for future decision making activities.
2. To analyze existing tools and research prototypes with respect to the proposed requirements.
3. To establish a vision for an integrated method and tooling for architectural guidance modeling.

The rest of the paper is organized as follows: In Section 2, we present related work. Section 3 describes our research method. Section 4 specifies the functional requirements for tools that support enhancing architectural raw knowledge into reusable design guided. These requirements are derived from the authors’ industrial experience as well as a review of research prototypes and tools. Section 5 reports on the results of our analysis of existing tools and research prototypes with respect to the functional requirements from Section 4. Section 6 analyzes our results and establishes an architectural vision for a tool that supports architectural knowledge reuse. Section 7 summarizes the paper with conclusions.

### II. RELATED WORK

The concept of architectural knowledge – defined as the integrated representation of the software architecture, the architectural decisions, and the external context/environment [23] – has been investigated by researchers since they started to consider the architectural decisions as important entities of a software system just like the architecture itself. In the last decade, the research community has elaborated the concept, clarified its definitions, terminologies and boundaries, established the ways of presenting the knowledge, and developed the approaches to manage the knowledge [12].

Applying general knowledge management principles [11] to the software engineering domain, two activities become essential for architectural knowledge management: creating (or capturing or documenting) knowledge and consuming (or

reusing or applying) knowledge. However, as we have shown in our previous work, the main focus of the software architecture community so far has been on knowledge capturing, not on knowledge reuse [20]. One may argue that when the knowledge is captured and made available to the others it is reusable; therefore any approach and tool that supports knowledge capturing also supports knowledge sharing and reusing automatically. But, according to practitioners' feedback, capturing the knowledge is not enough: a certain amount of knowledge post-processing is required to make the captured knowledge consumable [20]. Examples of knowledge post-processing are anonymizing the knowledge (e.g., remove sensitive personal information such as names of actual people, or replace them with role definitions such as "application architect" or "integration architect"), connecting the related knowledge entities (e.g., a decision about a message exchange pattern with a decision about a messaging provider software product), and removing the project-specific knowledge (chosen alternative) to make the knowledge reusable for other projects.

While the method we have created in our previous work for reusing architectural knowledge and enhancing the captured architectural knowledge into design guides<sup>1</sup> has demonstrated to be useful in the industry [19], better tool support is required to make the application of the method more efficient. As the first step, this paper explores the available architectural knowledge management tools and research prototypes to analyze how much they provide the required functionalities for architectural knowledge reuse and guidance development.

This paper is not the first survey in the software architecture domain to analyze and evaluate the architectural knowledge management tools. At least four preceding research papers have been published [2][14][16][25]. Although some papers have considered knowledge sharing as a functional requirement in their evaluation framework, their focus is not on knowledge post-processing and enhancing the knowledge into design guidance, which is the focus of our work. Furthermore, the last survey has been published in 2010 while in the last three years more tools and research prototypes have been developed or are under development. This paper covers these new tools as well. The mentioned research papers are valuable for our work; for instance, we have reused some of their functional requirements to establish the functional requirements in the section 4. In the next section, the method of the research will be explained.

### III. RESEARCH METHOD

As Fig. 1 shows, we started the research by creating functional requirements. To do so we explored three sources: 1) industrial experiences that originate in the authors' contribution in industrial projects and also their observations from various industrial domains (software development projects for Smart Grid, financial applications, etc.) 2) the tools and research prototypes that are not accessible but are specified in the literature 3) the tools that are accessible publicly on the

internet. The main way to reach to the second source was exploring the literature that has reviewed and compared the architectural knowledge management tools. As we mentioned earlier, the last comparative study of the architectural knowledge management tools was conducted in 2010 [2]. It still is a valuable source to explore the tools that had been developed until then. We discovered the newer tools either through the literature that tool developers have published or by contacting the researchers that we were informed are developing a tool.

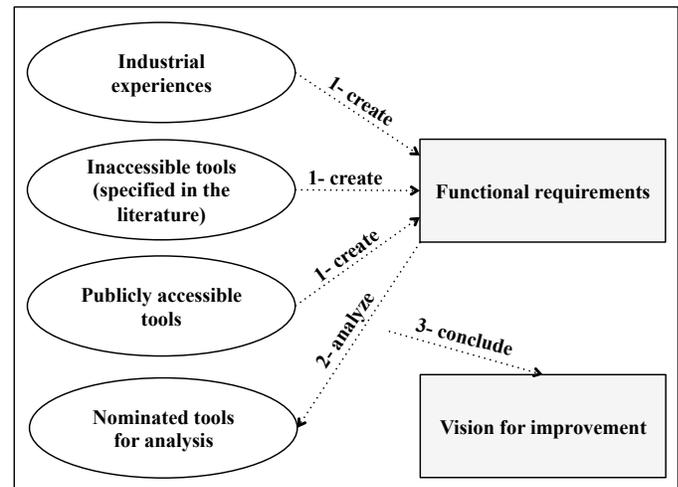


Figure. 1 Research activities and contributions

The second step of the research was to analyze the tools and research prototypes with respect to the functional requirements we proposed in the first step. The tools that are publicly accessible (the third source) were a more valuable source for us, because we could actually use them and test their functionalities against the list of requirements. Some of the inaccessible tools were also valid for analysis since their functionalities have been described concretely in the literature. We finalized the list of tools for analysis using the following criteria:

- The tool should be publicly accessible. For example, ADkwik [18] that is covered by previous tool evaluation studies [25] would be a candidate for our analysis, but it is not accessible anymore. However, there are some tools that are under development and therefore are not released yet, but their functional requirements are concretely described in literature; such tools do meet the criterion. One such tool is analyzed in Section 5.
- The installation and usage of the tools should be straightforward.
- The tool should be representative for its domain. For example if tool A and B exist for capturing the decisions, and tool B covers all features of tool A, we just choose tool B.

The third and last step was to summarize the tool analysis results. Based on the analysis results, we establish a vision for developing a tool that supports post-processing the captured architectural knowledge and developing a design guide.

<sup>1</sup> A design guide is a reusable asset containing knowledge about architectural decisions required in a particular domain [19]. As a reusable asset [22], a design guide has been curated, edited and quality assured for readability and reuse. We refer to this curation and editing as knowledge post-processing.

## IV. FUNCTIONAL REQUIREMENTS

This section describes functional requirements for developing an architecture guidance modeling tool. We will use the functional requirements to analyze the existing tools later (in Section 5). In the following, we will define some terms that are essential for describing the functional requirements first. Next, we will describe the actual functional requirements.

### A. Definitions

The required tool for supporting architecture guidance modeling should create and maintain a knowledge base (KB) that contains architectural entities. The tool should be able to post-process the entities (see Section 2 for examples) and enhance them into a design guide. Inspired by our previous work [19], we define the following entities that can be added to a KB:

- **Issue:** Any design issue that may occur in a software development project. It includes different properties mainly name, problem description, decision drivers and solution alternatives. Each alternative includes pros and cons, known uses and related background. For example in an enterprise architecture, an issue can be “enterprise integration pattern for designing the message channel between system A and system B”. The alternatives of the issue are “point-to-point-channel” and “publish-subscribe-channel”.
- **Decision:** A decision inherits its properties from an issue, but adds the outcome of the decision (chosen alternative and the rationale behind that). Therefore an issue can be converted to a decision by adding the outcome and vice versa. In the example we provided, the outcome may include “the publish-subscribe-channel” as the chosen alternative and “high number and change rate of the data sinks” as the rationale of the decision.
- **Group:** A group entity is an aggregation (or assembly) of other entities. An example for the usage of this container concept is a software project that includes some sub-projects and each sub-project includes issues and decisions.

This structure is not the only way of modeling the architectural entities. Tools can apply other metamodels such as those presented in [24] or [27].

### B. Functional Requirements

We categorize the functional requirements in two groups: 1) create and maintain knowledge and 2) consume knowledge. However, these categories have some overlaps and some requirements can belong to more than one category.

#### 1) Create and Maintain Knowledge

- **AddE** – Add an entity [6][17][25]: Insert an entity to KB. The following features are required:
  - *Rich text editor*
  - *A tag field (or a semantic-based approach) to make search easier*

- *Entity identification*
- *Entity name*
- *Entity description*
- *Entity stakeholders*
- *Entity version*
- *Entity confidentiality*
- *Issue level* (e.g. conceptual, technology, vendor asset [21])

Before the tool inserts an entity to KB, it should first search for available related entities and if there are some, it should suggest them to the user. If the user finds that the entity is already available in KB, (s)he can decide to cancel the procedure. This helps to reduce the redundancy. Sometimes the entity is not already available, but the search brings some related entities and the user can connect the new entity to the related ones (CnctE).

- **UpdE** – Update an entity [17][25]: Update an available entity. The features that are required for AddE apply here as well.
- **RmvE** – Remove an entity [17]: Remove an entity from KB. It should clean up all of the relations of the entity to the other entities.
- **MovE** – Move an entity [17]: Move an entity from one group to another group.
- **CnctE** – Connect (Relate) to an entity: Connect an entity to other entities (examples are issue to issue, issue to decision, issue to group).
- **UnlkE** – Unlink an entity: Unlink an entity from its parent or from its related entities without deleting the entity.
- **RevE** – Review an entity [6][25]: If an entity is supposed to be reviewed before inserting to KB, it should be sent to the reviewers. The reviewers should validate the entity before a specific time. Then based on the rates or opinions the reviewers give to the entity, the entity can be rejected, inserted (or edited and inserted) to KB.
- **ImpE** – Import an entity [17]: Import an entity from a file (for example a XML or JSON file) or a URL.
- **AnmE** – Anonymize an entity [17]: Sometimes an entity can be reused or shared or exported, but the project-specific information should not be shared with the others. This feature replaces the project-specific terms with a pseudonym.
- **MkeD** – Make a decision (convert an issue to a decision) [17]: When a decision on an issue is made, the decision should be inserted to KB. This functionality adds an outcome part to the issue and converts it into a decision.
- **GnrI** – Generate an issue from a decision: Sometimes a decision is made, but the related issue is not in KB. This functionality generates an issue from a decision by removing the outcome part of the decision entity

(and adding a possible recommendation). This feature is essential for upgrading decisions to guides (UpG).

- UplD – Upload documents [17]: An entity may include background information either as a link or as a document. This feature uploads a document to the entity.
- NtfS – Notify stakeholders [25]: Sometimes an entity would have more than one owner (stakeholder). This feature reports any changes to the entity to all stakeholders. Stakeholders should be able to disable/enable this feature. The implementation examples of this requirement are RSS (Rich Site Summary) and Atom [4].
- Conf – Configure the tool: The metamodel (default profile) that are required to insert an entity (such as entity version, entity stakeholders, entity confidentiality, issue level and so on; check AddE for more details), should not be fixed and unchangeable. This functionality customizes the attributes based on the project or organization needs.

## 2) Consume Knowledge

- SrhE – Search an entity [17][25]: Search KB for a group, an issue or a decision. The search can be done based on an entered text or by choosing an entity to find the relevant entities. Advanced search to limit the search results based on level, confidentiality, project, etc. should be supported.
- LstE – List entities: Make a list of all entities of a group.
- NKB – Navigate the knowledge base [17]: The user should be able to navigate between the groups, their sub-groups and their issues and decisions.
- ViwE – View an entity [6][17][25]: When the user finds the list of entities by searching them, or navigating KB, (s)he should be able to view each entity and its properties.
- RuseE – Reuse an entity [6][25]: Choose an entity in group A and copy it into group B. The confidentiality (access permission) of the entity should be checked first. The owner of group B might be able to view the entity of group A, but not to reuse it.
- ExpE – Export an entity [17]: Export an entity to a file. The confidentiality of the entity should be checked first: if export is not allowed by the owner of the entity, the export procedure should be rejected. If anonymized export is allowed, the entity should be anonymized first and then be exported. If export is allowed unconditionally, the entity can be exported without any pre-processing.
- ShrE [6][25] – Share an entity: Send the link of an entity to other stakeholders by email notification. First the confidentiality of the entity should be checked. Sharing is possible only if the entity is public or other stakeholders have access to the group.

- UpgG – Upgrade to guide: Convert past architectural knowledge into guidance for the future. First, the tool will ask about anonymization. If the user requests anonymization, the feature will anonymize all entities of the group (AnmE), otherwise leave them unchanged. The next step is to look up each decision and its related issue. If both a decision and its related issue are available, the feature will remove the decision (RmvE). If only the decision is available, it will generate an issue from the decision (GnrI) and remove the decision (RmvE) afterwards. The final result is a group and all of its sub-groups and each sub-group includes a list of design issues and each issue has some alternatives. This can be shared (ShrE) or exported (ExpE) as a design guide. Assume that a software developing firm has a project for developing a system for customer A. They have captured the decisions in a group called project A. The group includes various sub-groups (A1, A2, A3, etc.). Now by using this feature, they will have a list of issues and alternatives for each of the sub-groups and they can use it as a guide for making decisions in a similar project for developing a system for customer B. The architects and designers involved in the new project could be different, but the knowledge from previous project is reused in a structured manner.
- DAPI – Documented application programming interface: The tool should provide a public documented API to make it possible to be integrated with other architectural knowledge management or design modeling tools.

In the next section, we present the results of analysis of existing tools with respect to the mentioned functional requirements.

## V. ANALYSIS RESULTS

This section reports on the results of our analysis of existing tools and research prototypes with respect to the functional requirements from Section 4. First, we briefly introduce the five tools that are nominated for the analysis, and then we present the functional requirements that are satisfied by these tools.

1) *SAW*. Software Architecture Warehouse (SAW) is a Web-based tool to capture, manage and analyze architectural knowledge. It is implemented to help the entire software architecture design team achieve situational awareness about architectural decisions [15].

2) *Decision Viewpoints*. Decision Viewpoints is a documentation framework for architecture decisions. It uses the conventions of ISO/IEC/IEEE 42010 [26]. A tool is developed supporting the framework as an add-in for Sparx Systems' Enterprise Architect [5].

3) *AREL*. Architecture Rationale and Elements Linkage (AREL) is a Sparx Systems' Enterprise Architect plug-in that

creates architectural design with a focus on design rationale [3].

4) *SEURAT*. Software Engineering Using RAtionale system (SEURAT) is an Eclipse plug-in that aims to manage architectural knowledge from requirements to source code [8].

5) *Eclipse Process Framework (EPF)*<sup>2</sup>. EPF is an Eclipse-based method creation tool. In EPF, knowledge creation takes place in the tool; knowledge consumption, on the other hand, can be done in a Web browser.

Table I. shows which functional requirements are supported and which are not supported or partially supported by the introduced tools.

TABLE I. ANALYSIS OF TOOLS IN A NUTSHELL<sup>3</sup>

FR	SAW	Decision VP	AREL	SEURAT	EPF
AddE	Partially	Partially	Partially	Partially	Partially
UpdE	Yes	Yes	Yes	Yes	Yes
RemE	Yes	Yes	Yes	Yes	Yes
MovE	No	Yes	No	Yes	Yes
UlnkE	Yes	Yes	Partially	No	Yes
CnctE	Yes	Yes	Yes	No	Yes
RevE	No	No	No	No	Partially
ImpE	Partially	Partially	No	No	Yes
AnmE	No	No	No	No	No
MkeD	Yes	Yes	Yes	Yes	No
GenI	No	No	No	No	No
UpID	No	Yes	Yes	No	Yes
NtfS	Partially	No	No	No	No
Conf	No	No	No	Partially	No
SrhE	No	Yes	Yes	No	Yes
LstE	Yes	No	No	No	Yes
NKB	Yes	Yes	Yes	Yes	Yes
ViwE	Yes	Yes	Yes	Yes	Yes
RuseE	Partially	Yes	Yes	No	Yes
ExpE	Partially	Partially	No	No	Yes
ShrE	Partially	No	No	No	Yes
UpG	Partially	No	No	No	No
DAPI	No	No	No	No	Yes

<sup>2</sup> <http://projects.eclipse.org/projects/technology.epf>

<sup>3</sup> Detailed evaluation results omitted due to space constraints, but are available upon request.

## VI. VISION FOR FUTURE RESEARCH

In the previous section, we presented the results of our analysis. It showed the functional requirements that are supported by the available tools and research prototypes. Based on the data Table I. provides (the functionalities that are not focused by the available tools), we establish some directions for the next steps towards tool developing for architectural knowledge reuse and architecture guidance modeling:

1) *Separating issue (decision required) from decision (decision made)*: A design guide is mainly a list of design issues (decisions required) and their possible solutions (alternatives). To create a guide from captured decisions (decisions made), issue and decision should be separated and the tool should provide the possibility to generate an issue from a decision (GnrI). This will also make organizations less reluctant to share their knowledge with a community; because it guarantees that only the issue and its alternatives will be shared with the community and others will not be informed about their decision (chosen alternative).

2) *Providing default profile*: One of the main reasons architects state for their unwillingness to use architectural capturing tools is the time limitations [9]. To overcome this, tools should make capturing knowledge less time consuming. One of the solutions is providing a default profile for adding entities to the knowledge base.

3) *Providing knowledge confidentiality*: As we mentioned earlier, organizations are not eager to share all of their architectural knowledge with the community. There can be even a situation that in one organization, the knowledge of one project should not be shared with other projects. Therefore the confidentiality level of an entity should be defined for adding the entity to the knowledge base. The tool should always consider the confidentiality and intellectual property rights level of an entity before sharing, reusing or exporting the entity or creating architecture guidance (e.g., “open”, “copyright protected”, “company-internal”, and “confidential”).

4) *Configuring metadata*: Users should be able to customize the metadata (attributes profile) based on their organizational policies and concerns. IEC/IEEE/ISO 42010 is one, but not the only template to be supported (many more have been defined, e.g. [10]).

5) *Considering semantic tags*: The architectural knowledge base can grow very fast. Navigating a large knowledge base can be painful, e.g. if it takes a long time to find a knowledge entity. Providing semantic tags will make searching the knowledge base easier and more precise.

6) *Searching the knowledge base before inserting new knowledge*: To create a useful yet concise architecture guidance it is essential to reduce the redundancy of knowledge. To reach that, the tool should search the knowledge base in advance to inserting any new knowledge. It is also useful for finding relevant knowledge and connecting them together.

7) *Being consistent with real world situation*: In reality, large organizations develop software within various projects and sub-projects. The design guide would be more usable if it

was categorized into projects and sub-projects. Therefore grouping the entities of knowledge base to projects and sub-projects should be provided.

8) *Anonymizing the knowledge*: Rather than separating issues from decisions, anonymizing the knowledge also makes organizations more eager to share their knowledge with the community (see Section 2 for an example of a required anonymization).

9) *Providing programming interface*: The activities related to architectural knowledge management are very wide and it is not possible to have a holistic tool that supports all activities. The focus of the proposed tool in this research is on reusing architectural knowledge and enhancing a design guide. The tool should therefore provide an interface (API) to make it possible to be integrated with other architectural knowledge management or design modeling tools such as general-purpose wiki engines and Unified Modeling Language (UML) tools.

## VII. CONCLUSIONS

Reusing architectural decisions as design guides gives these decisions a more proactive role and therefore makes decision management more appealing and relevant to practitioners. In this paper, we leveraged our industrial experiences, our previous research work and also the current literature in the architectural knowledge community to establish functional requirements for future knowledge management tools that enhance architectural decisions to design guides. With respect to the functional requirements, we analyze representative tools and research prototypes. We reported that the available tools and research prototypes have made significant contributions in the area of architectural knowledge capturing, but still require a number of extensions so that the captured decision can serve as design guides in practice. We finalized the paper with a vision for method integration and tool improvement.

In the next step, we are going to evolve our design guidance enhancing framework to decrease the time and effort of design guidance generating by applying automatic information extraction approaches. The extracted architectural entities will feed the knowledge base (KB) in a more efficient way. We also intend to extend and integrate our method into existing and emerging tools (our own tools and those developed in the research community) – applying the vision we established in this paper.

## REFERENCES

- [1] A. Jansen, and J. Bosch, "Software architecture as a set of architectural design decisions", WICSA 2005, pp. 109–120, 2005.
- [2] A. Tang, P. Avgeriou, A. Jansen, R. Capilla, and M. A. Babar, "A comparative study of architecture knowledge management tools", JSS 83(3), pp. 352–370, 2010.
- [3] A. Tang, Y. Jin, and J. Han, "A rationale-based architecture model for design traceability and reasoning", JSS 80(6), pp. 918–934, 2007.
- [4] B. Hammersley, "Developing feeds with RSS and Atom", O'Reilly Media, Inc., 2005.
- [5] C. Manteuffel, D. Tofan, H. Koziolok, T. Goldschmidt, and P. Avgeriou. "Industrial implementation of a documentation framework for architectural decisions". WICSA 2014, pp. 225-234. IEEE, April 2014.
- [6] H. C. Tan, P. M. Carrillo, C. J. Anumba, M. Asce, N. D. Bouchlaghem, J. M. Kamara, and C. E. Udejaja, "Development of a methodology for live capture and reuse of project knowledge in construction", Journal of Management in Engineering 23(1), pp. 18–26, 2007.
- [7] ISO, "ISO/IEC 42010: Systems and software engineering - Recommended practice for architectural description of software-intensive systems", 2007.
- [8] J. Burge and D. Brown, "SEURAT: Integrated rationale management", ICSE 2008, pp. 835–838. ACM, 2008.
- [9] J. F. Hoorn, R. Farenhorst, P. Lago, and H. van Vliet, "The lonesome architect", JSS 84(9), pp. 1424–1435, 2011.
- [10] J. Tyree, and A. Akerman, "Architecture decisions: demystifying architecture", IEEE Software 22(2), pp. 19-27, 2005.
- [11] L. Aggestam, and P. Backlund, "Strategic knowledge management issues when designing knowledge repositories", ECIS 2007, pp. 528–539, 2007.
- [12] M. A. Babar, T. Dingsøyr, P. Lago and H. van Vliet, "Software architecture knowledge management: theory and practice", Springer, 2009.
- [13] M. Anvaari, R. Conradi, and L. Jaccheri, "Architectural decision-making in enterprises : preliminary findings from an exploratory study in Norwegian electricity industry", ECSA 2013, pp. 162–175, 2013.
- [14] M. Biehl, "Literature study on design rationale and design decision documentation for architecture descriptions", 2010.
- [15] M. Nowak and C. Pautasso, "Team situational awareness and architectural decision making with the software architecture warehouse", ECSA 2013, 2013.
- [16] M. Shahin, P. Liang, and M. R. Khayyambashi, "Architectural design decision: existing models and tools", 2009 Joint WICSA/ECSA, pp. 293–296, 2009.
- [17] N. Schuster, "ADkwik – A collaborative system for architectural decision modeling and decision process support based on Web 2.0 technologies", Stuttgart Media University, 2007.
- [18] N. Schuster, O. Zimmermann, C. Pautasso, "ADkwik: Web 2.0 Collaboration System for Architectural Decision Engineering", Proceedings of the Nineteenth International Conference on Software Engineering & Knowledge Engineering (SEKE 2007), Knowledge Systems Institute Graduate School, 2007. Pages 255-260.
- [19] O. Zimmermann, "Architectural decisions as reusable design assets", IEEE Software 28(1), pp. 64-69, 2011.
- [20] O. Zimmermann, T. Gschwind, J. Küster, F. Leymann, and N. Schuster, "Reusable architectural decision models for enterprise application development", QoSA 2007, pp. 157-166, 2007.
- [21] O. Zimmermann, U. Zdun, T. Gschwind, and F. Leymann, "Combining pattern languages and architectural decision models into a comprehensive and comprehensible design method", WICSA 2008, pp. 157-166, 2008.
- [22] Object Management Group, "Reusable Asset Specification (RAS) Version 2.2.", Available online at: <http://www.omg.org/spec/RAS/2.2>, 2005.
- [23] P. Avgeriou, P. Kruchten, P. Lago, P. Grisham and D. Perry, "Architectural knowledge and rationale: issues, trends, challenges." ACM SIGSOFT Software Engineering Notes 32.4, pp. 41-46. 2007.
- [24] P. Kruchten, "An ontology of architectural design decisions in software intensive systems", In 2nd Groningen Workshop on Software Variability, pp. 54-61, 2004.
- [25] P. Liang, and P. Avgeriou, "Tools and technologies for architecture knowledge management. (M. Ali Babar, T. Dingsøyr, P. Lago, & H. van Vliet, Eds.), pp. 91–111, 2009.
- [26] U. van Heesch, P. Avgeriou, and R. Hilliard, "A documentation framework for architecture decisions", JSS 85(4): pp. 795-820. 2012.
- [27] U. van Heesch, P. Avgeriou, and R. Hilliard, (2012). "Forces on architecture decisions: a viewpoint", 2012 Joint WICSA/ECSA, pp. 101-110, 2012.

# Aspect-Oriented Secure Connectors for Implementation of Secure Software Architecture

Chase Baker  
Department of Computer Science  
Texas Tech University  
Lubbock, USA  
chase.baker@ttu.edu

Michael Shin  
Department of Computer Science  
Texas Tech University  
Lubbock, USA  
michael.shin@ttu.edu

**Abstract** - This paper describes aspect-oriented secure connectors for implementing secure software architecture for distributed business applications. A secure connector for secure software architecture can be designed separately from application business components by considering different communication patterns between the components as well as security services required by application components. In this paper, the secure connector is implemented as an aspect-oriented secure connector separately from application business components. The aspect-oriented secure connector is structured with both a security service aspect and a communication pattern aspect that are separated from each other. Once aspect-oriented secure connectors are implemented, they can be reused for different applications with the security service and communication pattern aspects required between application components. In this paper, aspect-oriented secure connectors are used to implement an electronic commerce application.

**Keywords** – *aspect-oriented secure connector; security service aspect; communication pattern aspect; separation of concerns; aspect oriented programming*

## I. INTRODUCTION

Software solutions for distributed business applications are becoming increasingly complex in nature, due to the inclusion of various crosscutting concerns such as security, synchronization, and logging. These concerns are necessary for implementation in such applications because of the progressively more significant concerns that may be required for their intended uses. As a result, it is vital that these software solutions be implemented in such a manner that their implementation is as untangled as possible in order to reduce complexity and increase reusability. One way to avoid tangling requirements is to design and implement the software such that there is a clear separation of crosscutting concerns from application concerns.

Crosscutting security concerns can be designed as secure connectors [15] separately from application components. Security mechanisms are encapsulated in secure connectors that provide security services as well as communication patterns for the applications. These secure connectors are modular and more maintainable and reusable than tangled security and application logic in design found in systems without a clear separation.

To achieve this clear separation of concerns in implementation, secure connectors need to be implemented separately from application logic. Aspect-Oriented Programming (AOP) provides a powerful way to implement

such crosscutting concerns in implementation. Crosscutting security concerns can be implemented as security aspects in AOP, which are completely separated from the application logic of applications. Secure aspects are only executed when the services they provide are needed by the application.

This paper describes the mapping and implementation of aspect-oriented secure connectors from secure connectors designed for secure software architectures in distributed business applications. A mapping scheme is described to demonstrate how aspect-oriented secure connectors are implemented from secure connectors in design. The secure connector is implemented as an aspect-oriented secure connector separately from application business components. In particular, the aspect-oriented secure connector is structured with both a security service aspect and a communication pattern aspect that are separated each other. The aspect-oriented secure connectors are validated with the implementation of an electronic commerce application.

This paper is organized as follows. Section II covers related works to the current research. Section III describes the mapping of secure connectors in design to aspect-oriented secure connectors in implementation. Section IV contains the validation of our approach with a case study. Finally, section V serves as a conclusion to the paper and outlines future work.

## II. RELATED WORK

Separation of concerns is one of the major topics in software engineering today. Past research has proven there is a need to clearly separate concerns in complex systems in order to promote modularity, reusability, and maintainability [1]. The work provided an approach to modeling complex systems by ensuring that security and core business logic was separated in the requirements and design models during early stages of software development. Other work has begun the discussion on how to model complex systems with clear crosscutting concerns [2]. One of the proposed methods of modeling such systems includes utilizing aspect-oriented techniques to ensure a separation of crosscutting concerns. However, the work stresses the difficulty of finding a standard design model because of UML's informal nature. The benefits of an aspect-oriented approach have been studied extensively, mainly focusing on the maintainability and modularity of such systems [3]. The results of the research discovered that an aspect-oriented approach results in smaller, less complex, and far more modular implementation as well as shortened maintenance cycles. For

Java specifically, the AspectJ extension has been developed to provide Java support for modular designs to handling crosscutting concerns throughout a system with aspect-oriented methodologies [4].

However, there currently is no standard for representing systems with an aspect-oriented notation, and work has been done to prove the need for formal modeling of such systems to ensure the accuracy of the designs [5]. Extensions to the current UML framework with aspect-oriented concepts have been proposed, specifically with crosscutting concerns being considered early in design and then mapped to programming models with automatic techniques [6]. These extensions require aspects to be explicitly defined early in the design process in order to avoid a divergence during later mapping. Other approaches have been proposed, including aspect-oriented approaches to model-driven development [7] and even for larger, more general extensions to UML such as the Aspect Modeling Language [8]. The Aspect Modeling Language offers approaches for modeling aspect-oriented concepts in early design, requirements, and architecture models through code generation.

Other work has been done to propose aspect-oriented modeling notation at specific points during the software engineering process including early design [9], the requirements level [10], and even for general activity modeling [11].

While there exist proposals for modeling aspect-oriented systems in design, there is still a lack of thorough mapping schemes to evolve designs into implementation. Research has been done to stress the need for new mapping schemes for extending these early design models into implementation [12].

Developing such mapping schemes is not a trivial task, and work has been done to address the challenges one may face when mapping design models to implementation code [13].

Previous work directly related to the research in this paper has proposed a method for separating security concerns in design to security aspects in code [14]. This work provides an approach to map security components in design to secure aspects in implementation to promote reuse, reduced complexity, and increased maintainability during the implementation phase. Work has also been done to separate crosscutting concerns in the design phase using secure connectors while developing secure software architectures [15]. Such an approach promotes the separation of crosscutting concerns from core business logic by keeping the crosscutting concerns isolated in secure connectors. The work done in this paper focuses on an extension of that idea, specifically when secure connectors encapsulate multiple perspectives, such as security service and communication pattern perspectives, and there is a need to map such perspectives into separate aspects in implementation.

### III. FROM SECURE CONNECTORS TO ASPECT-ORIENTED SECURE CONNECTORS

Aspect oriented programming provides developers with a tool to separate various crosscutting concerns in their systems from core application functionality. It does this by promoting the separation of each crosscutting concern into its own aspect. An aspect in AOP can be broken down into four parts: the aspect itself, a join point(s), a pointcut(s), and advice.

An aspect can be viewed as a feature of a system that might be used at various points throughout a system’s implementation.

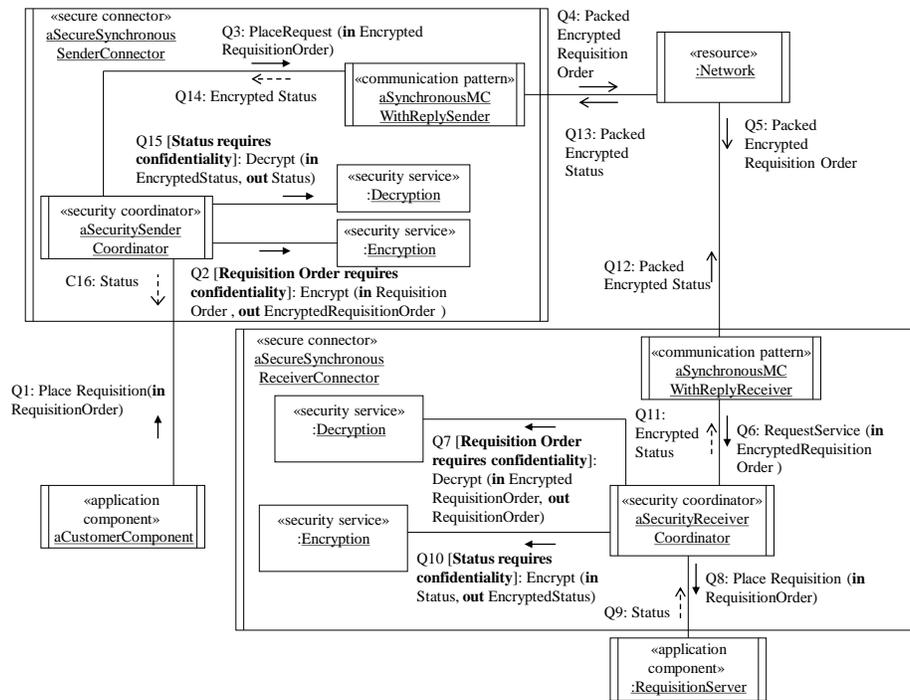


Figure 1. Applying Secure Synchronous Connector with Confidentiality Security Service in the Business to Business (B2B) Electronic Commerce Application

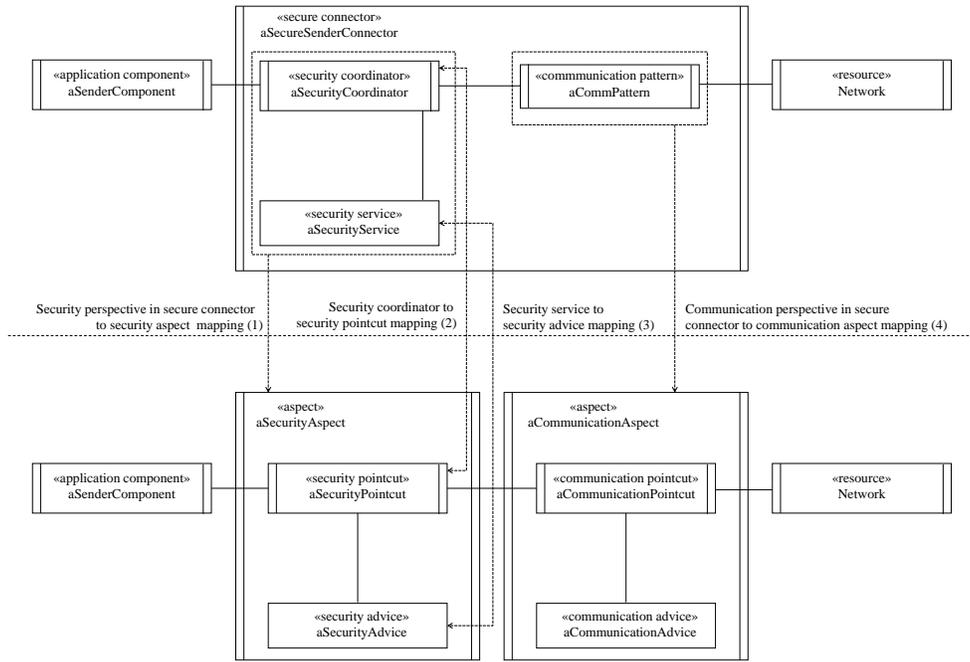


Figure 2. Mapping of Secure Connector with Security and Communication Perspectives to Security and Communication Aspects

For instance, confidentiality is used at multiple points in systems that require encryption and decryption of data to keep information secure. In this case, confidentiality would be a crosscutting concern of the system, as it deals more with security logic than the application logic of whatever the system might be.

An aspect has two important components that provide it with its functionality, namely the pointcut(s) and advice. Pointcuts refer to well-defined moments of execution in a system, commonly referred to as join points. These join points can be object initializations, method calls, or method executions for instance. A pointcut clearly defines the collection of join points at which an aspect should be injected for use. The advice of an aspect provides the additional code or methods required to give the aspect the functionality desired. In the confidentiality aspect example, the advice would include the algorithms and methods needed to apply encryption and decryption to the necessary data.

Now that our definition of an aspect is defined, we can determine how to map secure connectors in software architecture into aspect-oriented secure connectors in implementation. Consider a use case in a business to business (B2B) electronic commerce application that is responsible for placing a requisition order. This use case may require that the requisition order be encrypted before it is packaged and then sent to the server for storage via some communication pattern, such as synchronous communication with reply.

Fig. 1 depicts the UML communication diagram for placing a requisition order. An application component, aCustomerComponent, calls the Place Requisition method to send its order to a location on the network. It then passes through a secure connector that determines the requisition order requires confidentiality. The system then applies encryption to the requisition order. Upon receiving encryption, the Place Requisition method resumes and the secure connector

determines the requisition order requires a communication pattern be applied to it. The system prepares the requisition order for a synchronous communication with reply protocol and then sends the requisition order to the network where it can be taken to the server.

The mapping scheme of secure connectors to aspect-oriented secure connectors can now be considered. Notice that the security service is applied when the security coordinator in the secure connector has determined that the requisition order requires encryption or decryption. The responsibility of the security coordinator is identical to the behavior of an aspect's pointcut, which simply is used to determine when an aspect is required during runtime. Also, the security service, aSecurityService for example, is the implementation for carrying out that security service. This is identical to the contents of an aspect's advice. Similarly, the second perspective of the secure connector, a communication pattern, can be divided into two parts to represent it in a form of an aspect. The communication pattern concern has to decide if it is necessary to apply the pattern (pointcut) and if so, it should apply the communication pattern (advice). Thus we can notice that the contents of a secure connector, aSecureSenderConnector, are mapped to multiple aspects such as a security service aspect and a communication pattern aspect.

Taking what we just discussed, we can now construct a mapping scheme from secure connectors in software architecture to aspect-oriented secure connectors in implementation. Fig. 2 depicts the mapping scheme for this approach. Step one consists of mapping the secure connector's security perspective into an aspect object. Step two maps the security coordinator from the security perspective to security pointcuts in the aspect. Step three maps the security service from the security perspective to security advice in the aspect. Finally,

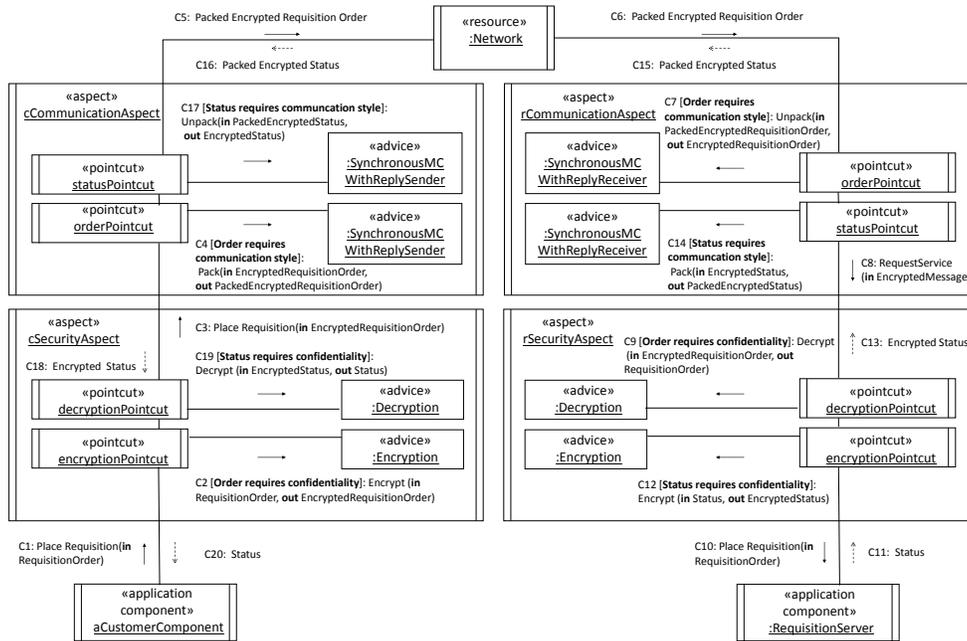


Figure 3. Secure Synchronous Message Communication with Reply featuring Security and Communication Aspects

step four divides the communication perspective into a communication pointcut to determine when to apply the communication pattern in the newly created communication advice.

With the mapping scheme developed, we can focus on mapping the secure connector from Fig. 1 to an aspect-oriented secure connector. In this model, we can see a number of crosscutting concerns that are represented by secure connectors with multiple perspectives. Within the customer-side connector, we have identified both security and communication perspectives. Within the server-side, we can identify the same security and communication perspectives. Therefore in this model, we have identified two crosscutting concerns that are candidates for evolving into aspects: security (confidentiality) and communication (synchronous with reply).

Looking closer at security, we can notice two different security services it offers: encryption and decryption. Therefore we will need to have two different advice blocks with their respective pointcuts featured in the mapped model. Encryption will be applied when an encryption pointcut determines the information requires encryption, and similarly decryption will be applied if a decryption pointcut determines the information requires decryption. This is applicable for both the sender and receiver sides of the routine.

For communication on the customer side, notice there exists a communication pattern object, aSynchronousMCWithReplySender. It is important to investigate what this component is trying to do. Looking at our diagram in Fig. 1, the communication pattern object will take the encrypted order and package it according to a predefined communication style (here being synchronous message communication with reply). Therefore, we can split this component into two parts during mapping: a pointcut to determine when to apply a specific communication style and a

block of advice containing the methods necessary to apply the communication style to the order. Similarly on the receiver side, aSynchronousMCWithReplyReceiver will be broken down into the same two objects, a pointcut to determine when a communication style is needed and advice to determine how to handle data that is being received.

Fig. 3 depicts the communication diagram with aspects mapped from the communication diagram with secure connectors in Fig. 1. The secure connector is broken down into two secure aspects, as we determined there were two separate crosscutting concerns contained within the connector. For each side of the system, we end up with a security aspect (cSecurityAspect, rSecurityAspect) and a communication aspect (cCommunicationAspect, rCommunicationAspect). For the security aspects, we have a pointcut to determine if decryption is required by the information being received (decryptionPointcut) and also a pointcut to determine if encryption is required by the information being sent (encryptionPointcut). Similarly in the communication aspects, we have two pointcuts to determine how to handle specific data types, orders (orderPointcut) and statuses (statusPointcut).

Immediately one can see the multiple benefits to this approach. The model now has all crosscutting concerns completely separated from any core application logic in the system. The modeled aspects can be easily removed and reused in other parts of the application or even in entirely new applications with very little modification required. This structure also allows the reader to clearly see the various crosscutting concerns contained in the system without much effort. This greatly reduces the effort required to comprehend more complex architectures.

#### IV. VALIDATION WITH CASE STUDY

It is important that the separation of concerns be prevalent in both the design and the implementation of the system, otherwise

```

public aspect cSecurityAspect {
    RequisitionOrder Encrypt(RequisitionOrder Order)[]
    pointcut encryptionPointcut(RequisitionOrder Order) :
        execution(* aCustomerComponent.PlaceRequisition(RequisitionOrder)) && args(Order);
    RequisitionOrder around() : encryptionPointcut(RequisitionOrder) && !within(cSecurityAspect +)
    {
        proceed();

        Object[] args = thisJoinPoint.getArgs();
        RequisitionOrder Order = (RequisitionOrder) args[0];

        RequisitionOrder EncryptedRequisitionOrder = Encrypt(Order);

        return(EncryptedRequisitionOrder);
    }
}

```

Figure 4. Security Aspect Implementation with Confidentiality Mapped From Security Perspective in Secure Connector Design

the benefits of the approach would be lost. With that in mind, a case study with a distributed business to business (B2B) electronic commerce application is implemented using the Java programming language and AspectJ, an aspect-oriented extension to assist in the separation of crosscutting concerns.

We will focus on the previous scenario of a customer component contacting a requisition server in order to send a requisition order in the following example. Specifically, we will be implementing the scenario that we previously mapped in Fig. 3, which is the synchronous message communication with reply and confidentiality security service.

This scenario makes use of a secure connector with two perspectives on both the sender and receiver sides. The perspectives are communication (synchronous message communication with reply) and security (confidentiality). We previously discussed the process used to map the perspectives from the secure connectors in Fig. 1 to the aspect-oriented secure connectors in Fig. 3. Using the mapping concepts, we can develop aspect-oriented secure connector code implementing our design.

Fig. 4 depicts the mapped implementation for the security aspect on the customer side, specifically the objects pertaining to the encryption of the order. In code, we have the security aspect, cSecurityAspect. This aspect will contain pointcuts and advice for both encryption and decryption. First, we define the pointcut for encryption. This representation of the pointcut reads "at the execution of the PlaceRequisition() method from the aCustomerComponent class with the parameter Order of RequisitionOrder type, run the advice for the encryptionPointcut."

The advice is contained within the block of code. In this example, around-type advice is used. Around-type advice is powerful in AspectJ, as it allows the passing of objects between aspects and other java packages. In this example, we have a proceed() statement that allows the PlaceRequisition() method to finish processing before we apply our advice. Once the PlaceRequisition() method has prepared the order, our advice then retrieves it from aCustomerComponent for use within the aspect. When it has been retrieved, our advice applies encryption to the order using an Encrypt() method we define elsewhere within our aspect. When encryption has been completed, we then

```

public aspect cCommunicationAspect {
    RequisitionOrder Pack(RequisitionOrder EncryptedRequisitionOrder)[]
    pointcut orderPointcut(RequisitionOrder EncryptedRequisitionOrder) :
        execution(* aCustomerComponent.PlaceRequisition(RequisitionOrder)) && args(EncryptedRequisitionOrder);
    RequisitionOrder around() : orderPointcut(RequisitionOrder) && !within(cCommunicationAspect +)
    {
        proceed();

        Object[] args = thisJoinPoint.getArgs();
        RequisitionOrder EncryptedRequisitionOrder = (RequisitionOrder) args[0];

        RequisitionOrder PackedEncryptedRequisitionOrder = Pack(EncryptedRequisitionOrder);

        return(PackedEncryptedRequisitionOrder);
    }
}

```

Figure 5. Communication Aspect Implementation Mapped From Communication Perspective in Secure Connector Design

return the EncryptedRequisitionOrder back into the flow of the system where we retrieved it from. This same process is used to map the decryption pointcut within the aspect.

Similarly, we map the implementation for the communication aspect on the customer side, specifically the objects pertaining to the packing and sending of a service request. The mapped implementation for the communication aspect on the customer side is shown in Fig. 5. In code, we have the communication aspect, cCommunicationAspect. Let's focus on the pointcut/advice combination required for packing a requisition order with a certain communication style before it is sent to the network (here it is synchronous message communication with reply). The representation of the pointcut reads "at the execution of the PlaceRequisition() method from the aCustomerComponent class with the parameter EncryptedRequisitionOrder of RequisitionOrder type, run the advice for the orderPointcut." Similar to the customer side, the server side in Fig. 3 can be implemented based on the mapping scheme described in Section III.

## V. CONCLUSION

The mapping scheme proposed shows that it is possible to map secure software architectures, specifically those using secure connectors as a way to separate crosscutting concerns from application logic, to an implementation that features aspect-oriented concepts. The various perspectives found in the secure connectors are mapped to individual aspects in code, while maintaining the intended separation of concerns in design with added benefits in implementation.

The results of the case study using the proposed approach show that mapping the various perspectives in secure connectors to aspect-oriented secure connectors in code results in a higher degree of the separation of crosscutting concerns. This approach achieves a much greater degree of modularity and reusability from the separation of crosscutting concerns. The new aspects can also be applied to other applications that require the same crosscutting concerns, requiring only minor changes for use in such applications. The resulting implementation is also far easier to maintain, as all crosscutting concerns are isolated to their own individual aspects.

Future work could include an automated or semi-automated mapping from design models to skeleton aspects in implementation. The mapping scheme described in section 3 can be implemented as a tool. For this, the mapping scheme described in this may need to be refined. The automated or semi-automated mapping tool can be helpful for large-scale applications.

## REFERENCES

- [1] M.E. Shin and H. Gomaa, "Modeling Complex Systems by Separating Application and Security Concerns," ICECCS '04 Proceedings of the Ninth IEEE International Conference on Engineering Complex Computer Systems Navigating Complexity in the e-Engineering Age, Florence, Italy, pp. 19-28, 2004.
- [2] Y. Fu, J. Ding, P. Bording, "An Approach for Modeling and Analyzing Crosscutting Concerns", *The 5th IEEE International Conference on Service Operations, Logistics and Informatics*, Chicago, USA, pp.91-97, 2009.
- [3] A. Hovsepian, R. Scandariato, S. Van Baelen, Y. Berbers, and W. Joosen. "From Aspect-Oriented Models to Aspect-Oriented Code? The Maintenance Perspective," *AOSD '10*, New York, NY, USA, pp. 85-96, 2010
- [4] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W.G. Griswold, "An Overview of AspectJ," *ECOOP '01 Proceedings of the 15th European Conference on Object-Oriented Programming*, Budapest, Hungary, pp. 327-353, 2001.
- [5] O.Aldawud, T. Elrad, and A. Bader, "A UML Profile for Aspect Oriented Modeling," *OOPSLA 2001 Workshop on AsoC in OOS*, Tampa, Florida, 2001
- [6] I. Groher and S. Schulze. "Generating Aspect Code from UML Models," *The 4th Aspect-Oriented Software Development Modeling with UML Workshop*, 2003
- [7] A. Solberg, D. Simmonds, R. Reddy, S. Ghosh, and R. France. "Using Aspect Oriented Techniques to Support Separation of Concerns in Model Driven Development," *Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC 2005)*, Edinburg, Scotland, UK, July 2005
- [8] I. Groher and T. Baumgarth. "Aspect-Oriented from Design to Code," *Early Aspects 2004, 3rd Aspect-Oriented Software Development Conference (AOSD 2004)*, 2004
- [9] R. France, I. Ray, G. Georg, and S. Ghosh, "Aspect-oriented approach to early design modelling," *Software, IEE Proceedings - Software Vol. 151 No. 4*, pp. 173- 185, 2004
- [10] J. Araújo, A. Moreira, and I. Brito, "Aspect-Oriented Requirements with UML," *Workshop on Aspect-Oriented Modeling with UML*, 2002.
- [11] R. Miles, *AspectJ Cookbook*. Beijing; Sebastopol, CA : O'Reilly Media, 2005.
- [12] N. Noda, T. Kishi, "Implementing Design Patterns Using Advanced Separation of Concerns," *OOPSLA 2001 Workshop on AsoC in OOS*, Tampa, Florida, 2001
- [13] M. Kramer, J. Kienzle. "Mapping Aspect-Oriented Models to Aspect-Oriented Code," *MODELS'10 Proceedings of the 2010 International Conference on Models in Software Engineering*, Oslo, Norway, pp. 125-139, 2011
- [14] C. Baker, M. Shin. "Mapping of Security Concerns in Design to Security Aspects in Code," *Software Security and Reliability Companion (SERE-C), 2012 IEEE Sixth International Conference on Software Security and Reliability*, Gaithersburg, MD, pp. 102-110, 2012.
- [15] M. E. Shin, B. Malhotra, H. Gomaa, and T. Kang, "Connectors for Secure Software Architectures," 24nd International Conference on Software Engineering and Knowledge Engineering (SEKE'2012), San Francisco, July 1-3, 2012.

# Applying Random Testing to Constrained Interaction Testing

Yasuhiro Hirasaki  
Graduate School of Information  
Science and Technology  
Osaka University  
Osaka, Japan  
Email: y-hirask@ist.osaka-u.ac.jp

Hideharu Kojima  
Graduate School of Information  
Science and Technology  
Osaka University  
Osaka, Japan  
Email: hkojima@ist.osaka-u.ac.jp

Tatsuhiko Tsuchiya  
Graduate School of Information  
Science and Technology  
Osaka University  
Osaka, Japan  
Email: t-tutiya@osaka-u.ac.jp

**Abstract**—This paper discusses interaction testing using random testing. Random testing can generate test cases very fast; but directly applying this method to interaction testing may result in insufficient interaction coverage if constraints exist over parameter values. We propose a random testing approach that is tailored to constrained interaction testing to solve this problem. In this approach, if a test case that was randomly generated violates the constraints, then new test cases are systematically generated to compensate the loss in interaction coverage that would be caused by simply discarding that constraint-violating test case. The technical challenge here is how to reduce the number of those newly generated test cases. We propose a novel algorithm for this purpose and two methods that can be incorporated in the algorithm. Experimental results show that the proposed approach can generate test suites very fast and that the proposed two methods work very effectively in reducing the number of additional test cases.

**Keywords**—Random testing, combinatorial interaction testing, constraints

## I. INTRODUCTION

Recently, software systems have increasingly become highly configurable, making testing all combinations of parameters unrealistic. *Combinatorial Interaction Testing* (CIT) is known as an effective testing technique that can be used to tackle this combination explosion problem [1], [2], [3], [4], [5]. A common CIT approach is *t-way testing* which requires covering all interactions involving  $t$  parameters, where  $t$  is an integer chosen by the tester, typically two or three. The effectiveness of this technique is backed by the belief and fact that many faults can be triggered by interactions involving a small number of parameters [6], [7]. On the other hand, CIT has a drawback of large execution time required to create a test suite. Especially when the system under test (SUT) has many parameters, current test case generation algorithms may take long time to produce a test suite.

*Random testing* has started gaining attention as an alternative to combinatorial interaction testing, as it can overcome this CIT's drawback. Because of its simplicity, random testing can generate test cases very fast. Recently, Arcuri and Briand showed that when using random testing, any  $t$ -way interaction is tested with at least probability 63% even if the test suite size is the same as the size of the smallest test suite for  $t$ -way testing [8].

However, when there are constraints over the input space of the SUT, naively using random testing can result in insufficient interaction coverage. This occurs because constraints may make some particular interactions hard to kill and simply discarding constraint-violating test cases may leave these die-hard interactions untested. An illustrative example will be shown in the next section.

Our work addresses this problem, aiming at making use of the efficiency in test case generation in the presence of constraints while maintaining the probabilistic guarantee of interaction coverage. In our approach, if a randomly generated test case violates the constraints, then the test case is replaced with a collection of test cases that are systematically generated to compensate the loss in interaction coverage that would be caused by simply discarding that constraint-violating test case. The challenge here is how to reduce the number of those newly generated test cases. We devise a novel algorithm for this purpose as well as two methods that can be incorporated with the proposed algorithm. We conduct an experiment to evaluate the performance of the proposed approach.

The remainder of this paper is organized as follows. Section II describes interaction testing,  $t$ -way testing, a common strategy of CIT, and random testing. Section III describes the overview of the proposed approach. Section IV describes the algorithm that generates a collection of test cases from a constraint-violating test case. Two methods are also presented that can be incorporated into the algorithm. Section V shows the results of an experiment. Section VI describes the threats to validity of the results. Section VII concludes this paper.

## II. BACKGROUND

### A. Model of SUT

Here we describe the model of an SUT. This model is commonly used in the literature on CIT. The SUT consists of a finite set of *parameters* and a finite set of *constraints*. We let  $k$  denote the number of parameters. The  $i$ th parameter has a finite domain  $M_i$  of possible values. We assume that parameters are arranged in descending order of the size of the domains. A *test case* is a value assignment to the parameters. In other words, a test case is an element of  $M_1 \times M_2 \times \dots \times M_k$ ; that is, a vector  $(v_1, v_2, \dots, v_k)$  such that  $v_i \in M_i$ . A *test suite* is a collection of test cases.

We often need to discuss incomplete test cases where values are not specified for some parameters. Following the terminology of [4], we call such an incomplete test case a *value schema*. We use symbol “\*” to represent that a value is not specified for a parameter and call this symbol a *wildcard*. Hence a value schema is formally defined as an element of  $M_1 \cup \{*\} \times \dots \times M_k \cup \{*\}$ .

TABLE I. MODEL OF AN SUT

Parameters and domains			
OS	Browser	Protocol	DBMS
Windows 7	IE	IPv4	MySQL
Windows 8	Firefox	IPv6	Sybase
Linux			Oracle

Constraints

If OS is Linux, then Browser is not IE.

For example, consider the SUT described in Table I. This example is adopted from [9]. Here there are four parameters, i.e., OS, Browser, Protocol and DBMS. As shown in this table, these parameters take either three or two values. The vector of these values, each for one parameter, is a test case. For example, (Windows 7, Firefox, IPv4, Oracle) is a test case. A value schema can have a wild card. For example, (Windows 7, \*, IPv4, \*) is a value schema. Note that any test case is also a value schema.

Each test case must satisfy all given *constraints* to be executed. A test case is said to be *valid* if it satisfies all the constraints; it is *invalid*, otherwise. Formally a set of constraints is represented as a mapping  $c : M_1 \times M_2 \times \dots \times M_k \rightarrow \{true, false\}$  such that  $c(v) = true$  if and only if test case  $v$  is valid. This notion can be naturally generalized to value schemas. A value schema is valid if and only if it can be extended to a valid test case by setting specific values to parameters that have a wildcard.

The example shown in Table I has a simple constraint:

If OS is Linux, then Browser is not IE.

Hence (Windows 7, IE, IPv4, Oracle) is a valid test case, whereas test case (Linux, IE, IPv4, Oracle) is invalid. On the other hand, (Linux, \*, IPv4, \*) is a valid value schema because it can be extended to, for example, (Linux, Firefox, IPv4, Oracle) which is a valid test case.

An *interaction* is a combination of values, each for a different parameter. An interaction is said to be a  $t$ -way interaction if it involves exactly  $t$  parameters. A  $t$ -way interaction is identical to a value schema that has exactly  $t$  parameters on which a value is specified. Hence we say an interaction is valid if its corresponding value schema is valid. An interaction is said to be *covered* by a test case or by a test suite if it occurs in the test case or at least one of the test cases in the test suite.

### B. Combinatorial Interaction Testing

Combinatorial interaction testing (CIT) is a testing technique that requires covering all interactions of interest. A common strategy is  $t$ -way testing, which requires covering all interactions involving  $t$  parameters. In this paper we focus on the problem of covering  $t$ -way interactions.

TABLE II. TEST SUITE THAT COVERS ALL 2-WAY INTERACTIONS

Test	OS	Browser	Protocol	DBMS
1	Windows 7	IE	IPv4	MySQL
2	Windows 8	Firefox	IPv6	MySQL
3	Linux	Firefox	IPv4	Sybase
4	Windows 8	IE	IPv6	Sybase
5	Windows 7	Firefox	IPv6	Oracle
6	Windows 8	IE	IPv4	Oracle
7	Linux	Firefox	IPv6	MySQL
8	Windows 7	Firefox	IPv6	Sybase
9	Linux	Firefox	IPv4	Oracle

Table II shows a test suite that meets the requirement of 2-way testing for the above example. As shown in this table, only nine test cases suffice to cover all 2-way interactions, whereas there are a total of  $3 * 2 * 2 * 3 = 36$  test cases.

### C. Applying Random Testing to Interaction Testing

A drawback with CIT is that time taken for the test case generation becomes long when the size of the SUT increases. Applying random testing to covering interactions is one possible way to overcome this drawback. Although there are many variants of random testing, we limit our attention to the simplest one where a test case is generated by sampling an element from the input space  $M_1 \times \dots \times M_k$  uniformly randomly. This can be performed by simply picking up one value from  $M_i$  for each parameter  $p_i$ ; thus test cases can be generated very fast.

In [8], Arcuri and Briand provided some fundamental results which formally show the fault detection capability of random testing in interaction testing.

*Theorem 1:* Suppose that there are no constraints. Any  $t$ -way interaction is covered with at least probability 63% by executing a test suite if random testing is used to construct the test suite and the size of the test suite is at least  $|M_1| * |M_2| * \dots * |M_t|$ .

Proof. Directly follows from Theorems 1 and 2 in [8].

However this probabilistic guarantee does not hold if there are constraints over the input space. The authors of [8] presented an extreme example as follows. Suppose that the SUT consists of  $k$  binary parameters  $\{p_1, p_2, \dots, p_k\}$  with  $M_i = \{0, 1\}$  and the following constraint:

$$p_1 = 1 \wedge p_2 = 1 \Rightarrow p_3 = 1 \wedge \dots \wedge p_k = 1$$

In this case, the 2-way interaction  $(1, 1, *, \dots, *)$  occurs only in one valid test case, i.e.,  $(1, 1, \dots, 1)$ . Therefore the probability that this interaction occurs in a randomly generated test case is only  $1/2^k$ . This probability decreases by half as  $k$  increases by one. Hence when  $k$  is large, there is almost no chance to test this particular interaction in random testing.

## III. OVERVIEW OF THE PROPOSED APPROACH

This section describes our proposed test case generation approach. The idea of the approach is as follows: Each test case is randomly created. If the test case is valid, then it is added to the pool of test cases. Otherwise, a set of valid test cases are created from that invalid test case and added to the test case pool. The algorithm that creates the set of valid test

cases needs to guarantee that all interactions in the invalid test case occur in at least one of these new valid test cases.

The overview of the algorithm is shown below. In the algorithm, an iteration consists of two steps and is repeated  $n$  times.

- Step 0: Let  $\mathcal{V}$  be the test suite, which is initially empty.
- Step 1: Generate a test case  $v$  by uniformly randomly drawing an element from the whole test case space.
- Step 2: If  $v$  is valid, then add  $v$  to  $\mathcal{V}$ . Otherwise, call the function  $createTestcases(v, t)$  which generates a collection of valid test cases from  $v$ . Add the returned valid test cases to  $\mathcal{V}$ .
- Step 3: If Steps 1 and 2 have been iterated  $n$  times, then stop; otherwise repeat from Step 1.

The collection of test cases returned by  $createTestcases(v, t)$  must satisfy the following two properties:

- Property 1 All test cases are valid.
- Property 2 For each valid  $t$ -way interaction that occurs in  $v$ , the interaction occurs in at least one of the test cases.

In the next section, we propose a design of this function.

Now suppose that  $createTestcases(v, t)$  works correctly. Then the following theorem holds.

*Theorem 2:* Suppose that  $n \geq |M_1| * |M_2| * \dots * |M_t|$ . Then, by executing the test suite generated by the above algorithm, every valid  $t$ -way interaction is covered with at least probability 0.63.

*Proof.* By the properties of  $createTestcases(v, t)$ , the probability of a valid interaction being covered is at least equal to the probability that the interaction would be covered if the constraints did not exist. From Theorem 1 the probability is at least 0.63.

The technical challenge here is the design of  $createTestcases(v, t)$ . If the number of test cases created by this function is large, then the benefit of using the proposed approach may be canceled out, because in that case the test suite becomes large, resulting in large cost of executing it. We address the issue of the design of this function in the next section.

#### IV. ALGORITHM FOR GENERATING NEW TEST CASES

This section describes the design of  $createTestcases(v, t)$ , the function that generates a collection of valid test cases that cover all  $t$ -way interactions that occur in  $v$ . A naive algorithm could for example work as follows. For every interaction that occurs in  $v$ , check whether it is valid or not. If it is valid, then a valid test case is generated by extending the interaction. This algorithm would generate  $k * (k - 1) * \dots * (k - t) / t!$  test cases from a single invalid test case in the worst case. On the other hand, our proposed algorithm outputs only  $t + 1$  test case in the best case.

##### A. Basic Algorithm

The basic algorithm is described as follows. We assume that  $t < k$ , since the case  $t = k$  is not interesting.

Input: invalid test case  $v$

Output: collection of test cases  $C$

- Step 0 Set  $v$  to value schema  $v'$ . Initialize  $C$  to empty.
- Step 1 Let  $S$  be the set of parameters that have values in  $v'$ . Split  $S$  into  $A_1, A_2, \dots, A_{t+1}$  such that  $A_i \cap A_j = \emptyset$  ( $i \neq j$ ),  $A_1 \cup A_2 \cup \dots \cup A_{t+1} = S$ , and  $|A_i| > 0$ .
- Step 2 Create a value schema from  $v'$  by replacing the values on the parameters in  $A_i$  with a wildcard. A total of  $t + 1$  value schemas are generated.
- Step 3 For each of the  $t + 1$  value schemas, determine whether it is valid or not and perform the following process.
  - 1) If it is valid, then extend it to a valid test case by replacing wildcards in it with specific values and add the test case to  $C$ .
  - 2) If it is invalid and has exactly  $t$  values, then simply terminate this step.
  - 3) If it is invalid and has more than  $t$  values, recursively perform the process from Step 1 with  $v'$  set to this value schema.

Once all steps have terminated, the resulting  $C$  becomes the output.

Here we explain this algorithm using a concrete example. Suppose that the SUT has six binary parameters  $p_1, \dots, p_6$  which take either 0 or 1 and a constraint such that

$$p_1 = 1 \Rightarrow p_2 = \dots = p_6 = 1$$

Now suppose that test case  $v = (1, 1, 1, 0, 0, 0)$  has been randomly generated and that  $createTestcases(v, 2)$  is executed since this test case is invalid. Figure 1 schematically describes how the execution of  $createTestcases(v, 2)$  progresses.

In Step 0,  $v'$  is set to  $(1, 1, 1, 0, 0, 0)$ . In Step 1 in the first iteration of the algorithm,  $S$  is set to  $\{p_1, \dots, p_6\}$ , since all the parameters have an assigned value in  $v'$ .  $S$  is partitioned into three mutually disjoint sets:  $A_1, A_2, A_3$ . Here suppose that  $A_1 = \{p_1\}$ ,  $A_2 = \{p_2, p_3\}$ ,  $A_3 = \{p_4, p_5, p_6\}$ . In Step 2, the three value schemas shown below are derived from  $A_1, A_2, A_3$ :  $(* , 1, 1, 0, 0, 0)$ ,  $(1, *, *, 0, 0, 0)$ , and  $(1, 1, 1, *, *, *)$ . Step 3 is executed for each of the three value schemas. Value schema  $(* , 1, 1, 0, 0, 0)$  is valid, that is, it can be extended to a valid test case. The only valid test case obtained by extending this value schema is  $(0, 1, 1, 0, 0, 0)$ ; thus this test case is generated and added to collection  $C$  (Step 3-1). Value schema  $(1, 1, 1, *, *, *)$  is also valid; thus a test case obtained by extending it is generated and added to  $C$ . Test case  $(1, 1, 1, 1, 1, 1)$  is the only valid test case that can be obtained from the value schema.

In contrast, value schema  $(1, *, *, 0, 0, 0)$  is invalid, and thus the algorithm is repeated from Step 1 with  $v'$  set to  $(1, *, *, 0, 0, 0)$  (Step 3-3). In the new iteration,  $S$  becomes  $\{p_1, p_4, p_5, p_6\}$ . Suppose that  $S$  were partitioned into  $A_1 = \{p_1\}$ ,  $A_2 = \{p_4, p_5\}$  and  $A_3 = \{p_6\}$  (Step 1), resulting in value schemas  $(* , *, *, 0, 0, 0)$ ,  $(1, *, *, *, *, 0)$  and  $(1, *, *, 0, 0, *)$  (Step 2). Step 3 is executed for each of the

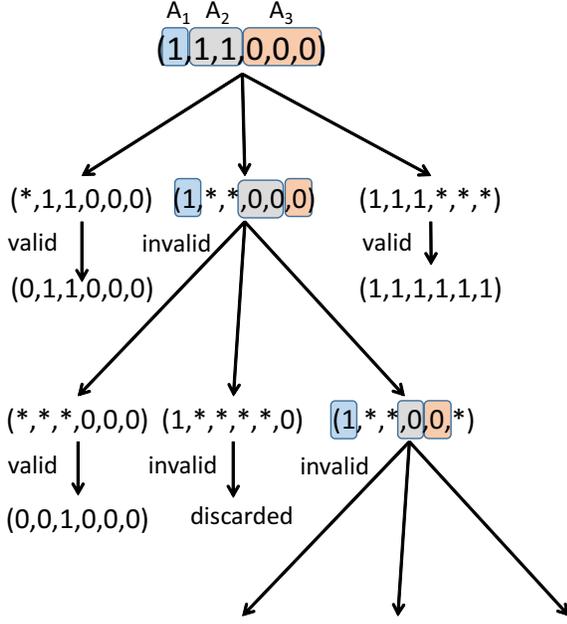


Fig. 1. Execution of  $createTestcases((1, 1, 1, 0, 0, 0), 2)$

value schemas. Value schema  $(*, *, *, 0, 0, 0)$  is valid; thus it is extended to a valid test case and added to  $C$ . There are a few possible test cases that can be extended from this value schema (e.g.,  $(0, 0, 1, 0, 0, 0)$ ). In such a case, one of them is randomly selected. Value schema  $(1, *, *, *, *, 0)$  is invalid and it has only two parameters that have an assigned value; thus this value schema is simply discarded (Step 3-2). Note that this value schema can be considered as a two-way interaction and this interaction can never be covered by any test case. Value schema  $(1, *, *, 0, 0, *)$  is invalid and thus the algorithm is repeated from Step 1 with  $v' = (1, *, *, 0, 0, *)$ .

One can show that output  $C$  satisfies Properties 1 and 2 as follows. Property 1 trivially holds because any test case in  $C$  is added to  $C$  in Step 3-1 and the test case is always a valid test case that is generated by extending a valid value schema.

The reason why Property 2 holds is as follows. If a value schema  $v'$  is invalid and has more than  $t$  values (Steps 1 and 3-3), then every valid  $t$ -way interaction occurs in at least one of the  $t + 1$  value schemas generated from  $v'$  in Steps 1 and 2. If a value schema is valid (Step 3-1), then it can be extended to a valid test case and the test case contains all  $t$ -way interactions that occur in the value schema. If a value schema is invalid and has exactly  $t$  values (Step 3-2), then the only  $t$ -way interaction that occurs in the value schema is trivially invalid.

This algorithm leaves some room for detailed design. In the rest of this section, we show two methods that can be incorporated into the basic algorithm.

### B. Avoiding Generating Redundant Test Cases

This method is to use an additional termination condition for recursion of the algorithm of  $computeTestsets(v, t)$ . This condition is that in Step 3 of the algorithm the value schema processed in the current step is identical to or “part of” a value

schema that has already occurred in the current execution of the algorithm. We say that a value schema  $v_1$  is part of another value schema  $v_2$  if every parameter that is assigned a value in  $v_1$  has the same value in  $v_2$ .

For example, in our running example illustrated in Figure 1, value schema  $(*, *, *, 0, 0, 0)$  is part of  $(*, 1, 1, 0, 0, 0)$  which has already occurred. Thus  $(*, *, *, 0, 0, 0)$  is not processed and Step 3 handling  $(*, *, *, 0, 0, 0)$  simply terminates. The correctness of the algorithm is maintained because all valid interactions in  $v_1$  are covered by the test cases derived from  $v_2$ .

### C. Strategic Parameter Set Splitting

In the basic algorithm of  $createTestcases(v, t)$ , Step 1 divides  $S$  into  $t + 1$  mutually disjoint sets, where  $S$  is the set of parameters that have already been assigned a specific value in value schema  $v$ . However, Step 1 of the basic algorithm does not specify how to partition the parameter set  $S$  into  $t + 1$  disjoint sets. A simple design choice is to uniformly split the set into  $t + 1$  sets of almost the same size.

The heuristic proposed here splits  $S$  in a different manner, aiming to reduce the number of test cases generated. The heuristic is based on two ideas. The first idea is the belief that if many of the  $t + 1$  value schemas derived from the invalid test case are valid, then the number of newly generated test cases becomes small, because the number of recursions becomes small. The second idea is that when a value schema is valid, if it has many parameters that have an assigned value, then the resulting test cases become few in number, because many interactions are covered by the single valid test case derived by extending the value schema.

Based on the ideas, we devise a greedy-type heuristic for parameter set splitting. Here we describe the heuristic using the running example. First, the first group of parameters,  $A_1$  is determined. Starting from  $(*, *, *, *, *, *)$ , parameters that have a value are gradually increased, while keeping the value schema is valid as follows.

- $(*, *, *, *, *, 0) \rightarrow$  valid
- $(*, *, *, *, 0, 0) \rightarrow$  valid
- $(*, *, *, 0, 0, 0) \rightarrow$  valid
- $(*, *, 1, 0, 0, 0) \rightarrow$  valid
- $(*, 1, 1, 0, 0, 0) \rightarrow$  valid
- $(1, 1, 1, 0, 0, 0) \rightarrow$  invalid

As a result,  $(*, 1, 1, 0, 0, 0)$  is derived and  $A_1$  is set to  $\{p_1\}$ . The second parameter group  $A_2$  is determined the same way, starting from  $(1, *, *, *, *, *)$ . However, this time this is not possible, because:

- $(1, *, *, *, *, 0) \rightarrow$  invalid

In such a case, the remaining parameter set  $\{p_2, p_3, p_4, p_5, p_6\}$  is divided into subsets of the same size. Since  $A_2$  and  $A_3$  are yet to be determined, the parameter set is split into two halves  $\{p_2, p_3\}$  and  $\{p_4, p_5, p_6\}$  and  $A_2$  is set to  $\{p_2, p_3\}$ . Finally  $A_3$  is determined as  $\{p_4, p_5, p_6\}$ , as  $A_3$  must be  $S \setminus (A_1 \cup A_2)$ .

TABLE III. RESULTS OF THE EXPERIMENT

Problem	Parameters	Constraints	$n$	Setting	Coverage (%)	Test suite size			Execution time (ms)
						Min	Ave	Max	
Apache	$2^{15}3^84^45^16^1$	$2^33^14^25^1$	30	H1	100.0	1,532	2,323	2,990	8,774
				H2	100.0	1,683	2,490	3,294	7,167
				H3	99.9	50	88	106	882
				H4	99.9	41	75	123	767
GCC	$2^{18}3^{10}$	$2^{37}3^3$	9	H1	100.0	5,665	7,317	8,366	29,026
				H2	100.0	7,104	8,539	10,482	21,288
				H3	100.0	224	268	351	1,564
				H4	100.0	125	221	318	1,409
Bugzilla	$2^{49}3^14^2$	$2^43^1$	16	H1	100.0	368	742	1,328	599
				H2	100.0	241	557	980	450
				H3	99.9	24	53	98	90
				H4	99.9	23	49	80	79
SPIN-S	$2^{13}4^5$	$2^{13}$	16	H1	100.0	296	461	570	276
				H2	100.0	139	342	521	115
				H3	99.9	52	81	132	131
				H4	99.9	25	55	90	41
SPIN-V	$2^{42}3^24^{11}$	$2^{47}3^2$	16	H1	100.0	5,785	6,415	7,195	4,864
				H2	100.0	3,893	4,903	5,994	3,571
				H3	100.0	641	795	1,023	928
				H4	99.9	496	763	1,058	991

## V. EXPERIMENT

We performed an experiment in order to evaluate the performance of the approach and to clarify the effects of the proposed methods. In this experiment, we consider the problem of testing two-way interactions ( $t = 2$ ). The experiment was performed on a PC with Windows 8 Professional 64 bits OS, Intel Core i7-3537U 2GHz CPU and 8Gbyte memory. The algorithm was implemented using the Java language. Our implementations use binary decision diagrams (BDDs) [10] to perform tests whether a value schema is valid or not. This could be done by using other constraint solvers. For BDD manipulations, we use JBDD libraries.<sup>1</sup>

We applied the proposed approach to five SUT models which represent well-known real-life applications. These applications and models are explained in [11] in details. The parameters of these models are explained in the Parameters column of Table III in the form  $l_1^{m_1}l_2^{m_2}\dots$  which means that the model has  $m_i$  parameters that have a domain of size  $l_i$ . The Constraints column shows the number of constraints and the number of parameters involved in each of the constraints. If the SUT model has  $z$  constraints that involves  $w$  parameters, then the fact is represented as  $w^z$ . For example, the SPIN-S model consists of 18 parameters, including 13 binary parameters and five parameters whose domain size is four, and 13 constraints that involve two parameters.

For each SUT model, we set  $n$ , the number of iterations of the main algorithm, to  $|M_1| * |M_2|$  where  $M_1$  and  $M_2$  are the two largest parameter domains (see Section II-A). By Theorem 2, this number guarantees that the test suite generated by the algorithm can cover any two-way interaction with at least probability 63%.

The previous section proposed an additional termination condition for recursion and described two different design options for splitting parameter set  $S$ . As a result, we have four different settings in using the proposed approach.

- H1: Uniformly random parameter set splitting.

- H2: Uniformly random parameter set splitting and the additional termination condition for recursion.
- H3: Strategic parameter set splitting.
- H4: Strategic parameter set splitting, and the additional termination condition for recursion.

Our interest is how many test cases are added to provide the probabilistic guarantee. We measure the size of the resulting test suite to evaluate the algorithm with the four different settings. We also measured the execution time of the algorithm and the ratio of the covered interactions to all valid interactions. We executed 100 runs for each given SUT model. The minimum, the maximum and average of test suite size over the 100 runs are shown in Table III.

The theoretical result (Theorem 2) guarantees that every interaction can be covered with 63%; but the experimental results show that the coverage actually achieved was much higher than this lower bound and reached nearly 100%. This can be accounted for the fact that the size of the resulting test suite constructed by the algorithm was larger than  $n$ .

One can see from Table III that significant reduction in test suite size was achieved with strategic parameter set splitting (the cases of H3 and H4). The reduction achieved by using the additional recursion termination condition was not as drastic as in the case of strategic parameter set splitting but still substantial (the cases of H2 and H4).

The effects of these methods vary for different problems. The problems of Apache, GCC, and Bugzilla have less severe constraints than the other problems and exhibited greater reduction in test suite size. Roughly speaking, using both methods (the case of H4) test suite size was reduced by 90% or more compared to the case of H1. The two methods also achieved significant reduction in test suite size for the remaining two problems, SPIN-S and SPIN-V; but the degree of reduction was less than the other problems. This result is counter-intuitive because severer constraints require adding more test cases and thus the two methods should have more tangible effects, though this was not the case in the experiment. We plan to further investigate how constraints affect the performance of the proposed approach in future.

<sup>1</sup><http://javaddlib.sourceforge.net/jdd/>

TABLE IV. PERFORMANCE OF CASA

Problem	Average Execution time (s)	Average Test suite size
Apache	150.97	31.4
GCC	551.39	19.0
Bugzilla	21.78	16.0
SPIN-S	16.49	19.4
SPIN-V	103.03	34.6

As the reduction in test suite size directly led to the reduction of execution time, execution time was also significantly decreased using strategic parameter set splitting, although this method required an increased execution time per test case.

For comparison purposes, we measured the execution time of a CIT test suite generation tool. We chose CASA [12]. CASA uses a simulated annealing-based algorithm and has proven to be very effective in minimizing CIT test suites [12]. Note that CIT tools guarantee to cover every valid interaction. We executed 10 runs for each given SUT model and measured the size of the resulting test suites and the execution time. Table IV shows the average size of test suites and the average execution time over the 10 runs.

Comparing Tables III and IV, one can see that the execution time of the proposed random testing algorithm is much smaller by some order of magnitude than that of CASA. Table IV also shows the size of the test suites obtained by CASA. The size is much smaller than the proposed random testing approach; but the ratio between the two approaches significantly varies for different problems.

These results seem to show that there is no clear winner. The proposed random testing approach is superior in execution time for test case generation and can achieve high interaction coverage. CIT tools guarantee perfect interaction coverage and is very effective in minimizing test suite size. Hence the best approach depends on testing scenarios. A thorough discussion on this topic can be found in [8].

## VI. THREATS TO VALIDITY

In the experiment we examined only five models of SUTs. Although these models were derived from real-world applications, it is unclear whether or not they can be regarded as representatives of typical real-world software systems. Hence the observations made from the experimental results may not necessarily hold in general.

Because of the nature of random testing, we executed 100 runs of the test suit generation process for each different setting in the experiment. This number of runs may not be sufficient to draw solid statistical conclusions. We plan to apply some statistical test on the results obtained.

## VII. CONCLUSION

This paper addressed the problem of applying random testing to the constrained interaction testing problem. The proposed approach compensates the loss in interaction coverage by generating new test cases if a randomly generated test case violates the given constraints. We devised an algorithm for generating these test cases with two methods that can be incorporated into the algorithm. The experimental results showed that using both methods can significantly reduce the number of test cases that are added. Possible directions of future work

include conducting more comprehensive experimental studies and improving the proposed algorithms.

## ACKNOWLEDGMENTS

We thank Mr. Shingo Tanak for his preliminary work. Some ideas in this paper are due to him. We also thank Mr. Daiki Shigeoka for providing the performance results of CASA.

## REFERENCES

- [1] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton, "The AETG system: An approach to testing based on combinatorial design," *IEEE Trans. on Software Engineering*, vol. 23, no. 7, pp. 437–444, 1997.
- [2] M. Grindal, J. Offutt, and S. F. Andler, "Combination testing strategies: A survey," *Software Testing, Verification and Reliability*, vol. 15, no. 3, pp. 167–199, Sep. 2005.
- [3] R. Mandl, "Orthogonal latin squares: An application of experiment design to compiler testing," *Communications of the ACM*, vol. 28, no. 10, pp. 1054–1058, 1985.
- [4] C. Nie and H. Leung, "A survey of combinatorial testing," *ACM Computing Surveys*, vol. 43, pp. 11:1–11:29, Feb. 2011. [Online]. Available: <http://doi.acm.org/10.1145/1883612.1883618>
- [5] K. Tatsumi, "Test case design support system," in *Proc. of International Conference on Quality Control (ICQC'87)*, 1987, pp. 615–620.
- [6] D. R. Kuhn, D. R. Wallace, and A. M. Gallo Jr., "Software fault interactions and implications for software testing," *IEEE Trans. on Software Engineering*, vol. 30, no. 6, pp. 418–421, 2004.
- [7] B. J. Garvin and M. B. Cohen, "Feature interaction faults revisited: An exploratory study," in *ISSRE*, 2011, pp. 90–99.
- [8] A. Arcuri and L. Briand, "Formal analysis of the probability of interaction fault detection using random testing," *IEEE Transactions on Software Engineering*, vol. 38, no. 5, pp. 1088–1099, 2012.
- [9] D. R. Kuhn, R. N. Kacker, and Y. Lei, "Automated combinatorial test methods: Beyond pairwise testing," *CrossTalk: Journal of Defense Software Engineering*, vol. 21, no. 6, pp. 22–26, Jun. 2008.
- [10] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Trans. Comput.*, vol. 35, no. 8, pp. 677–691, Aug. 1986. [Online]. Available: <http://dx.doi.org/10.1109/TC.1986.1676819>
- [11] M. B. Cohen, M. B. Dwyer, and J. Shi, "Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach," *IEEE Trans. on Software Engineering*, vol. 34, pp. 633–650, Sep. 2008. [Online]. Available: <http://portal.acm.org/citation.cfm?id=1439185.1439210>
- [12] B. Garvin, M. Cohen, and M. Dwyer, "Evaluating improvements to a meta-heuristic search for constrained interaction testing," *Empirical Software Engineering*, vol. 16, no. 1, pp. 61–102, 2011. [Online]. Available: <http://dx.doi.org/10.1007/s10664-010-9135-7>

# An Extensible Framework to Implement Test Oracles for *Non-Testable Programs*

Rafael A. P. Oliveira<sup>\*†</sup>, Atif M. Memon<sup>†</sup>, Victor N. Gil<sup>‡</sup>, Fátima L. S. Nunes<sup>‡</sup> and Márcio Delamaro<sup>\*</sup>

<sup>\*</sup> Dept. of Computer Systems, University of São Paulo – (ICMC/USP)  
São Carlos, SP, Brazil

Email: rpaes@icmc.usp.br delamaro@icmc.usp.br

<sup>†</sup> Department of Computer Science, University of Maryland – UMD  
College Park, MD, USA

Email: atif@cs.umd.edu

<sup>‡</sup> Dept. of Computer Systems, University of São Paulo – (EACH/USP)  
São Paulo, SP, Brazil

Email: victor.gil@usp.br fatima.nunes@usp.br

**Abstract**—Test oracles evaluate the execution of SUTs (*Systems Under Test*) supporting testers to decide about correct outputs and behaviors. “Non-testable” systems are cases in which the testers must spend extraordinary efforts to judge SUT’s outputs. Currently, some contemporary non-testable programs are represented by systems with complex outputs such as GUIs (*Graphical User Interface*), Web applications, and *Text-to-speech* (TTS) systems. Currently, there is a lack of knowledge associated with automated test oracles and SUTs with complex outputs. Extensible testing frameworks are necessary to provide the reuse of components and the sharing of knowledge in this field. This paper presents an extensible framework to support the development of test oracles for non-testable programs with complex outputs. In addition, we present an alternative to reuse software engineering components through plug-ins-based frameworks. The framework adapts CBIR (*Content-Based Image Retrieval*) concepts to enable testers to specify test oracles. The framework matches concepts of signal feature extraction, similarity functions, and object comparisons to obtain a Java program that compares two objects, responding how similar they are, according to a threshold. We performed proofs of concept using two empirical studies and the results showed our framework is useful to alleviate human-oracle efforts supporting human decisions. In addition, the plug-ins-based framework we present is a contribution toward a reusing of components on test oracles for systems with complex outputs.

**Keywords**—test oracle; software testing; knowledge engineering.

## I. INTRODUCTION

Test oracles are instruments to examine particular executions of an SUT (*Systems Under Test*), supporting software testing designers to decide about correct outputs [1]. In testing environments, test oracles must deal with SUT’s outputs or behaviors after a determined input. Test oracles may assume several forms: programs, functions, sets of data, tables of values or even the tester’s knowledge about the SUT [2]. Regarding complex testing scenarios, including strategies for test case generation and coverage reports, a test oracle is a requisite to insure the testing productiveness. Automated test oracles can increase the test productivity and decrease testing costs [3].

“Non-testable” programs are SUTs in which an oracle does not exist or the tester must expend some exceptional

amount of time to determine whether the current output is correct [4]. Despite being a well-known concept, non-testable programs and systems hard to test are still common. For instance, software systems whose outputs represent complex data for test automation. In this context, some examples of contemporary non-testable programs are: (1) systems with complex GUIs (*Graphical User Interface*) whose visual information is reflected in different screen resolution, sizes, and orientations; (2) *Text-To-Speech* (TTS) systems that convert written text in audio files; (3) some Web applications whose result quality depends on the browsers; (4) *Virtual Reality* (VR) environments, and others. Systems with complex outputs are becoming standard and the software industry is developing more attractive and exquisite applications. In addition, contemporary software has a vast array of technologies including the hardware platforms, *Operating Systems* (OS), and programming languages to support complex scenarios.

In the context of software testing, complex outputs make oracle verification difficult. Existing studies on test oracles for systems with complex outputs use manual (human-oracle), semi-automated or ad-hoc techniques [5]. Due to their complexity, test oracle analysis of complex output commonly requires sensory and perceptual aspects of a human being: (1) vision and (2) hearing. For instance, vision is required for checking orientations and visibility of components in GUIs, Web applications, and VR environments. Hearing is needed for checking naturalness, intonations, and pauses in TTS systems, in which audio files are the output. However, manual approaches are unproductive and inappropriate to support the large demand for the software. A vast array of technologies including mobile devices, hardware platforms, OS, and programming languages are in favor of the complex output’s systems.

To supply the test industry with supporting strategies to systematize and automate testing techniques for systems with complex outputs, contributions from *Software Engineering* (SE) and *Knowledge Engineering* (KE) are necessary. Flexible and reusable approaches may represent a way to alleviate efforts on testing those systems. Every advancement or increase in knowledge of testing systems with complex outputs have to be adequately reported to enable its reuse.

In this context, knowledge-based approaches enhance the effectiveness of components and procedures reuse. Regarding testing approaches for contemporary non-testable systems, the knowledge has to be shared in order to follow satisfactorily new technologies.

In this paper we present an extensible open-source framework to support the development of flexible test oracles for systems with complex outputs. O-FIm/CO (*Oracle For Images and Complex Outputs*)<sup>1</sup> adapts CBIR (*Content-Based Image Retrieval*) concepts to allow tester defining flexible and adaptable test oracles, as well as reusing plug-ins' implementations. To provide an environment of automated test oracles, the framework matches concepts of signal feature extraction, similarity functions, and object comparisons. CBIR is the application of computer vision and *Image Processing* (IP) techniques to help in organizing digital images using their visual content [6]. Regarding an image database, according to one or more extracted features, CBIR systems locate images that are similar to a query image. Through its reusable plug-ins, O-FIm/CO allows the generalization of these concepts for complex outputs such as GUI screenshots and audio files. The O-FIm/CO flexibility allows testers to obtain a Java program that compares two objects, responding if they are similar or not, according to a threshold.

Besides some reusable testing plug-ins by themselves, these paper's contributions rely on the step-by-step feasible strategy to implement flexible test oracles for systems hard to test. We evaluate our technique using two practical examples of real-world systems with complex outputs: one for graphical outputs (GUIs), and one for audio outputs (TTS). We developed test oracles that use features from outputs as sources of information for comparison between model outputs and current SUT's outputs. In this context, the goals of this paper are: (1) Presenting an extensible framework for testing systems with complex output, using reusable plug-ins; (2) Describing a feasible way to write flexible test oracles for complex output domains; and (3) Presenting two practical examples of the framework's practical effectiveness on evaluating complex outputs.

## II. O-FIm/CO

Besides technical and structural details of O-FIm/CO, this section presents overall concepts of the "oracle problem" and CBIR.

### A. The Oracle Problem

When it is impossible or too difficult to decide about the correctness of test outputs, the result is a scenario called the oracle problem [4], [7]. Further, the oracle problem is the absence of an accurate test oracle or cases in which it is too expensive to apply the oracle. The oracle problem may occur depending on the SUT. In such cases, the test oracle is not able to support the right decisions. Due to the oracle problem, the tester often classifies the design of the test oracle as a complex and cognitive activity rather than being a routine activity [1]. Approaches to alleviate the oracle problem, in general, must deal with two problems: "false positives" (when a test result incorrectly rejects a true null hypothesis) and "false negatives" (the failure to reject a false null hypothesis).

Among several contemporary examples of testing scenarios associated with the oracle problem, our scope in this paper are SUT with graphical (GUIs, processed images, maps, routes) or audio (voice synthesis) outputs. Regarding testing strategies for these systems, often the tester has to play the role of the oracle, making the test activity unproductive and error-prone. However, the complex output domains lead the tester to decide about the correctness of the SUT manually.

Figure 1 presents different testing scenarios relying on the complexity of the SUT's outputs. In the Figure 1b, the SUT represents a TTS system, in which the tester (human-oracle) has to listen to the result that represents an understandable speech representing a written text. Similarly, Figure 1c represents a testing work-flow for an SUT with a graphical output, where the tester has to check several graphical aspects. For instance, regarding GUIs, the visual aspects must work properly independently of platform, screen resolutions, screen orientation, color systems, monitor settings, and *Look and Feels* (L&Fs). In both cases (Fig. 1b and 1c), the complexity of the SUT's output are considered. On the other hand, Figure 1a presents a testing scenario where the SUT has trivial outputs and automated testing procedures can be implemented without extraordinary efforts.

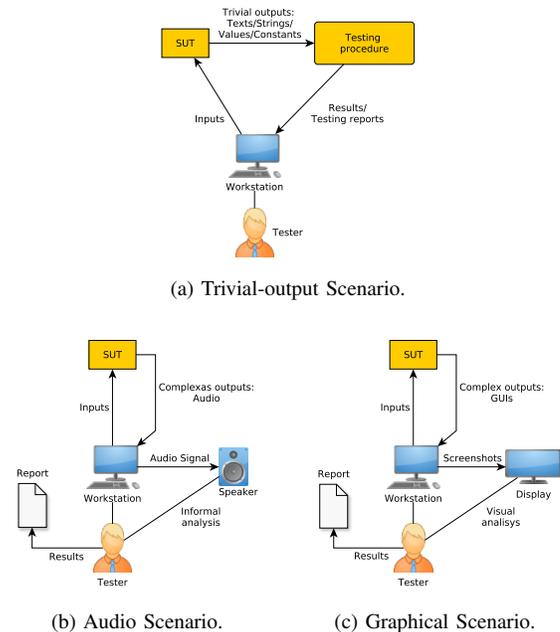


Fig. 1: Testing Scenarios Depending on the SUT's Output.

### B. Content-Based Image Retrieval

CBIR is any technology which helps organize digital image files using its visual content [6]. CBIR systems allow retrieving a finite set of images similar to a reference image. The similarity criteria are obtained by features extraction from the image related to shape, color, and texture. Feature extraction is the method used to obtain some information or particular data from the image. A set of extracted features composes a feature vector that will be considered in its retrieval. The feature vector is not sufficient to determine the result. It is

<sup>1</sup>see: <http://csl.icmc.usp.br/pt-br/projects/o-fim-oracle-images>

necessary to measure how similar two feature vectors are by using some sort of distance measure called a similarity function. Then, image comparisons are performed using values of features extracted and a similarity function. This process results in numeric values that represent the distance between the images.

### C. O-FIm/CO: Oracle For Images and Complex Outputs

O-FIm/CO enables testers to apply test oracle strategies in SUTs with complex outputs. O-FIm/CO extends concepts presented in previous works [8] regarding specific domains, CAD (*Computer-Aided Diagnosis*), and Web applications as SUTs with graphical outputs. In the scope of this paper, O-FIm/CO represents an alternative of testing verification for cases in which the complex outputs limit testers to applying traditional oracle strategies.

A precondition in using O-FIm/CO is the need for a reliable and representative output from which it is possible to establish a baseline for comparing objects under testing. For example, when it is desired to evaluate the appearance of a GUI in a particular L&F, it requires an environment from which it is possible to establish a standard image (screenshot) for comparison during the test. This standard screenshot may represent, for example, a GUI in a default screen resolution and orientation. On the other hand, regarding TTS systems, two empirical strategies are possible: (1) use two different systems and compare their outputs; and (2) compare the output produced by a TTS system with the audio of a person who reproduces the text input.

Figure 2 presents, in detail, the generic structure of O-FIm/CO. Regarding the structure presented, O-FIm/CO has an organization that matches different modules and resources in a common environment. These modules play fundamental rules to provide an oracle setting. These modules are: (1) “core”; (2) “parser”; (3) “wizard”; (4) “plug-ins”; (5) “API” (*Application Program Interface*); and (6) “oracle application”.

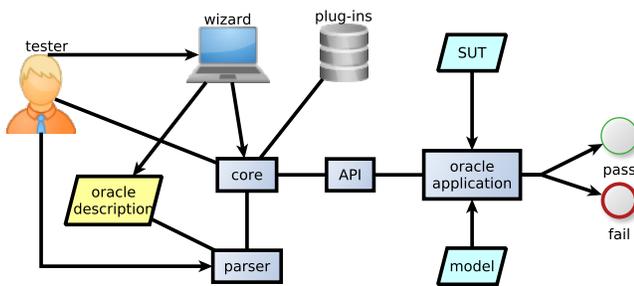


Fig. 2: O-FIm/CO General Structure.

“Plug-ins” may represent feature extractors for complex outputs or similarity functions. In the O-FIm/CO context, plug-ins are the main contribution from testers and they represent a reusable package of Java classes. In these plug-ins, testers must implement algorithms able to quantify features (feature extractors) or to figure out the difference between objects in accordance with their features (similarity functions). The framework is extensible due to its acceptance of plug-ins, providing the apportion of knowledge. In order to associate

these plug-ins and O-FIm/CO, testers must implement some specific Java interfaces. Then, the requirement is that when developing an extractor, the tester creates a Java class that implements one of these interfaces. There are different interfaces for audio outputs, graphical outputs, and similarity functions. After that, “testers” are able to install/uninstall plug-ins in O-FIm/CO in two different ways: (1) using line commands that are interpreted by the O-FIm/CO “core”; (2) using a friendly “wizard”. Exploring ClassLoaders, O-FIm/CO loads plug-in packages that are able to act over an SUT (audio or graphical output) extracting relevant information to compose test oracles.

Once the setup described above is made, testers may specify their test oracles using the O-FIm/CO “wizard”. To do so, feature extractors, extractor parameters, and similarity functions must be specified. The final specification is saved in a text file that represents an “oracle description”. Figure 3 presents an example of an oracle description including a similarity function, two extractors, and their parameters. During a test execution, a “parser” analyzes oracle descriptions and sets up the test. Finally, using an intuitive “API”, testers must write an “oracle application” that matches framework, oracle descriptions, SUT’s outputs, and models. The oracle application is a Java program that analyzes a set of complex SUT outputs, extracting features specified by testers, returning if they are similar to a model or not, according to a threshold (precision).

```

similarity Euclidean
extractor Color {rectangle = [100 100 30 40] sc = 1.33}
extractor ConnectedComponent {rectangle = [0 0 128 64]}
precision = 0.87

```

Fig. 3: Generic Instance of an Oracle Description.

## III. PROOFS OF CONCEPT

In this section we present empirical analysis whose goals are determining whether test oracles for SUT with complex outputs can benefit from our technique, enabling the reuse of components and the knowledge sharing among researchers. Then, the study is designed to answer the following *Research Questions* (RQ):

- RQ1: What is the feasibility of knowledge sharing through reusable plug-ins when automating test oracles for SUTs with different complex outputs?
- RQ2: Is this approach (adapting CBIR concepts) flexible enough to support oracle automation for different complex SUTs?

To answer the above questions, we have used real system applications for two critical complex scenarios: (1) graphical outputs and (2) audio outputs. Regarding graphical outputs we use O-FIm/CO to verify test results of a system with GUIs, exploring the framework resources on verifying acceptable behaviors of a TTS system.

### A. Complex Scenario 1: Complex GUIs

In the first scenario we use the O-FIm/CO approach to support the automated testing of GUI-based programs. This example of usage aims to evaluate the framework resources on alleviating manual verification of GUI visual requirements. These visual requirements include loading the SUT in different

L&Fs, different screen resolutions, and OSs. Then, our strategy implements an oracle application able to evaluate screenshots exposing GUIs from different aspects and to support decisions about their correctness regarding a model.

1) *Subject Application*: The subject application empirically evaluated is the text editor jEdit<sup>2</sup>. This application is released as free software with open source code. jEdit is a programmer's text editor written in Java and its main GUI has a huge panel for text editing, a tool-bar, and a side including a package explorer. Figure 4a presents the main jEdit GUI and its components. Figure 4b represents this same GUI under a different L&F. jEdit has a mean of 8000 downloads weekly and its development team is always fixing bugs reported by users.

2) *Experimental Setup*: In order to verify jEdit GUI appearance under critical situations, we have changed the default parameters of its appearance. Further, this proof of concept was designed to access the effectiveness of our technique in detecting possible failures related to visual aspects of GUIs derived from visual settings by users. Using a 20" monitor, we have exposed jEdit's main GUI to the following situations:

- Execution on six different screen resolutions (800x600, 1024x768, 1152x864, 1280x1024, 1366x768, and 1600x900);
- Execution on three different L&Fs (CDE-Motif, Metal, and Nimbus);;
- Execution on two different OS. (Linux and Microsoft Windows®); and
- Execution on two different GUI sizes (Maximized and Regular).

Regarding all possible combinations of visual aspects, we manually took 72 (36 for each OS) screenshots representing the visual information provided by the jEdit GUI. To perform this analysis, we have regarded the jEdit GUI with a resolution of 1366x768 and the Metal L&F as a model. Then depending on the OS, and the GUI size, different models were considered. We assured in advance that each model contained the appropriate requirements for the proper use of the application.

jEdit may have vision problems that can negatively affect its functions. Depending on the screen resolution, toolbar buttons can disappear and, then, become inaccessible for final users, so this constitutes a sort of visual bug. Indeed, this is a common problem for several GUI-based applications. Then, in this empirical evaluation we seek to automate test oracles for identifying this and similar situations. We have adopted a strategy that considers applying feature extractors in different points of jEdit's GUI. In Figure 4a we highlight the four main spots of interest in jEdit's GUI. Additional visual errors are supposed to be detected by this oracle: distorted components, and inoperable or invisible buttons. Then, based on comparisons with model features, our test oracles judge the test as *fail* when these conditions are identified.

To implement test oracles able to check the situations described above, we have developed five O-Flm/CO plug-ins: (1) Color count, (2) Sobel vertical, (3) Sobel horizontal, (4) Connected component, and (5) *Euclidean Distance* (ED). Along the framework website, all of these plug-ins (including scenario 2) are available as open source code software for possible experiments.

Color count is an extractor that, after pre-processing that converts the SUT screenshot into a gray scale image, counts the number of different gray levels in the region set by the parameter area. The number of different colors is normalized by 256 (number of levels). This extractor is useful in complex toolbars and informative panels to quantify their diversity of color.

Sobel vertical and Sobel horizontal are two edge detection algorithms for highlighting image patterns. After binarizing the GUI screenshot and applying the Sobel operator, this extractor counts the number of vertical or horizontal edge pixels. Then, the algorithm normalizes the number of pixels by the perimeter of the GUI. This plug-in quantifies visual patterns in some GUI components, such as combo boxes, toolbars, and information panels.

Connected component is able to count the number of different components in a determined region of the GUI. Using a gray scale image, this extractor identifies components by counting the number of similar pixels using an IP technique known as area-point transformation. This number is normalized by image width. This extractor can be used mainly to identify missing components after changing L&Fs and screen resolutions.

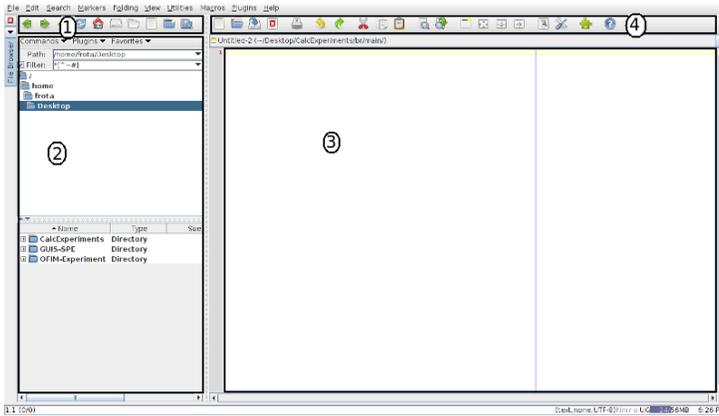
Euclidean Distance is a similarity function that corresponds to the function that measures the distance between two vectors of length  $n$ . The result 0.0 represents the maximum similarity.

After installing these plug-ins using the framework's wizard, we developed two different oracles. The first oracle (Oracle 1) rule is: to apply color count extractor in Rectangles 1, 3, and 4 of Figure 4a; Sobel vertical and Sobel horizontal are applied in Rectangle 2; Finally, the ED function checks the difference between the vectors (SUT versus Model). The second test oracle (Oracle 2) rule is: to apply the connect components extractor in Rectangles 1 and 4; in addition, the color count extractor acts in Rectangle 3; the ED function is used to obtain the final difference between the SUT and model.

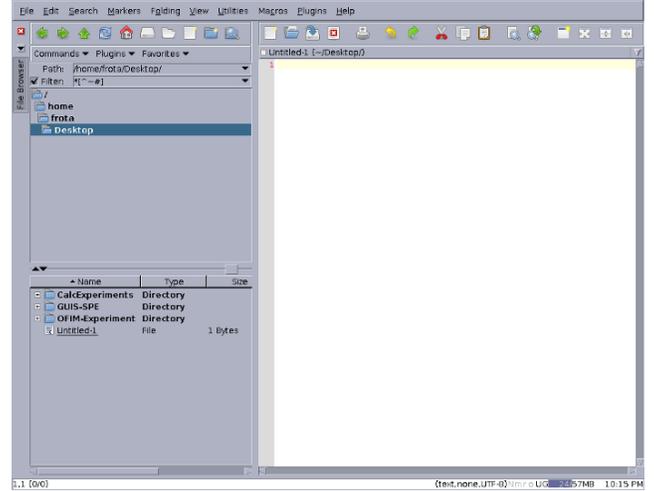
For both oracles, the threshold is set using the distance of the model and the screenshot of the SUT on a 1280x1024-Nimbus GUI. This threshold (precision) is due to a visual analysis that reveals most of the screenshots on resolutions 1280x1024, 1600x900, and 1366x768 always maintaining the necessary functional requirements to be considered as approved ("pass"). In the same line, Nimbus is the L&F that remains the minimal visual conditions of the GUI.

3) *Experimental Results*: After the test conduction, a manual detailed analysis of false (positive or negatives) results was carried out. Table I presents the false alarms noticed by our automated oracle regarding the whole proof of concept, including oracles, false positives, false negatives, and error rate. Each row represents one oracle in a specific OS under 36 different visual conditions. The error rate represents the sum of false positives and false negatives for each row. Broadly, for the Linux OS, considering the Maximized screenshots, the 1280x1024, 1366x768 e 1600x900 resolutions did not present any visual problem of loss of components or inactivity of buttons. However, the 800x600, 1024x760 e 1152x864 have presented some loss or distortion of components and therefore it was expected that the oracles fail their tests. For regular sizes GUIs, model images were considered in regular sizes and thus, it was expected that none of the screenshots presented problems. Analyzing of the screenshots of the Windows operating system, we note that the only difference to be considered in

<sup>2</sup>see: <http://www.jedit.org/>



(a) Default jEdit GUI



(b) jEdit GUI: Motif L&amp;F

Fig. 4: jEdit GUIs.

comparison to previous analyzes is that the Maximized size screenshots with 1152x864 did not present visual problems.

TABLE I: Error Summary for Complex Scenario 1.

# Oracle	OS	False positives	False Negatives	Error rate
1	Linux	3	0	8.3% (3/36)
1	Windows	1	2	8.3% (3/36)
2	Linux	2	0	5.5% (2/36)
2	Windows	1	0	2.8% (1/36)
<b>Total</b>		<b>7</b>	<b>2</b>	<b>6.25% (9/144)</b>

### B. Complex Scenario 2: TTS Systems

This second scenario aims to demonstrate how to explore the framework O-FIm/CO to develop automated test oracles for SUTs with complex outputs given in audio format. This scenario's scope is limited to TTS systems. Many embedded systems use TTS applications to read e-mails or social network updates, to read books or headlines for blind people, to read traveling directions, news, weather forecasts, and other written materials. However, the mainstream adoption of TTS is severely limited by its quality. Pronunciation and intonation problems make the speech synthesized highly unnatural. Despite their importance, TTS systems have their quality assessed by manual and unproductive processes. Informal human interpretations and different manual approaches are the most common procedures to evaluate the output of TTS systems [5]. In this empirical evaluation, using audio files in Wave format as the source of information for verifications, we automate test oracles to support testers' decisions about the correctness of TTS's outputs through comparison of two TTS systems.

1) *Subject Application*: Outputs from two real TTS systems are considered in this empirical analysis: (1) *TTS CPqD* (version 3.3)<sup>3</sup>; and (2) *Google®Translate Text-to-Speech*<sup>4</sup>. CPqD is the major Brazilian provider of telecommunications and solutions. TTS CPqD is a system able to synthesize texts written in Portuguese in speech signals near human speech. Then, Portuguese is the idiom considered in this evaluation.

<sup>3</sup>see: <http://www.cpqd.com.br/>

<sup>4</sup>see: <https://code.google.com/p/java-google-translate-text-to-speech/>

2) *Experimental Setup*: The aim of this analysis is to check two different TTS systems and compare their outputs. Typically, outputs from one TTS have to be regarded as model. TTS CPqD is a commercial application, then we have considered it as our model. Consequently, Google Translate TTS represents our SUT. In this context, in order to generate a data set, 100 Portuguese words were selected. These words were generated by the CPqD system loaded with three different texts from random popular news website. The choice of words did not follow any restriction, and it was obtained following the order of their occurrence. At the end of the process, 100 audio files were generated. Finally, the same words were used as input to our SUT.

We have implemented an automated test oracle exploring two test oracles: (1) Vowel extractor; and (2) Phoneme extractor. "Vowel extractor" is able to analyze an audio signal identifying the presence and the instant in which a Portuguese phoneme (A [a/, /e/], E [e/, /ε/], I [i/], O [o/, /o/], and U [u/]) is identified. Then, from a file representing a voice signal, the algorithm is able to return useful testing information. This information is specified through an XML (*eXtensible Markup Language*) file. "Phoneme Extractor" analyzes an audio signal, reporting the occurrences of three most common Portuguese phonemes (K [t/], T [t/], and M [m/]). Similar to the vowel extractor, testers are able to set parameters for the phoneme extractor using an XML file. Then, the extractor identifies the presence and the moment of specified phonemes in an audio file.

Both extractors went through a process of parameter calibration to work properly in the context of our TTS system using our data set including more than 100 words. After installing the plug-ins into the framework, we designed a test oracle using the extractors aforementioned to evaluate in order to assess the accuracy of the identification of the phonemes. This experience also aims to estimate the possibility of using oracle mechanisms for TTS systems test.

3) *Experimental Results*: We manually analyzed the results regarding the oracle acting in the set of data. This analysis aimed to measure the hits achieved in the identification of

phonemes of interest. Table II and Table III present a summary of errors and hits of the vowel extractor and the phoneme extractor, respectively. The tables reveal the hits regarding the number of occurrences of the feature extracted. Through a detailed evaluation of the results, we identify the differences between speed and voices of both systems as the main cause of errors.

TABLE II: Summary for the Vowel Feature Extractor.

Vowel		Number of occurrences					
		CPqD system			Google TTS		
		0	1	>1	0	1	>1
<b>A</b>	/a/, /e/	71%	85%	57%	60%	68%	64%
<b>E</b>	/e/, /ɛ/	65%	76%	60%	73%	61%	50%
<b>I</b>	/i/	89%	72%	50%	74%	59%	25%
<b>O</b>	/o/, /ɔ/	75%	69%	40%	61%	72%	40%
<b>U</b>	/u/	68%	68%	100%	93%	40%	100%
<b>hits</b>		<b>74%</b>	<b>74%</b>	<b>61%</b>	<b>72%</b>	<b>60%</b>	<b>56%</b>

TABLE III: Summary for the Phoneme Feature Extractor.

Phoneme		Number of occurrences			
		CPqD system		Google TTS	
		0	1	0	1
<b>K</b>	/k/	74%	53%	81%	63%
<b>T</b>	/t/	85%	48%	68%	52%
<b>M</b>	/m/	73%	47%	52%	20%
<b>hits</b>		<b>77%</b>	<b>49%</b>	<b>67%</b>	<b>45%</b>

#### IV. RESULT DISCUSSIONS

Regarding the research questions aforementioned (Sec. IV), our results contributed to clarify some points. We consider our approach to check SUT's outputs of distinct non-testable programs feasible. Some efforts from testers are necessary to assimilate the entire technique, however elementary concepts of programming and IP are enough to adopt our strategy and automate test oracles, as well as answer RQ1. Then, we consider O-FIm/CO an extensible framework, which enables testers to use pre-implemented plug-ins, as a potential form of knowledge sharing on automating test oracles for SUTs with complex outputs.

Regarding RQ2, as in most SE studies, we cannot be sure that the subject programs and the empirical analysis we adopted are representative to answer this question completely. Two scenarios are not enough to generalize our results for another complex scenario such as Web applications and RV environments. We need further analyses and more research focused on getting shared knowledge from other studies.

In addition to the pre-defined RQs, through the conduction of this study, we noticed that several research investments are necessary from testers. We consider these non-experimental findings as trade-offs on using O-FIm/CO. Among these trade-offs we highlight: (1) implementing or finding plug-ins; (2) seeking reliable output models; and (3) using the API to implement oracle applications. However, once the testers are done with investments 1 and 2, the technique is useful to alleviate oracle efforts on different contexts. In this context, all these investments address the usage of O-FIm/CO to a concept known as "amortization cost". Generally this concept refers to cases in which the testers spend an amount of time and effort to set a favorable scenario to be explored again and again.

Among the limitations associated with the wide usage of O-FIm/CO, we highlight that the correctness and precision of its

oracles actually depend on the quality of the oracle programs (threshold and plug-ins), which are manually developed by domain engineers. Then, testers must to select adequate plug-ins to achieve accurate results. In addition, the approach as whole needs an assumption – some reliable expected output must be given. In practice, testers and developers have to decide, based on system specifications, about reliable sources of information to be considered in testing. This decision evolves important features that have to remain apart of environments variations.

#### V. CONCLUSION

This paper proposes an extensible and reusable framework to support test oracle automation for non-testable programs, alleviating human-oracle efforts through reusable testing plug-ins and enabling the sharing of knowledge in this field. Based on two research questions, we conducted empirical analysis on two different complex scenarios: graphical and audio outputs. Our findings showed the feasibility of our approach and the possibility of reuse of plug-ins. In addition, we highlight that complementary experiments are necessary to consider this approach in wider scenarios such as, the usage of the framework to evaluate visual aspects of more than five GUI-based systems. A trade-off analysis reveals that tester has to dedicate extra effort in the very beginning of the testing project to achieve a favorable automated scenario. We believe extensible frameworks set possible solutions to the lack of knowledge associated with automated test oracles and SUT with complex outputs. The framework and all plug-ins developed and explored to collect our results are available as free software along with its documentation. This opens channels for technology transfer and reveals a solid contribution of the research reported in this paper.

#### ACKNOWLEDGMENT

FAPESP – Grant Numbers 2012/06474-1 and 2013/01775-6. Much of this work was conducted at the University of Maryland at College Park, MD, USA, during an one-year visit by the first author.

#### REFERENCES

- [1] W. Chan and T. Tse, "Oracles are hardly attain'd, and hardly understood: Confessions of software testing researchers," in *Proceedings of the 13<sup>th</sup> International Conference on Quality Software (QSIC 2013)*, Boston, USA, 2013, pp. 245–252.
- [2] P. Mateo and P. Usaola, "Bacterio oracle: An oracle suggester tool," in *Proceedings of the 25<sup>th</sup> International Conference on Software Engineering and Knowledge Engineering (SEKE 2013)*, Boston, USA, 2013, pp. 300–305.
- [3] M. Staats, G. Gay, and M. Heimdahl, "Automated oracle creation support, or: How I learned to stop worrying about fault propagation and love mutation testing," in *Proceedings of the 34<sup>th</sup> International Conference on Software Engineering (ICSE 2012)*, Zurich, Switzerland, 2012, pp. 870–880.
- [4] E. J. Weyuker, "On Testing Non-Testable Programs," *The Computer Journal*, vol. 25, no. 4, pp. 465–470, Nov. 1982.
- [5] P. Taylor, *Text-to-Speech Synthesis*, 1st ed. Cambridge University Press, 2009.
- [6] R. Datta, D. Joshi, J. Li, and J. Z. Wang, "Image Retrieval: Ideas, Influences, and Trends of the New Age," *ACM Computing Surveys*, vol. 40, no. 2, pp. 1–60, Apr. 2008.
- [7] M. D. Davis and E. J. Weyuker, "Pseudo-oracles for non-testable programs," in *Proceedings of the ACM '81 Conference*, ser. ACM '81. New York, NY, USA: ACM, 1981, pp. 254–257.
- [8] M. E. Delamaro, F. L. S. Nunes, and R. A. P. Oliveira, "Using concepts of content-based image retrieval to implement graphical testing oracles," *Software Testing, Verification and Reliability*, vol. 23, no. 3, pp. 171–198, 2013.

# Empirical Comparison of Intermediate Representations for Android Applications

Yauhen Leanidavich Arnatovich, Hee Beng Kuan Tan,  
Sun Ding, Kaping Liu  
INFINITUS, Infocomm Centre of Excellence,  
School of Electrical and Electronic Engineering,  
Nanyang Technological University,  
50 Nanyang Avenue, Singapore 639798  
{yauhen001, ibktan, ding0037, kpliu}@ntu.edu.sg

Lwin Khin Shar  
Interdisciplinary Centre for Security, Reliability and Trust,  
University of Luxembourg, Luxembourg  
lwinkhin.shar@uni.lu

**Abstract**—In Android-based mobile computing, since the original Java source code is irretrievable from Dalvik bytecode, intermediate representations (IRs) were developed to represent Dalvik bytecode in readable form. To date, SMALI, JASMIN, and JIMPLE are all used as Android application IRs by mobile developers, testers and researchers. Here, we compare these three IRs via randomized event-based testing (Monkey testing) to determine that which most accurately preserves the original program behaviors in terms of the number of successfully injected events. As such program behaviors are critical to mobile security, the choice of IR is crucial during software security testing. In our experiment, we developed an event-based comparative scheme, and conducted a comprehensive empirical study. Statistical comparison of the three IRs' program behaviors shows that SMALI behaves closest to the original applications and hence is the most suitable for software security testing as the most accurate alternative to the original Java source code (which is usually not publicly available).

**Keywords**—intermediate representation; program behaviors; event-based testing; Android computing; SMALI; JASMIN; JIMPLE

## I. INTRODUCTION

Android operating system usage is widespread, and many Android applications are being developed every day. All these applications are originally written in the Java programming language. To be executed on an Android-powered mobile device, Java source code must be compiled to *Java* bytecode, and then, transformed into *Dalvik* bytecode.

It is a fact that the *Dalvik* Virtual Machine (DVM) was especially developed for the Android mobile platform considering the limitations of mobile devices. In sharp contrast, the DVM is a register-based environment whereas the *Java* Virtual Machine (JVM) is stack-based. As the JVM and the DVM have different internals, there is no way to retrieve an original Java source code from *Dalvik* bytecode. Therefore, third-party tools can be used to obtain appropriate intermediate representations (IRs) such as SMALI [1], JASMIN [2], and JIMPLE [3] from *Dalvik* bytecode.

The latest Android research suggests that SMALI [4][5][6][7][8][9], JASMIN [10], and JIMPLE [11][12][13][14][15] can be used for Android software security testing purposes. But, the point at issue is that it is still unknown which IR most accurately preserves the original program behaviors. For example, carrying out software security testing

on the least accurate IR could mislead test results by reflecting the original program behaviors mistakenly, i.e. it could change the original program internals and consequently generate new, change or eliminate existing vulnerabilities of the original application. In sharp contrast, if we used the most accurate IR for software security testing, such an IR could mostly retain the number of existing security risks of the original application, and provide the most accurate testing results so that the IR would be the most suitable for testing purposes. Consequently, the most accurate IR could guarantee an ability to detect a similar number of security risks to that of an original Java source code. Therefore, it is necessary to investigate which IR is the most accurate and loses the least of the original program behaviors during disassembling.

Inspired by this interest, our motivation, in this paper, is to find which IR most accurately preserves the original program behaviors in terms of the number of successfully injected events. This research is intended to help developers, testers and researchers to most accurately detect mobile security risks of Android applications without having an original Java source code.

Our main contributions of this study are the following:

- We developed an event-based comparative scheme to find which IR most accurately preserves the original program behaviors in terms of the number of successfully injected events. We run 520 applications from Google Play on our scheme and find that SMALI behaves closest to the original applications.
- Using an automated event-injected testing approach and statistics, we compare program behaviors of SMALI, JASMIN, and JIMPLE to original ones. Our results represent the benefits of the use of SMALI against JASMIN and JIMPLE for software security testing.

These research results can guide mobile developers, testers and researchers to use an appropriate IR in order to most accurately carry out software security testing of Android applications without having an original Java source code.

The rest of the paper is organized as follows: Section II covers the Dalvik Virtual Machine and the Java Virtual Machine features, and shows the differences between them. Section III describes our experiment design in detail. Section IV represents our experimental results. Section V discusses related work. Section VI concludes our paper.

## II. FEATURES OF THE DALVIK VIRTUAL MACHINE AND THE DALVIK EXECUTABLE FILE FORMAT

In this section, we describe the special features of the DVM as a virtual machine especially developed for Android-based mobile computing, and stress the differences between the DVM and the JVM. Our goal, in this section, is to show that the *Java* bytecode structure has nothing similar to *Dalvik* bytecode, and consequently it would not be true to directly apply existing software security testing techniques to *Dalvik* bytecode.

### A. The Dalvik VM Design Overview

The *Dalvik* VM is not a *Java* VM. *Dalvik* is the name of the Virtual Machine in which Android applications are to run. The *Dalvik* VM is a register-based environment, and it runs classes compiled by a *Java* compiler *javac* that have been further transformed into a single class (*classes.dex*) by the Android SDK default *dx* tool. After applying *dx* tool, the *Dalvik Executable (.dex)* has only a distant relationship with *Java* bytecode (*.class*). The *Dalvik* VM executes the *Dalvik Executable (.dex)* file, which is optimized for minimal memory consumption.

As both applications and system services of the Android OS are developed in the *Java* language, the *Dalvik* VM has been written so that a device can run multiple VMs efficiently. Every Android application runs in its own process, with its own instance of the *Dalvik* VM, and it is referred to as *sandboxing* applications. When an Android-powered device is started, a single virtual machine process called *Zygote* is created, which preloads and pre-initializes core library classes. Once the *Zygote* has been initialized, it will reside and listen for socket requests coming from the runtime process, which indicates that it should generate new VM instances based on the *Zygote* VM instance. Thus, by spawning new VM processes from the *Zygote*, the startup time of another VM is highly minimized. All other *Java* programs or services are originated from this process, and run as their own process or threads in their own address space. Since every application runs in its own process within its own virtual machine, it takes advantage of not only efficient running of multiple VMs, but also fast creation of new VMs. The *Dalvik* VM relies on the Linux kernel for underlying functionality such as threading and low-level memory management.

The core library classes (*.libc*) that are shared across the *Dalvik* VM instances are commonly only read by applications. When the (*.libc*) classes are needed, the memory from the shared *Zygote* VM process is simply copied to the forked child process of the application's *Dalvik* VM. Such behavior allows it to maximize the total amount of shared memory while still restricting applications from overlapping with each other and providing security across application and sandboxing individual processes [16][17][18].

### B. The Dalvik Executable File Format (.dex)

The *Dalvik Executable (.dex)* format is designed to meet the requirements of systems that are constrained in terms of memory and processor speed. The *.dex* design is primarily driven by the sharing of data between running processes. The main difference between *Java* bytecode (*.class*) and *Dalvik* bytecode (*.dex*) is that all the classes of the Android application are packed into one file. This is not simply packing, all the classes in the same *Dalvik* bytecode (*.dex*) file share the same field, method, tables and

other resources. In the *Dalvik* VM, classes from the same *Dalvik Executable (.dex)* file are loaded by the same class loader instance.

The *.dex* file format uses mechanisms for conserving the RAM of mobile device running Android OS. A *Constant Pool* stores all literal constant values used within the class such as string constants used in code as well as field, variable, class, interface, and method names. Instead of storing these values in the class, they always can be found by their indices in the *Constant Pool*. In the case of the *.class* file, each class has its own *Constant Pool*. But, the *.dex* file contains many classes in one file named *classes.dex*. All of these classes share the same type-specific *Constant Pool* in order to avoid the duplication of constants in the *.dex* file [16][18].

## III. EXPERIMENT DESIGN

For our experiment, we use an Android-powered physical device running Android 4.1.2, *android-apktool* and *dex2jar* assembler/disassembler tools, *Soot* *Java* optimization framework, the *Dumb Monkey* test, and a *1-Sample Sign* test statistic. Our experiment was conducted on a Core i5-2400 @ 3.10GHz with 8GB of RAM machine running Windows 8.

### A. Experimental Design and Scheme

In our experiment, we identify which reassembled Android applications (SMALI, JASMIN, or JIMPLE) most accurately preserves the original program behaviors in terms of the number of successfully injected events. For that purpose, we downloaded 520 top free applications from Google Play (by 20 top free applications from 26 different categories). All downloaded applications are compatible with our testing Android-powered physical device. Since we have a data set of applications from Google Play, we consider them to be benign applications (not malware), and consequently not obfuscated or encrypted. Hence, we processed the 520 Android applications as follows: *disassemble*, *assemble*, *sign*, *align*, *install*, and run the *Dumb Monkey* test. Next, we reassembled applications with the aid of *android-apktool*, *dex2jar*, and *Soot* tools in order to obtain SMALI, JASMIN, and JIMPLE IRs, respectively. Also, we did not modify any IR code during reassembling. We applied the *Dumb Monkey* test to 520 Android applications. Every *Dumb Monkey* test generates a sequence of pseudo-random events based on a specified *seed* value. In our experiment, every *Dumb Monkey* test contains 30 trials for every category. Every trial contains the same set of 20 applications (from a particular category) assembled from SMALI, JASMIN, JIMPLE, and original applications. We use the constant *seed* value within a trial (to test the same set of 20 SMALI, JASMIN, JIMPLE, and original applications) to generate the same sequence of pseudo-random events, and be able to statistically compare the program behaviors of the IRs to the original ones in terms of the number of successfully injected events. Every trial has a unique *seed* value (unique sequence of pseudo-random events), which changes 30 times (since we have 30 trials) within a category.

The *Dumb Monkey* test has been chosen as our aim is to find which IR most accurately preserves the original program behaviors in terms of the number of successfully injected events. When the *Dumb Monkey* test has been done for every category, we apply a *non-parametric 1-Sample Sign* test statistic to be able

to conclude how close a particular IR reflects the behaviors of the original applications.

For our experiment, we designed and applied the following event-based comparative scheme:

- Disassemble Android applications by using *android-apktool*, *dex2jar*, and *Soot* tools. *Android-apktool*, *dex2jar*, and *Soot* disassemble Android package (.apk) into SMALI (*Dalvik Assembler*), JASMIN (*Java Assembler*), and JIMPLE code (*Java Simplified*), respectively (see Section III C).
- Assemble Android applications by using *android-apktool*, *dex2jar*, and *Soot* tools. All Android applications were assembled without any code modifications.
- Sign Android applications with private keys. This is a mandatory requirement for any Android application if a particular application should be installed whether on an Android emulator or an Android-powered physical device. We applied a standard Java package to run the signing process.
- Align Android packages by using default Android *zipalign* tool. The archive alignment tool provides important optimization of Android packages (.apk). This tool aligns the Android packages on 4-byte boundaries and reduces the amount of RAM consumed when running the application. Notwithstanding, this tool is not a mandatory requirement to install applications, but highly recommended by Android developers.
- Install applications on an Android-powered physical device. We use a testing physical device running Android 4.1.2.
- Apply the *Dumb Monkey* test to all installed applications (to every 20 top free applications from 26 different categories). The *Dumb Monkey* test meets our requirements, and is sufficient to reveal the difference in behaviors between the IRs and the original applications. We run the *Dumb Monkey* test with the same sequence of pseudo-random events for SMALI, JASMIN, JIMPLE, and original applications. After every *Dumb Monkey* test, we stop all running applications to return them to the initial state and repeat the *Monkey* test again.
- Apply *non-parametric* statistical test. To identify whether a particular application failed the *Dumb Monkey* test, we use a *1-Sample Sign* test, and consequently we are able to reveal the difference in behaviors between applications assembled from SMALI, JASMIN, JIMPLE, and the original applications. In particular, we statistically compare program behaviors of the IRs to the original ones.

### B. Android Experimental Environment

In our experiment, we use the Android Debug Bridge (*adb*), and an Android-powered physical device. The Android Debug Bridge is a versatile Command Line tool that permits communication with an Android emulator instance or an Android-powered physical device [19][20].

To install and run the *Dumb Monkey* test on the Android-powered physical device, we use the *cmd* tool. Specifications of the Android-powered physical device on which we conducted our experiment are indicated below in Table I.

We configured the Android-powered physical device as follows:

- We install applications on the Android-powered physical device with the aid of *adb* tool.
- At the first time, we manually run the applications to provide the required information (login/password, phone number, and other normal means of access), for example, applications with a sign-in screen (Skype, OneDrive (formerly SkyDrive), Facebook, Facebook Messenger, Spotify, and others), and applications with a required phone number (Viber, WhatsApp, WeChat, and others). However, we could not provide the information for two-step authentication (one-time password, verification code), which requires human interaction. Also, we keep all applications running and they can be seen as “running services” or “cached background services” on the Android-powered physical device to allow the *Monkey* test to carry out testing.
- We have a gallery of the most popular audio/video files (.mp3, .mp4, .avi), pictures (.jpeg), and office documents (.doc, .xls, .ppt, .pdf) for the applications in case they use the gallery during the testing.

The Android-powered physical device was connected to the Internet, Bluetooth was turned on, and the device was on a mobile carrier service during the experiment.

### C. Android Tools

In our experiment, we use some well-known existing reverse engineering tools for Android applications.

#### 1) *Android-apktool: Dalvik Disassembler*

To conduct the experiment on SMALI, we chose the disassembler/assembler named *android-apktool*. Android applications typically use resources provided by the Android OS. The *android-apktool* needs these framework files to decode and build Android applications correctly. To install the framework, we run the following *cmd* code:

1. *apktool if <framework\_name>*

The parameter *<framework\_name>* specifies the path to the framework *apk*-file. Next, two commands are used to disassemble and assemble Android packages, respectively:

1. *apktool d <apk\_name>*

2. *apktool b <directory\_with\_decompiled\_apk\_name>*

When an Android application is assembled, this tool automatically creates *build/* and *dist/* directories. The *dist/* directory contains the fully assembled Android package (.apk).

TABLE I. ANDROID-POWERED PHYSICAL DEVICE SPECIFICATIONS

Android version	4.1.2
RAM	2GB
Internal storage	32GB
External storage	Not available
Camera	Front/Main
Display	4.7 inch

## 2) Dex2jar: Java Disassembler

The *dex2jar* is another tool that provides another IR named JASMIN. To run the disassembling/assembling process, the following *cmd* code can be used:

1. *d2j-dex2jar -f-o <jar\_name1> <apk\_name>*
2. *d2j-jar2jasmin -f-o <directory1> <jar\_name1>*
3. *d2j-jasmin2jar -f-o <jar\_name2> <directory1>*
4. *d2j-jar2dex -f-o classes.dex <jar\_name2>*

For all 4 above-mentioned commands, the first parameter is an output and the second one is an input file or folder. The *classes.dex* file, which is assembled from JASMIN, should be directly replaced in the Android package (*.apk*).

## 3) Soot: Java Optimization Framework

*Soot* is a language manipulation and optimization framework consisting of intermediate languages for the Java programming language. The *Soot* tool requires Android APIs to disassemble/assemble Android packages correctly. *Soot* automatically checks the appropriate API level according to the application requirements and uses those APIs for disassembly and assembly. In our experiment, we used the *Soot* project downloaded from the *Soot* repository [21]. To run *Soot*, we use Eclipse IDE with specific arguments to *Soot*:

- src-prec, <[jimple|apk]>
- process-dir, <apk\_folder>
- android-jars, <android\_api\_jar\_folder>

The *classes.dex* file, which is assembled from JIMPLE, should be directly replaced in the Android package (*.apk*).

## D. The Dumb Monkey Test

To process the large number of Android applications, we use an automated *Monkey* test tool [22][23]. We applied the *Dumb Monkey* [24] bundled with the standard Android SDK. We created a batch script file to launch the *Dumb Monkey* test automatically. Our batch script contains the following command:

1. *adb shell monkey -p <package\_name> -v <pseudo\_event\_count>*

We specified two parameters *<package\_name>* with a particular Android (*.apk*) package. In our experiment, this parameter is used to identify a single package accessible to *Monkey* without any dependencies on other packages. The *<pseudo\_event\_count>* parameter is set to 2000 pseudo-random events.

## IV. STATISTICAL ANALYSIS RESULTS

We applied the event-based comparative scheme, discussed in Section III, to 520 Android applications to find which IR most accurately preserves the original program behaviors in terms of the number of successfully injected events. We applied a *non-parametric 1-Sample Sign* [25][26] test statistic to the *Dumb Monkey* test results to conclude if a particular application failed the *Monkey* test. From the *1-Sample Sign* test results, we count the number of applications failing the *Dumb Monkey* test for every category, and, based on the obtained numbers, graphically show the difference in program behaviors between SMALI, JASMIN, JIMPLE, and the original applications. We perform statistical analysis with the aid of MINITAB statistical software. To be able to apply the *1-Sample Sign* test, we have performed a normality test on the data (*Monkey* test results) and found that the data are non-normally distributed and non-

symmetric.

We run 30 trials for 20 Android applications from every category. After the *Dumb Monkey* test, the result contains 600 values (each value represents the number of successfully injected events) for a particular category. To compare the IRs to the original applications appropriately, the same sequence of pseudo-random events was applied to the applications assembled from SMALI, JASMIN, JIMPLE, and original applications.

We chose the following hypotheses testing criteria to verify whether the *Dumb Monkey* test was passed:

$$H_0 : \mu = \mu_0$$

$$H_1 : \mu < \mu_0$$

where  $H_0$  is null hypothesis and  $H_1$  is an alternative hypothesis. If  $H_0$  is true, a particular application passed the *Dumb Monkey* test. If  $H_0$  is not true, we accept an alternative hypothesis  $H_1$ . To carry out the *1-Sample Sign* test, we set  $\mu_0$  to 2000 pseudo-random events (the maximum number of pseudo-random events to be injected). To interpret the *1-Sample Sign* test results, we use a *P-value* approach with a 0.05 level of significance. After performing the *1-Sample Sign* test, we graphically show the final results to find the most accurate IR.

Figs. 1-3 clearly show the difference between SMALI, JASMIN, JIMPLE, and original applications, and help visually identify the IR, which most accurately preserves the original program behaviors. The dashed lines in Figs. 1-3 plot the number of applications which failed the *Dumb Monkey* test for SMALI, JIMPLE, and JASMIN applications, respectively, against the application categories on the horizontal *x-axis*. The dotted lines represent the number of the original applications which failed the *Dumb Monkey* test.

Comparing Figs. 1-3, it can be seen that the lines in Fig. 1 are the closest to each other, from which it is concluded that SMALI most accurately preserves the original program behaviors in terms of the number of successfully injected events. Thus, the applications, assembled from SMALI, behave closest to their respective original applications.

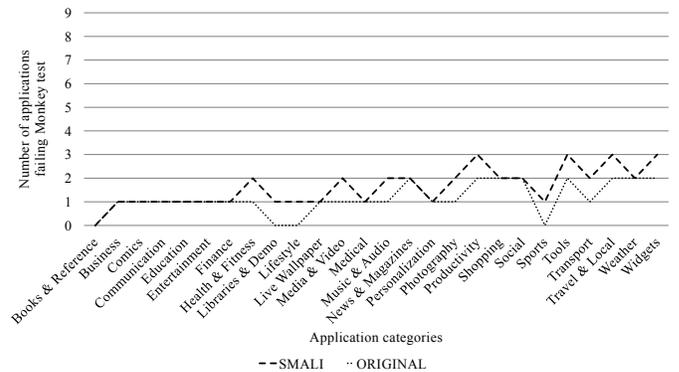


Figure 1. Number of SMALI vs. Original applications failing Monkey test.

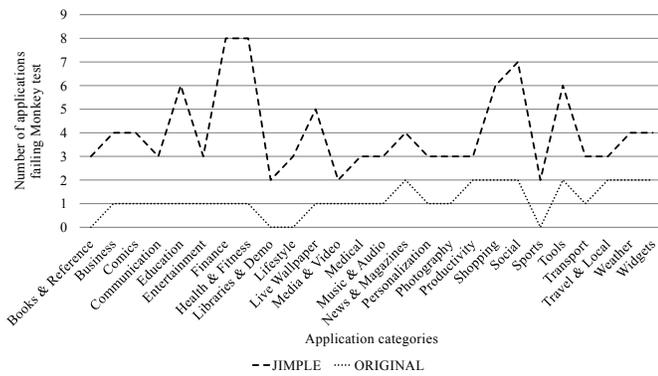


Figure 2. Number of JIMPLE vs. Original applications failing Monkey test.

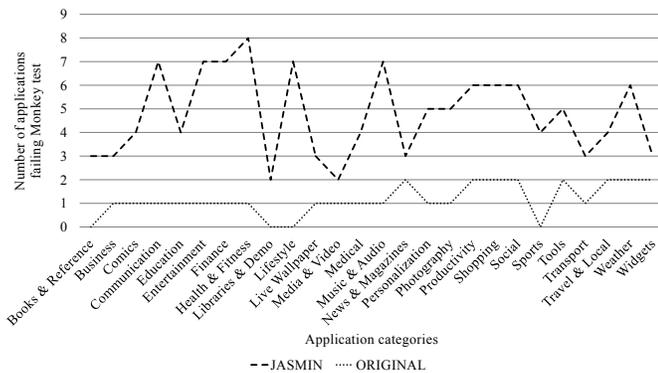


Figure 3. Number of JASMIN vs. Original applications failing Monkey test.

As JASMIN and JIMPLE both perform very similarly to each other and it is difficult to identify the difference in behaviors between them, we provide statistics in Table II to show the difference in accuracy of program behaviors between the IRs and the original applications.

## V. RELATED WORK

Previous studies were based on source code semantic analysis applying static or dynamic tools. The previous approaches were connected with extracting duplicated parts of the code, finding dependencies within the code written in different languages, identifying functionality from executable source which is poorly documented, or detecting merging and splitting of files and functions in procedural code.

Marcus and Maletic [27] propose an approach to scan the original code and try to identify the parts of the source code of similar high-level implementations. This approach uses an information retrieval technique to carry out a static analysis, and determine semantic similarities between source code documents and executable source code.

Ying et al. [28] propose a technique to predict source code changes by mining change history because it is difficult to find the entity dependency between source codes written in different languages. The new approach is to help developers identify relevant source code during a modification task. The proposed solution applies data mining techniques to determine change patterns from the change history of the code.

TABLE II. PROGRAM BEHAVIORS PRESERVATION

Intermediate Representation	Preserved Program Behaviors of Original Applications (%)
SMALI	97.69
JIMPLE	85.58
JASMIN	81.92

Eisenbarth et al. [29] present a semiautomatic technique that reconstructs the mapping for features triggered by the user and exhibits observable behaviors. This technique allows for the distinction between general and specific computational units with respect to a given set of features. This technique combines dynamic and static analyses.

Godfrey and Zou [30] propose an extended detection of merging and splitting of files and functions in procedural code. The improved approach shows how reasoning about how call relationships have changed and where merges and splits have occurred. Thus, it helps to recover some information about the context of the design change.

We have proposed another approach based on statistical comparison of the program behaviors. In sharp contrast to existing approaches, we are focusing on the program behaviors, but not any changes of source code. We identify the differences in the program behaviors between the IRs and the original applications based on the injection of the same sequences of pseudo-random events.

## VI. CONCLUSION

In this work, we examined three different IRs on 520 Android applications. We compared SMALI, JASMIN, and JIMPLE IRs to the original Android applications in terms of the number of successfully injected events via an event-based comparative scheme to determine the difference in their behaviors. We found that SMALI most accurately preserves and provides the closest reflection of the original program behaviors. We suggest that SMALI can be used by developers, testers and researchers as the most accurate alternative to an original Java source code for software security testing.

## ACKNOWLEDGMENT

Lwin Khin Shar was supported by the National Research Fund, Luxembourg (grant FNR/P10/03).

## REFERENCES

- [1] Android security related tools. <http://ashishb.net/security/android-security-related-tools/>, 2013. Accessed: 08-Mar-2014.
- [2] Soot: a Java Optimization Framework. <http://www.sable.mcgill.ca/soot/>, 2008. Accessed: 08-Mar-2014.
- [3] Analyzer and Disassembler for Java class files. <http://classfileanalyzer.javaseiten.de/>, 2008. Accessed: 08-Mar-2014.
- [4] L. Batyuk, M. Herpich, S. A. Camepe, K. Raddatz, A. Schmidt, S. Albayrak, "Using static analysis for automatic assessment and mitigation of unwanted and malicious activities within Android applications," in 6th Intern. Conf. MALWARE'11, pp. 66-72, IEEE, 2011.
- [5] C. Zheng, S. X. Zhu, S. F. Dai, G. F. Gu, X. R. Gong, X. H. Han, W. Zou, "SmartDroid: an automatic system for revealing UI-based trigger conditions in android applications," in Proc. SPSM'12, pp. 93-104, 2012.
- [6] Smali: An assembler/disassembler for Android's dex format. <https://code.google.com/p/smali/>, 2009. Accessed: 08-Mar-2014.
- [7] J. Hoffmann, M. Ussath, T. Holz, M. Spreitzenbarth, "Slicing droids: program slicing for smali code," in Proc. SAC'13, pp. 1844-1851, 2013.

- [8] R. Johnson, Z. H. Wang, C. Gagnon, A. Stavrou, "Analysis of Android Applications' Permissions," in 6th Intern. Conf. SERE-C'12, pp. 45-46, IEEE, 2012.
- [9] V. Rastogi, Y. Chen, X. Jiang, "DroidChameleon: evaluating Android anti-malware against transformation attacks," in Proc. CCS'13, pp. 329-334, 2013.
- [10] Jasmin – Java Assembler Interface. <http://jasmin.sourceforge.net/about.html>, 1996. Accessed: 08-Mar-2014.
- [11] A. Bartel, J. Klein, Y. Le Traon, M. Monperrus, "Dexpler: Converting Android Dalvik Bytecode to Jimple for Static Analysis with Soot," in Proc. SOAP'12, pp. 27-38, 2012.
- [12] R. Vallée-Rai, E. Gagnon, L. Hendren, P. Lam, P. Pominville, V. Sundaresan, "Optimizing Java bytecode using the Soot framework: Is it feasible?," in Compiler Construction, Springer Berlin Heidelberg, pp.18-34, 2000.
- [13] R. Vallée-Rai, P. Co, E. Gagnon, L. Hendren, P. Lam, V. Sundaresan, "Soot: a Java bytecode optimization framework," in CASCON'10, pp. 214-224, IBM Corp., 2010.
- [14] R. Vallée-Rai, L. J. Hendren, Jimple: Simplifying Java Bytecode for Analyses and Transformations. Sable Research Group, McGill University, 1998.
- [15] P. Parížek, O. Šerý, J. Kofroň, P. Jančík, Soot Framework. Faculty of Mathematics and Physics, Charles University in Prague, 2013.
- [16] D. Bornstein, Dalvik VM Internals. Google, 2008.
- [17] D. Ehringer, The Dalvik virtual machine architecture. Technical report, Google, 2010.
- [18] J. Huang, Understanding the Dalvik Virtual Machine. Developer, Oxlabs, 2012.
- [19] Android Debug Bridge, Android Developers. <http://developer.android.com/tools/help/adb.html>, 2009. Accessed: 08-Mar-2014.
- [20] M. Gargenta, Learning Android. O'Reilly Media, Inc., 2011.
- [21] Soot: A Java optimization framework. <https://github.com/Sable/soot>, 2014. Accessed: 08-Mar-2014.
- [22] S. Christey, The Infinite Monkey Protocol Suite (IMPS). RFC Editor, United States, 2000.
- [23] N. Nyman, Using Monkey Test Tools. Software Testing & Quality Engineering, 2000.
- [24] UI/Application Exerciser Monkey, Android Developers. <http://developer.android.com/tools/help/monkey.html>, 2009. Accessed: 08-Mar-2014.
- [25] W. J. Conover, Practical Nonparametric Statistic, 3rd ed., John Wiley & Sons, Inc., 1999.
- [26] D. J. Sheskin, Handbook of Parametric and Nonparametric Statistical Procedures, 3rd ed., Taylor & Francis, 2003.
- [27] A. Marcus, J. I. Maletic, "Identification of high-level concept clones in source code," in Proc. ASE'01, pp. 107-114, IEEE, 2001.
- [28] A. T. T. Ying, G. C. Murphy, R. Ng, M. C. Chu-Carroll, "Predicting source code changes by mining change history," IEEE Trans. Software Eng., vol. 30, no. 9, pp. 574-586, 2004.
- [29] T. Eisenbarth, R. Koschke, D. Simon, "Locating features in source code," IEEE Trans. Software Eng., vol. 29, no. 3, pp. 210-224, 2003.
- [30] M. W. Godfrey, L. Zou, "Using origin analysis to detect merging and splitting of source code entities," IEEE Trans. Software Eng., vol. 31, no. 2, pp. 166-181, 2005.

# Effectiveness of Automated Function Testing with Petri Nets: A Series of Controlled Experiments

Dianxiang Xu

Department of Computer Science  
Boise State University  
Boise, ID 83725, USA  
dianxiangxu@boisestate.edu

Ning Shen

Department of Computer Science  
Boise State University  
Boise, ID 83725, USA  
ningshen@u.boisestate.edu

**Abstract**— Existing work has developed techniques for automated generation of function tests from high-level Petri nets. Yet there is no empirical evidence that demonstrates the cost-effectiveness of this approach. This paper presents a series of controlled experiments to evaluate the fault detection capabilities of various strategies for test generation from high-level Petri nets. We built test models and generated executable test code for three subject programs. Then we executed the test code against more than 300 mutants of the subject programs. Each mutant is a variation of the original subject program with one injected fault. The experiment results show that (a) the reachability coverage-based tests are more effective than the tests for state coverage and for transition coverage, (b) postcondition-based test oracles are more effective than state-based oracles, (c) robustness tests with invalid inputs are critical to improving fault detection capability. In particular, the reachability coverage-based tests together with robustness tests killed 99.7% of the mutants.

**Keywords**— Software testing, model-based testing, Petri nets, fault injection, test coverage, test oracle

## I. INTRODUCTION

Model-based testing makes use of explicit models of a system under test (SUT) for generating test cases and verifying conformance between the SUT and the models [16]. It is an appealing method for function testing of software because of several benefits [14]. First, the modeling activity helps clarify test requirements, which are critical to effective testing. Second, generation of test cases can be automated or partially automated. Automation enables more test cycles and assures the required coverage of test models. Third, model-based testing can help improve fault detection capability due to the increased number and diversity of test cases.

Among the various modeling formalisms for model-based testing, UML state machines are the most popular. Empirical studies have shown that testing with state machines can detect many faults [1][14]. Nevertheless, the expressiveness of UML state machines is limited because all states need to be enumerated explicitly and it is difficult to specify and reason about test data. To address these issues, we have demonstrated that high-level Petri nets can be an expressive formalism for model-based testing because they can represent both control flows and data flows. Specifically, we have developed a tool,

MISTA (Model-based Integration and System Test Automation)<sup>1</sup>, for generating executable tests from function nets, which are lightweight high-level Petri nets [12][13]. It can generate tests to meet a given coverage criterion, such as state coverage, transition coverage, and reachability coverage. It also provides an expressive way for describing the relations between the model-level elements and the implementation-level constructs in the target language or test environment so as to automatically transform model-level tests into executable code. MISTA supports not only various programming and scripting languages (Java, C#, C, C++, HTML, Python, and VB), but also a number of test execution frameworks, such as xUnit for testing Java and C# programs, Robotium for testing Android mobile applications, Selenium IDE for testing web applications, and Robot Framework for keyword-based testing.

Our prior work has also applied MISTA to function testing of real-world systems [12]. However, cost-effectiveness of the automated testing remains an open issue. This paper presents a series of controlled experiments to evaluate the cost-effectiveness of various strategies for test generation from function nets. These strategies involve different types of test inputs (normal tests and robustness tests), different coverage criteria of test models (state coverage, transition coverage, and reachability coverage), and different methods for defining test oracles (states as oracles and postconditions as oracles). To the best of our knowledge, this is the first empirical study that evaluates the cost-effectiveness of automated model-based testing with high-level Petri nets.

The remainder of this paper is organized as follows. Section II gives an introduction to function nets and test generation. Section III describes the research questions and experiment setup. Section IV presents the results; Section V reviews related work. Section VI concludes this paper.

## II. FUNCTION NETS AND TEST GENERATION

### A. Function Nets

Function nets in MISTA are a lightweight version of high-level Petri nets: predicate/transition nets [3] and colored Petri

---

<sup>1</sup> MISTA 1.0, together with examples and links to Youtube demonstrations, can be downloaded at <http://cs.boisestate.edu/~dxu/research/MBT.html>.

nets [4]. It improves usability yet maintains the expressiveness of high-level Petri nets. A function net  $N$  is a tuple  $\langle P, T, F, I, L, \varphi, M_0 \rangle$ , where:

- (1)  $P$  is a set of places (i.e., predicates),  $T$  is a set of transitions,  $F$  is a set of normal arcs, and  $I$  is a set of inhibitor arcs.
- (2)  $L$  is a labeling function on arcs  $F \cup I$ .  $L(f)$  is a label for arc  $f$ . Each label is a tuple of constants and variables.
- (3)  $\varphi$  is a guard function on  $T$ .  $\varphi(t)$  is a guard condition for transition  $t$ .
- (4)  $M_0$  is a set of one or more initial markings.

In this paper, multiple initial markings (i.e., states) can be associated with the same net structure. Suppose  $M_0^k$  is an initial marking and  $M_0^k(p)$  is the set of tokens residing in place  $p$ . A token in  $p$  is a tuple of constants  $\langle X \rangle$ , denoted as  $p(X)$ . The zero-argument tuple is denoted as  $\langle \varphi \rangle$ . For token  $\langle \varphi \rangle$  in  $p$ , we simply denote it as  $p$ . We also associate a transition with a list of variables as formal parameters, if any.

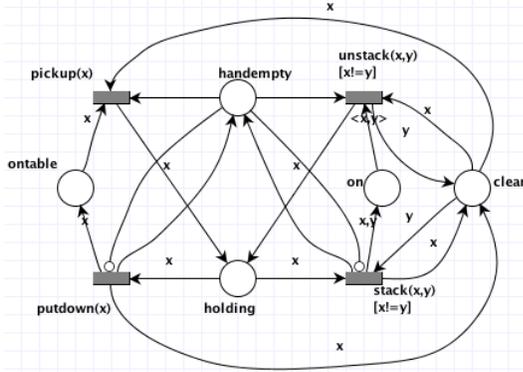


Figure 1. Function net for blocks.

Figure 1 shows an example, where *holding*, *clear*, *on*, *handempty*, and *ontable* are places (circles), *stack*, *unstack*, *pickup*, and *putdown* are transitions (rectangles). The guard condition of *stack(x,y)* is  $x!=y$  (enclosed in brackets). An arrow (e.g., from *holding* to *stack*) represents a normal arc; a line segment with a small solid diamond on both ends (e.g., between *stack* and *handempty*) represents an inhibitor arc. Each arc can be labeled by a tuple of variables and/or constants. If an arc is not labeled (i.e., the arc from *handempty* to *pickup*), the default label is the zero-argument tuple  $\langle \varphi \rangle$ .

Let  $p$  and  $t$  be a place and transition, respectively.  $p$  is called an input (or output) place of  $t$  if there is a normal arc from  $p$  to  $t$  (or from  $t$  to  $p$ ).  $p$  is called an inhibitor place if there is an inhibitor arc between  $p$  and  $t$ . Let  $x/V$  be a variable binding, meaning that variable  $x$  is bound to value  $V$ . A variable substitution is a set of variable bindings. For example,  $\{x/1, y/2\}$  is a substitution where  $x$  and  $y$  are bound to 1 and 2, respectively. Let  $\theta$  be a variable substitution and  $l$  be an arc label.  $l/\theta$  denotes the tuple (or token) obtained by substituting each variable in  $l$  for its bound value in  $\theta$ . For instance,  $l/\theta = \langle 1, 2 \rangle$  if  $l = \langle x, y \rangle$  and  $\theta = \{x/1, y/2\}$ .

Transition  $t$  is said to be enabled by  $\theta$  under a marking if:

- (1) each input place  $p$  of  $t$  has a token that matches  $l/\theta$ , where  $l$  is the label of the normal arc from  $p$  to  $t$ ;

- (2) each inhibitor place  $p$  of  $t$  has no token that matches  $l/\theta$ , where  $l$  is the label of the inhibitor arc between  $p$  and  $t$ ; and
- (3) the guard condition of  $t$  evaluates true according to  $\theta$ .

Suppose an initial marking for the function net in Figure 1 is  $\{holding(1), clear(2)\}$ . *stack* is enabled by  $\theta = \{x/1, y/2\}$  because token  $\langle 1 \rangle$  in the input place *holding* matches  $\langle x \rangle/\theta$ , token  $\langle 2 \rangle$  in the input place *clear* matches  $\langle y \rangle/\theta$ , there is no token in the inhibitor place *handempty* that matches  $\langle \varphi \rangle$ , and the guard condition  $x!=y$  is true according to  $\theta$ .

Enabled transitions can be fired. Firing  $t$  with  $\theta$  under  $M_0^k$  leads to a new marking  $M_1^k$  as follows:

- (1) For each input place  $p$  of  $t$  with input arc label  $l$ , the unified token  $l/\theta$  is removed from  $p$ , i.e.,  $M_1^k(p) = M_0^k(p) - \{p(l/\theta)\}$ .
- (2) For each output place  $p$  of  $t$  with output arc label  $l$ , a new token  $l/\theta$  is added to  $p$ , i.e.,  $M_1^k(p) = M_0^k(p) \cup \{p(l/\theta)\}$ .

Again, consider an initial marking  $\{holding(1), clear(2)\}$  in Figure 1. It enables *stack(x,y)* by  $\theta = \{x/1, y/2\}$ . Firing *stack* with  $\theta = \{x/1, y/2\}$ , i.e., *stack(1,2)*, removes token  $\langle 1 \rangle$  from *holding*, removes token  $\langle 2 \rangle$  from *clear*, add new token  $\langle 1, 2 \rangle$  to *on*, add new token  $\langle \varphi \rangle$  to *handempty*, and add new token  $\langle 1 \rangle$  to *clear*. Thus the firing leads to a new marking  $\{on(1,2), handempty, clear(1)\}$ .

We denote a sequence of transition firings as  $M_0^k [t_1\theta_1 > M_1^k \dots [t_{n-1}\theta_{n-1} > M_{n-1}^k [t_n\theta_n > M_n^k]$ , or simply  $t_1\theta_1 \dots t_n\theta_n$ , where  $M_0^k$  is an initial marking,  $\theta_i$  ( $1 \leq i \leq n$ ) is the substitution for firing transition  $t_i$  under  $M_{i-1}^k$ , and  $M_i^k$  is the resultant marking. A marking  $M_n^k$  is said to be reachable from  $M_0^k$  if there is such a firing sequence that transforms  $M_0^k$  to  $M_n^k$ .

## B. Test Generation

A test case consists of test input and test oracle. The test input of a (normal) test is a valid firing sequence  $t_1\theta_1 \dots t_{n-1}\theta_{n-1} t_n\theta_n$ . For each transition  $t_i(x_1, \dots, x_m)$  and substitution  $\theta_i = \{x_1/a_1, \dots, x_m/a_m\}$ ,  $t_i\theta_i$  ( $1 \leq i \leq n$ ), also denoted as  $t_i(a_1, \dots, a_m)$ , is a test input to the SUT. The input of a robustness test is an invalid firing sequence  $t_1\theta_1 \dots t_{n-1}\theta_{n-1} t_n\theta_n$ , where  $t_1\theta_1 \dots t_{n-1}\theta_{n-1}$  is a valid firing sequence but  $t_n\theta_n$  is not a valid firing. Robustness tests aim at exercising the SUT with invalid inputs so as to verify whether or not the SUT responds correctly. In principle, invalid events can be modeled in function nets by using additional transitions. However, representing various invalid events explicitly in function nets tends to make it difficult to manage test models. Thus, MISTA provides a technique for generating robustness tests automatically.

MISTA provides various methods for automated generation of test inputs from function nets. In this paper, we focus on coverage-based test generation, including the following:

- **State coverage:** A test suite for state coverage covers all markings reachable from the initial markings.
- **Transition coverage:** A test suite for transition coverage covers all transitions reachable from the initial markings.

- **Reachability coverage:** A test suite for reachability coverage covers all state-transitions reachable from the initial markings.
- **Reachability coverage with robustness tests.** A test suite for reachability coverage with robustness tests enhances the test suite for reachability coverage with a robustness test at each reachable marking, if exists.

Coverage-based test suites generated by MISTA are minimized in that: (a) MISTA terminates test generation once the coverage goal is achieved, and (b) MISTA never produces duplicate tests as it organizes all tests in a state transition tree that starts from the initial markings. The reachability coverage test suite subsumes both state coverage test suite and transition coverage test suite. It is subsumed by the suite for reachability coverage with robustness tests.

When generating a normal test input  $t_1\theta_1 \dots t_{n-1}\theta_{n-1}t_n\theta_n$ , MISTA allows the oracles to be created in the following ways:

- **States as test oracles:** The test oracles are the resultant markings (states) of respective transition firings,  $M_1^k \dots M_{n-1}^k M_n^k$ . For each place  $p \in P$  and each token  $\langle b_1, \dots, b_m \rangle \in M_i^k(P)$ ,  $p(b_1, \dots, b_m)$  is expected to evaluate true in the SUT. So MISTA converts  $p(b_1, \dots, b_m)$  to an assertion after test input  $t_i\theta_i$ .
- **Postconditions as test oracles:** The test oracles are the effects of respectively transition firings. As mentioned before, the effects of firing  $t_i\theta_i$  include: (a) removing tokens from its input places under  $M_{i-1}^k$ . Each of these tokens should not appear in the resultant state. So MISTA converts it to a negative assertion after test input  $t_i\theta_i$ . (b) adding new tokens to its output places. We convert each of the new tokens it to an assertion after test input  $t_i\theta_i$ .
- **States and postconditions as test oracles:** It is a combination of the above two methods. The test oracles include both resultant markings and postconditions of respective transition firings.

For the invalid firing  $t_n\theta_n$  of a robustness test  $t_1\theta_1 \dots t_{n-1}\theta_{n-1}t_n\theta_n$ , the test oracle is the marking before  $t_n\theta_n$ , meaning that  $t_n\theta_n$  should not change the system state. The generation of executable code for test inputs and test oracles is beyond the scope of this paper. The details can be found in [12][15].

### III. EXPERIMENT SETUP

Our experiments aim to answer the following questions:

- Q1: What are the fault detection capabilities of coverage-based test suites?*
- Q2: How do robustness tests improve fault detection capability?*
- Q3: How do state-based oracles and postcondition-based oracles contribute differently to fault detection?*

Our experiments are based on three Java programs: cruise control, vending machine, and blocks game. Cruise control and

vending machine are two popular examples for demonstrating model-based testing. In this paper, the cruise control code originated from [9], and the vending machine code was written by the authors. As shown in Figure 2, the cruise control model is essentially a low-level Petri net. It has no variables, similar to a traditional state machine. The vending machine model is shown in Figure 3. It sells three types of drinks: coffee, soda, and juice. The blocks program originates from the classical planning program in the field of artificial intelligence [10]. In MISTA, it is used as an example for demonstrating the expressive power of function nets. As shown in Figure 1, the blocks model captures both control-related and data-related interactions between four software components: *pickup*, *putdown*, *stack* and *unstack*. These interactions can be interpreted in first-order logic and the state transitions depend on various initial states. We believe that it is difficult to specify the behaviors with proposition-level UML state machines.

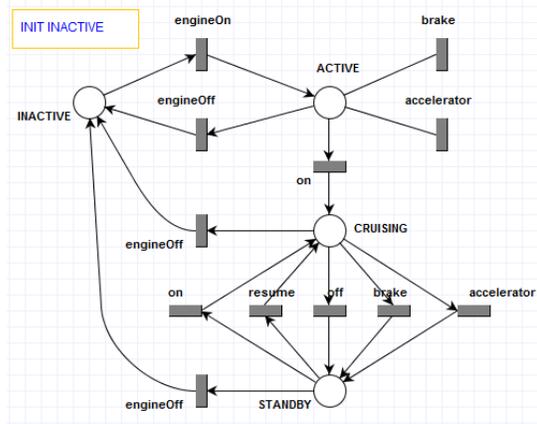


Figure 2. Function net for cruise control

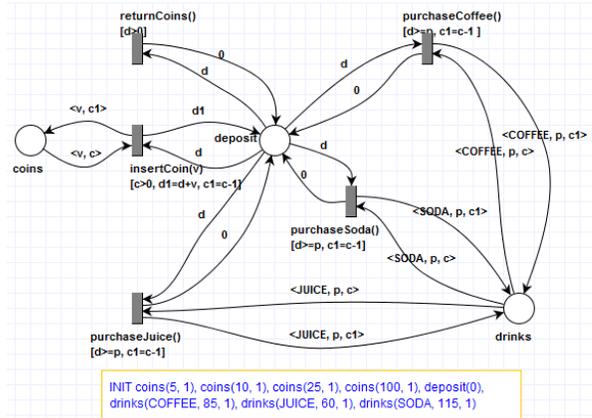


Figure 3. Function net for vending machine

TABLE I. METRICS OF SUBJECT PROGRAMS

Metric	Subjects	Cruise Control	Vending Machine	Blocks Game
Lines of code		854	430	166
Number of classes		12	5	2
Number of classes for mutation		2	3	2
Mutants generated by MuJava		122	102	55
Mutants created manually		0	11	34
Total mutants		122	113	89

Table I presents the main metrics of the subject programs. Although they are very small in terms of lines of code, they allow us to build systematic test models for the thorough evaluation of testing effectiveness. Our prior work has used MISTA to test large-scale real world applications [12], but focused on system components, rather than entire systems.

For each subject, we conducted the experiment as follows:

- (1) Build the function net (test model) and MIM (model-implementation mapping for test code generation);
- (2) Generate test code for each testing strategy under evaluation (i.e., each combination of coverage choices and oracle choices as described in Section II.B);
- (3) Execute all the test code against the subject program. If there is a failure, then either the subject program or the test code has a problem. In this case, fix the problem in the subject program or repeat steps (1) and (2);
- (4) Create mutants of the subject program using MuJava [8]. MuJava is a tool for generating Java mutants, executing test code against the mutants, and reporting execution results. Each mutant is a modified version of the subject program with one fault injected. We also created additional mutants by manually injecting the types of faults not covered by MuJava. For example, faults that do not achieve the desired postconditions may be due to missing statements. MuJava does not generate such mutants because removing an arbitrary statement may lead to a syntax error. Table I shows the number of mutants created for each subject program.
- (5) Execute all the test code in (2) against each mutant using MuJava. A mutant is said to be killed by a test suite if the execution of the test suite reports a failure.

#### IV. EXPERIMENT RESULTS

##### A. Coverage-based Tests

We use mutant-killing ratio (i.e., number of killed mutants divided by the total number of mutants) as an indicator of fault detection capability. Figure 4, Figure 5, and Figure 6 show the number of killed mutants and the mutant-killing ratio of each coverage-based test suite in cruise control, vending machine, and blocks. “State”, “Transition”, “Reachability”, and “Reachability+robustness” refer to test suites for state coverage, transition coverage, reachability coverage, and reachability coverage with robustness tests, respectively.

In cruise control and vending machine, the three test oracle methods happen to be identical. Thus, there is only one test suite for each of the coverage criteria in Figure 4 and Figure 5. The overall mutant-killing ratios for all subject programs in Figure 7 indicate that the test suites of state coverage are less capable than the test suites of transition coverage, which are less capable than the test suites of reachability coverage. The state coverage test suite and the transition coverage test suite are both a subset of the reachability coverage test suite. The test suites of reachability coverage with robustness tests are the most powerful – they killed 99.7% of all mutants. They killed all mutants of cruise control and vending machine. The only

live mutant of blocks has an endless loop created by MuJava. In this case, MuJava terminated the execution without failure because it used a timeout.

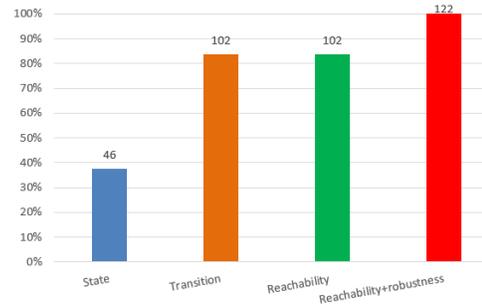


Figure 4. Effectiveness of coverage-based tests for cruise control

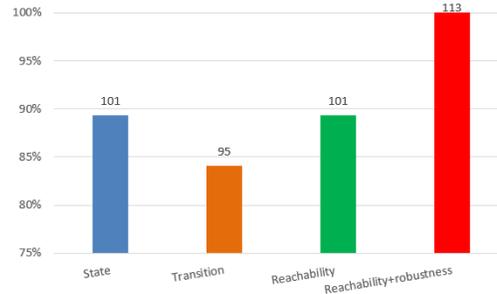


Figure 5. Effectiveness of coverage tests for vending machine

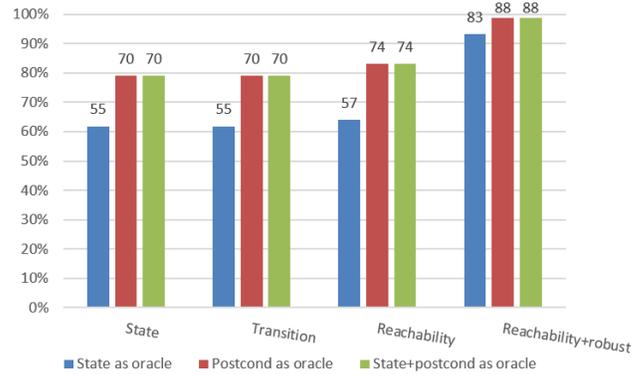


Figure 6. Effectiveness of coverage-based tests for blocks

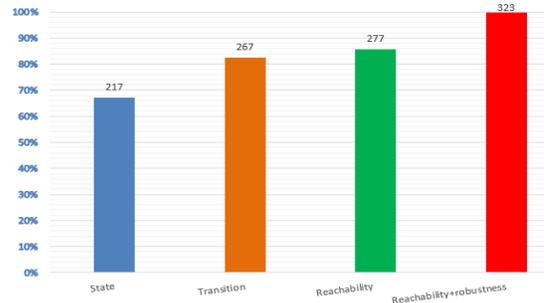


Figure 7. Overall effectiveness of coverage-based tests

Note that the comparison of test suites for state coverage and transition coverage varies from application to application. In cruise control, transition coverage is more capable (Figure 4). In vending machine, however, state coverage is more capable (Figure 5). They are the same in blocks (Figure 6).

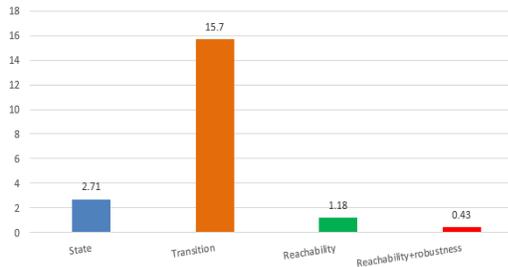


Figure 8. Number of mutants killed per test

Figure 8 shows the average number of mutants killed by each test in coverage-based test suites. Although the test suites for state coverage and transition coverage are less capable than those for reachability coverage and reachability coverage with robustness tests, their individual tests are more effective. In particular, each test in the transition coverage suite killed 15.6 mutants. This has two implications: (1) when testing resource (e.g., time and budget) is very limited, transition and state coverage are better choices. These tests should be conducted first. (2) When the testing process progresses, it gets more and more expensive to find additional faults because more and more tests need to be created and executed.

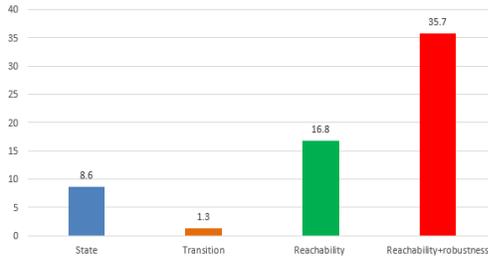


Figure 9. Lines of test code / mutants killed

Figure 9 shows the average lines of test code for killing mutants. The executable test code is generated automatically. Before test code generation, helper code (e.g., package statement) needs to be provided. In cruise control and blocks, the helper code has only two lines (i.e., package and import statements). In vending machine, the helper code has 16 lines (package and import statements and a method for verifying actual results). These numbers are insignificant compared to the total lines of test code. Table II below shows the lines of test code for reachability coverage and reachability coverage with robustness tests. Figure 9 shows the transition coverage tests used 1.3 lines to kill one mutant, whereas the reachability coverage with robustness tests used 35.7 lines to kill one mutant. Fig. 9 demonstrates the same phenomenon as Fig. 8.

### B. Robustness Tests

In MISTA, robustness tests are generated only for reachability coverage. So we evaluate their effectiveness by comparing the test suites for reachability coverage and for reachability coverage with robustness tests, as shown in Table II. *Reachability+R* denotes reachability coverage with robustness tests and LOC refers to lines of test code. Generally there are more robustness tests than normal tests (in terms of both number of tests and lines of code). The robustness tests contributed to the killing of 37 additional mutants – the test suites for reachability coverage killed none of these mutants.

TABLE II. EVALUATION OF ROBUSTNESS TESTS

Category		Cruise Control	Vending Machine	Blocks Game	Total
Tests	Reachability	9	204	21	234
	Reachability+R	25	648	72	745
	Increased	16	444	51	511
LOC	Reachability	100	4,022	520	4,642
	Reachability+R	217	10,076	1,230	11,523
	Increased	117	6,054	710	6,881
Mutants	Reachability	102	101	83	286
	Reachability+R	122	113	88	323
	Increased	20	12	5	37

Figure 10 shows the consequences of using robustness tests – the total number of tests increased by more than 200%, the total lines of code increased by 150%, and the mutant-killing ratio increased by only 13%. This indicates that, when the testing process progresses, much more tests would be required in order to find additional faults. While robustness tests are critical to fault detection, they are also costly because there are often numerous invalid inputs.

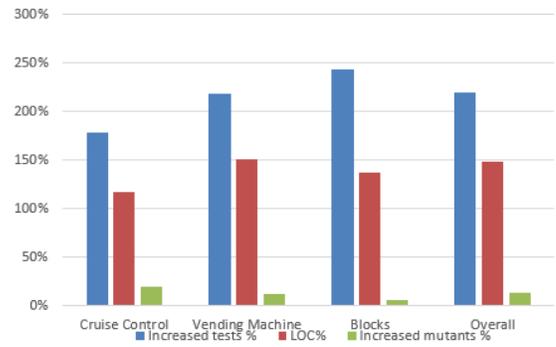


Figure 10. Cost effectiveness of robustness tests

### C. Test Oracles

In cruise control and vending machine, the three test oracle methods have no difference. In blocks, using the combination of states and postconditions as oracles is the same as using postconditions as oracles (refer to Figure 6) because all mutants killed by the state oracles are also killed by the postcondition oracles. Figure 11 shows the number of killed mutants and the percentage of killed mutants increased by using the postcondition oracles for each of the coverage-based test suites in blocks. The postcondition oracles are more capable than the state oracles because the state oracles do not check negative effects of test actions (i.e., token removal).

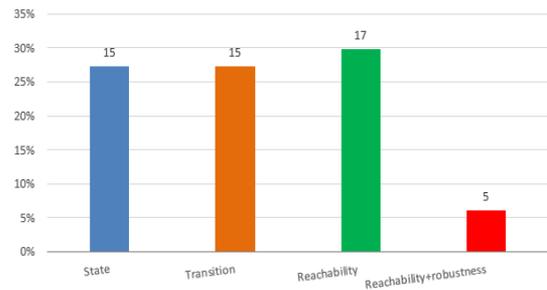


Figure 11. Effectiveness of postcondition oracles

#### D. Threats to Validity

The above experiment results have demonstrated that function testing with Petri nets can be highly effective in fault detection. The main threats to validity are discussed below.

First, the three subjects are very small programs in terms of lines of source code. The benefit is that we can build systematic test models and generate comprehensive tests suites. For complex applications, however, it can be difficult to build systematic test models with function nets. In addition, the methods for generating coverage-based tests may not scale up.

Second, the test oracles in the subject programs can all be represented formally and thus converted into executable assertions. For real-world applications, it is often a challenge to define precise test oracles.

Third, the three subjects are all Java programs. Although Java is a representative object-oriented language, the fault detection capability of a test generation method can depend on particular programming constructs and paradigms.

Finally, the evaluation of fault detection capability is based on fault injection, which is a common approach to the evaluation of testing effectiveness [5]. The mutants were either generated by MuJava or created manually. Although they have covered many types of bugs, they do not necessarily represent all possible faults in real-world software.

#### V. RELATED WORK

Zhu and He [17] have proposed a methodology for testing high-level Petri nets. The methodology consists of four testing strategies: transition-oriented testing, state-oriented testing, data flow-oriented testing, and specification-oriented testing. Each strategy is defined in terms of an adequacy criterion for selecting test cases and an observation scheme for observing a system's dynamic behavior during test execution. It is not concerned with how tests can be generated to meet the adequacy criteria. Lucio et al. [7] proposed a semi-automatic approach to test case generation from CO-OPN specifications. CO-OPN is a formal object-oriented specification language based on abstract data types and Petri nets. This approach transforms a CO-OPN specification into a Prolog program for test generation purposes. Lucio developed SATEL [6], a language for expressing test intentions of object-oriented CO-OPN specifications of reactive systems. Test intentions can be used to produce a reasonable or practicable number of test cases, including negative tests (i.e., robustness tests in this paper). Desel et al. [1] have proposed a technique to generate test inputs (initial states) for the simulation of high-level Petri nets. The above work targets the testing and simulation of Petri nets. Wang et al. [11] have proposed class Petri net machines for specifying inter-method interactions within a class and generating method sequences for unit testing. Manual work is needed to make the sequences executable.

#### VI. CONCLUSIONS

We have presented three controlled experiments for evaluating the cost-effectiveness of model-based testing with

high-level Petri nets. To the best of our knowledge, this paper is the first empirical study of its kind. The experiment results show that testing with high-level Petri nets can be highly effective in fault detection and that robustness tests and postcondition oracles are both critical to finding faults.

Applying model-based testing to large-scale real-world software may result in very complex test models which yield a large number of test cases. It may be infeasible to generate test suite for reachability coverage with robustness tests, although it is more capable than state and transition coverage test suites. Our future work aims to evaluate the scalability of various test generation strategies in MISTA.

#### ACKNOWLEDGMENT

This work was supported in part by US NSF under grants CNS 1123220 and CNS 1359590.

#### REFERENCES

- [1] Briand, L.C., Di Penta, M., Labiche, Y. "Assessing and improving state-based class testing: A series of experiments", IEEE Trans. on Software Engineering, vol. 30, no. 11, pp. 770-793, Nov. 2004.
- [2] Desel, J., Oberweis, A., Zimmer, T., Zimmermann, G. "Validation of information system models: Petri nets and test case generation," Proc. of the IEEE International Conference on Systems, Man, and Cybernetics (SMC'97), pp. 3401-3406.
- [3] Genrich, H.J. "Predicate/transition nets," Petri Nets: Central Models and Their Properties, pp. 207-247, 1987.
- [4] Jensen, K. Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use, vol. 26: Springer-Verlag, 1992.
- [5] Jia, Y. and Harman, M. "An analysis and survey of the development of mutation testing." IEEE TSE, Vol. 37, No. 5, pp. 649-678, 2010.
- [6] Lucio, L. "SATEL — A test intention language for object-oriented specifications of reactive systems," Ph.D. Dissertation, UNIVERSITÉ DE GENÈVE, Centre Universitaire d'Informatique. 2009.
- [7] Lucio, L., Pedro, L., and Buchs, D. "Semi-automatic test case generation from CO-OPN specifications," Proc. of the Workshop on Model-Based Testing and Object-Oriented Systems, pp. 19-26, 2006.
- [8] Ma, Y.S., Offutt, J., and Kwon, Y.R. MuJava: An automated class mutation system, Journal of Software Testing, Verification and Reliability, 15(2):97-133, June 2005. <http://cs.gmu.edu/~offutt/mujava/>
- [9] Magee, J. and Kramer, J. Concurrency: State Models and Java Programs. 2nd Edition, John Wiley & Sons Ltd, 2006.
- [10] Russell, S. and Norvig, P. Artificial Intelligence: A Modern Approach, Prentice Hall, 1995.
- [11] Wang, C. C., Pai, W.C., Chiang, D.-J. "Using Petri net model approach to object-oriented class testing," Proc. of the IEEE International Conf. on Systems, Man, and Cybernetics (SMC'99), pp. 824-828, Oct. 1999.
- [12] Xu, D. "A tool for automated test code generation from high-level Petri nets," Petri Nets 2011, LNCS 6709, pp. 308-317, Newcastle, June 2011.
- [13] Xu, D., Tu, M., Sanford, M., Thomas, L., Woodraska, D. and Xu, W. "Automated security test generation with formal threat models," IEEE TDSC. Vol. 9, No.4, July/August 2012, pp. 525-539.
- [14] Xu, D., Xu, W., and Wong, W.E. "Automated test code generation from class state models," IJSEKE, 19(4): 599-623, June 2009.
- [15] Xu, D., Xu, W., Kent, M., Thomas, L., Wang, L. "An automated test generation tool for software quality Assurance," IEEE Trans. on Reliability, Under review after minor revision.
- [16] Zander, J., Schiefewrdecker, I., and Mosterman, P. J. (eds.). Model-Based Testing for Embedded Systems, CRC Press, 2011.
- [17] Zhu, H. and He, X. "A methodology for testing high-level Petri nets," Information and Software Technology, vol.44, 473-489, 2002.

# Automatic XACML Requests Generation for Testing Access Control Policies

Yongchao Li, You Li, Linzhang Wang

State Key Laboratory of Novel Software Technology

Department of Computer Science and Technology

Nanjing University, Nanjing, 210046, China

Email: {lyc, leo86}@seg.nju.edu.cn, lzwang@nju.edu.cn

Guanling Chen

Department of Computer Science

University of Massachusetts Lowell

1 University Avenue, Lowell, Massachusetts, 01854, USA

Email: glchen@cs.uml.edu

**Abstract**—XACML has become increasingly popular for specifying access control policies in mission critical domains to protect sensitive resources. However, manually crafted XACML policies may contain errors which can only be identified with manual policies review. Recent progress in policy testing still requires tedious and inefficient manual efforts to compose access requests. In this paper, we propose an automatic XACML requests generation for testing access control policies by employing symbolic execution techniques. Firstly, the access control policy under test is converted into semantically equivalent C Code Representation (CCR). Secondly, the CCR is symbolically executed to generate test inputs. Finally, the test inputs are used to compose access control requests, which can be automatically evaluated with existing tools. We also implemented a prototype tool called XPTester (Xacml Policy Tester) and conducted extensive experiments upon real-world policies to demonstrate the scalability, efficiency and effectiveness.

**Keywords**—Access control policy; XACML; test generation; symbolic execution

## I. INTRODUCTION

Strong access control is necessary to protect the sensitive resources for security and privacy, and XACML [1] has become popular for access control specification and enforcement for protecting sensitive resources. XACML is also increasingly being adopted in mobile systems, especially in mission critical domains [2], [3], [4], [5], [6], [7]. The effectiveness of an access control system in protecting sensitive resources, however, relies on the correctness of access control policies. On the other hand, manually created access control policies, e.g., written in XACML, may be faulty and is prone to severe security consequences. According to the Internet Security Threat Report of 2013 by Symantec, the volume of web-based attack in 2012 has increased almost one-third of that in 2011<sup>1</sup>. An early study by University of Pennsylvania ranks *Broken Access Control* in the second place of top 10 web application security vulnerabilities.<sup>2</sup> Faulty access control configuration or specification that imports errors into access control policies is one of the most common vulnerabilities that lead to web-based attacks.

Therefore, XACML policies should be tested to ensure that they were specified as expected before they are implemented.

For testing an XACML policy, the testers need to send access requests to the policy and observe the access decisions. Every unexpected decision indicates that errors exist in the XACML policy. Test requests that cover the policy sufficiently are more effective to discover errors in the policy. However, it is tedious and inefficient to compose XACML requests manually. Therefore, it is attractive to apply automated testing approaches to test the XACML policies. There are several research work on testing XACML policies that are proposed recently [8], [9], [10], [11]. However, they are either not scalable or not effective enough. For example, Cirg [10] is based on change-impact analysis, which is a high time-consuming process and its efficiency is related to the size of the policy. So Cirg suffers from scalability issues and its application to real-world policies is limited. X-CREATE simply takes all possible combinations of the attribute values in the policy to generate test requests. This will lead to a lot of redundant test requests thus reducing the test efficiency.

In this paper, we propose an automatic, scalable and effective approach that employs symbolic execution techniques to generate access requests to test XACML access control policies. Because symbolic execution is able to generate high coverage tests for programs, we consider leveraging it to compose adequate test requests for access control policies. Firstly, the access control policy under test is converted into semantically equivalent C Code Representation (CCR). Secondly, the CCR is symbolically executed to generate test inputs. Finally, the test inputs are used to compose access control requests for testing. The results of the experiments on real-world XACML policies demonstrated that our approach is efficient and effective in aiding the testers to detect faults in the policies.

The contributions of this paper are as follows:

- We propose an automatic access request generation approach for testing access control policies by leveraging existing symbolic execution technique.
- We bridge non-executable XACML and executable code by using a code generation approach based on code schemas.
- We have implemented a supporting prototype tool, XPTester, and conducted controlled experiments using real-world access control policies.

The rest of the paper is organized as follows. Section II

<sup>1</sup>[http://www.symantec.com/content/en/us/enterprise/other\\_resources/b-istr-main\\_report\\_v18\\_2012\\_21291018.en-us.pdf](http://www.symantec.com/content/en/us/enterprise/other_resources/b-istr-main_report_v18_2012_21291018.en-us.pdf)

<sup>2</sup>[http://www.upenn.edu/computing/security/swat/SWAT\\_Top\\_Ten.php](http://www.upenn.edu/computing/security/swat/SWAT_Top_Ten.php)

introduces the background and provides a motivation example. Section III details the proposed approach. In Section IV we present the evaluation of our approach. In Section V we discuss the most related work, and we conclude and discuss the future work in Section VI.

## II. BACKGROUND

### A. XACML

XACML is an attribute based access control system and was proposed by OASIS<sup>3</sup> in 2003. In XACML, attributes are associated with four elements, that is, Subject, Action, Resource and Environment. An XACML policy is composed of three elements: PolicySet, Policy and Rule. A PolicySet may contain a sequence of Policies or PolicySets and a Policy consists of a sequence of Rules. Each of the three elements consists of at most one Target, which specifies the constraints on Subject, Action, Resource and Environment to restrict the type of requests to which the PolicySet, Policy or Rule can be applied. The authorization decision to the access requests is defined and stored in every Rule as Effect, the possible value of which is Permit or Deny. In systems that adopt XACML, a component named PEP (Policy Enforcement Point) composes access requests in XACML format and sends them to another component called PDP (Policy Decision Point) for requests evaluation. The PDP makes authorization decision based on the XACML policy. Sometimes there will be more than one Policies or Rules can be applied to an access request, but only one authorization decision is allowed for every single request. Then in XACML, Combining Algorithm is designed to handle such a situation. There are Policy Combining Algorithm and Rule Combining Algorithm dealing with conflict of Policies and Rules, respectively. In this paper, we deal with the most common four of the twelve types of Combining Algorithm: First-Applicable, Permit-Overrides, Deny-Overrides and Only-One-Applicable.

### B. Symbolic Execution

The concept of Symbolic Execution[12] was first put forward by JC King in 1976. It is proposed to facilitate the automated generation of high code coverage test cases. In symbolic execution, instead of supplying normal inputs to the program, the inputs are made symbolic to represent any values. Symbolic execution process is same as normal execution except that the values obtained during execution may be formulas on symbolic values. During symbolic execution, the executing path will fork at condition statement like if-else. Both branches will be executed and the corresponding constraints will be added to the current path constraint. At the end of an execution path, the path constraint on symbolic variables will be solved for concrete values. Thus a concrete input that can make this path be executed is obtained. After all paths are executed and the corresponding path constraints are solved (if solvable), a test suite is obtained, containing test inputs that is capable of covering the executed paths during symbolic execution. There are several tools supporting symbolic execution, i.e., KLEE [13]. KLEE is able to generate tests that achieve high coverage on a diverse set of complex and environmentally-intensive programs, specifically for C

language. For efficiency purpose, we chose KLEE in our framework for test case generation.

### C. A Motivating Example

Here we use a motivating example concerning mobile security to further demonstrate the importance of testing the access control policies applied to the mobile devices. In recent years, in order to improve working efficiency of the employees as well as cutting costs, more and more companies allow their employees to bring their personal devices into the workplace and use them to do daily work. This practice is called Bring Your Own Device (BYOD) [14]. Thus the employees can use their personal devices to access the company's privileged applications and information networks. At the same time the devices may also be connected to the Internet or have installed many other applications that may include malwares. On one hand, strong access control policies must be applied to protect the private resources of the company, i.e., to specify who can access what resources. On the other hand, before implementing the access control policy, testers should test the potentially complex policy thoroughly to prevent faulty policy causing severe security consequences (e.g confidential information leakage). This motivates us to investigate effective and efficient approaches to automatically test XACML policies.

## III. APPROACH

In this section, we describe the proposed approach in details. The framework of our approach is illustrated in Figure 1. There are three steps in the framework for a complete testing process, each of which is represented by a rectangle in the figure. Firstly, the XACML policy under test is converted into semantically equivalent C Code Representation (CCR). Secondly, test inputs of the CCR are generated via symbolic execution and afterwards are used to compose XACML test requests. Finally, the test requests are evaluated against the XACML policy under test to generate a testing report.

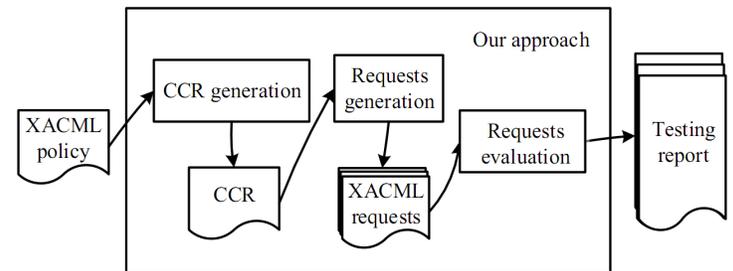


Fig. 1: Framework of our approach

### A. Generate C Code Representation

1) *Attribute Numeralization*: Symbolic execution is more capable of dealing with variables of integer type. However, the attribute values in XACML policies are in various types including string, URL, boolean and so on. Therefore, in order to take advantage of symbolic execution, XACML policy needs to be preprocessed by numeralizing the attributes appeared in the policy. Each attribute is assigned with a unique integer value to represent the attribute value [15]. We put each pair

<sup>3</sup><https://www.oasis-open.org/>

of mapping between an attribute value and its corresponding integer value into an AIM (Attribute Integer Mapping) set. One example of AIM is listed in TABLE I.

TABLE I: Example of AIM sets

Attribute value	Attribute type	Integer value
Technician	string	1
Manager	string	2
View	string	3

2) *Code Schema*: Real-world XACML policies may be complex and large, which makes it challenging to effectively converting them to code representations. The core of CCR generation is the *Code Schema*, which is shown in Figure 2. It is designed to facilitate systematic and scalable CCR generation from XACML policies. The Code Schema specifies the structure of the CCR for a XACML policy. Precisely, PolicySets, Policies and Rules are represented as functions in CCR according to the Code Schema. A CCR is composed of a sequence of the function definitions. The functions typically contain an *if* statement representing the evaluation of Target. Code in the *if* statement is called *Evaluation code block*, which may contain calls to other functions for XACML elements. Based on the Code Schema, we are able to systematically generate CCR from XACML policies in any size or complexity.

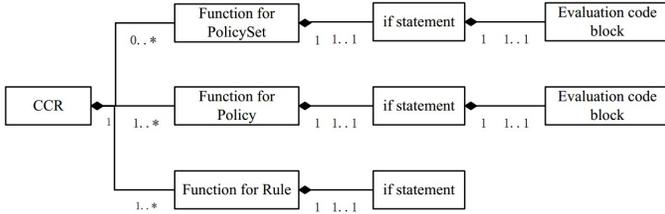


Fig. 2: Code Schema of C Code Representation (CCR)

3) *Mapping Rules from Combining Algorithms to Code*: While it is easy to represent the policy with *if-else* statements, the greatest challenge to convert an XACML policy into CCR is the complex semantics of Combining Algorithms. There are four Combining Algorithms defined in XACML: *First-Applicable*, *Deny-Overrides*, *Permit-Overrides* and *Only-One-Applicable*. The different semantics of Combining Algorithms are reflected in CCR by the distinctions between Evaluation code blocks. Table II shows the mapping rules from Combining Algorithms to specific Evaluation code blocks. When converting the XACML policies, the specific Evaluation code block will be selected from the mapping rules according to the type of Combining Algorithm to compose the CCR.

4) *Algorithm for CCR Generation*: The CCR generation algorithm, which is shown in Algorithm 1, converts an XACML policy into its CCR. The algorithm traverses the XACML policy in DFS (Depth First Search) manner starting from the root (PolicySet or Policy). **BuildPolicySet**, **BuildPolicy** and **BuildRule** are functions that convert PolicySet, Policy and Rule into corresponding functions. In **BuildPolicySet** or **BuildPolicy**, it checks the type of Combining Algorithm

TABLE II: Mapping rules from *Combining Algorithm* to *Evaluation code blocks*

Combining Algorithm	Evaluation code block
First-Applicable	if((effect = Policy_x(Sub, Act, Res, Env)) != -1 return effect;
Deny-Overrides	if((effect = Policy_x(Sub, Act, Res, Env)) != -1) if(effect == 1)return 1;
Permit-Overrides	if((effect = Policy_x(Sub, Act, Res, Env)) != -1) if(effect == 0)return 0;
Only-One-Applicable	if((effect = Policy_x(Sub, Act, Res, Env)) != -1) applicable = 1; if((effect = Policy_y(Sub, Act, Res, Env)) != -1){ if(applicable == 1) return -1; applicable = 1; } if(applicable == 0)return -1; return effect;

defined in the PolicySet (Policy). Based on the type of Combining Algorithm detected, the algorithm chooses the proper Evaluation code block by referring to the mapping rules from XACML policy to Evaluation code blocks. It then fills the Evaluation code blocks with the integer values for the attribute values found in Target. Subsequently, the algorithm recursively converts the PolicySets, Policies or Rules contained in the PolicySet or Policy until arriving at the end of the policy.

### B. Generate Test Requests via Symbolic Execution

For test generation, automation, adequacy, non-redundancy, and efficiency are mutually constraint factors. Symbolic execution techniques satisfy all above mentioned factors simultaneously. In our approach, we use a state-of-the-art symbolic execution tool KLEE [13] to explore every possible path of CCR to generate test inputs to cover adequate paths in an efficient manner. In the generated CCR, there are four integer variables (*Sub*, *Act*, *Res*, *Env*) defined at the start of the main function, representing Subject, Action, Resource and Environment, respectively. They should be made symbolic by using *klee\_make\_symbolic*. The CCR will be compiled into executable program using LLVM [16], and symbolically executed by KLEE to obtain test inputs for the CCR.

Symbolic execution generates a set of test cases containing inputs for CCR. Each test case contains values of four CCR inputs, which are Sub, Act, Res and Env. For each of these values, our framework queries the AIM sets for the corresponding attribute values. Once the attribute values of Subject, Action, Resource and Environment are obtained, an XACML request can be constructed.

### C. Evaluate the Test Requests

Our framework composes a set of test requests based on the test inputs generated from CCR, each of which is an access request. We use PDP to evaluate each test request to produce an authorization decision. We instrumented the PDP to record the evaluation trace (i.e., sequence of policy elements that are applicable to the request) into a testing report. Each decision will be compared with what is specified in the access control requirement specification (supplied by the tester). If the decision is inconsistent with the corresponding expected

---

**Algorithm 1** Converting a XACML *policy* into CCR

---

**Require:**

- *XP*, the XACML policy to convert.
- *AS*, the AIM sets

**Ensure:**The CCR for the XACML *policy*.

```
1: CCR ccr = NULL;
2: Function func = NULL;
3: for each PolicySet PS in XP do
4:   func = BuildPolicySet(PS, ccr);
5:   ccr.addFunction(func);
6: end for
7: for each Policy P in XP do
8:   func = BuildPolicy(P, ccr);
9:   ccr.addFunction(func);
10: end for
11: ccr.buildMainFunction();
12: return ccr;
13: function BUILDPOLICYSET(PolicySet PS, CCR ccr)
14:   query into AS to find integer values for the attributes;
15:   create if statement according to the Target of PS;
16:   CombiningAlgorithm CA
17:   = PS.CombiningAlgorithm;
18:   create Evaluation code block according to CA;
19:   for each PolicySet ps in PS do
20:     Function func = BuildPolicySet(ps, ccr);
21:     ccr.addFunction(func);
22:   end for
23:   for each Policy p in PS do
24:     Function func = BuildPolicy(p, ccr);
25:     ccr.addFunction(func);
26:   end for
27:   return function for PS;
28: end function
29: function BUILDPOLICY(Policy P, CCR ccr)
30:   query into AS to find integer values for the attributes;
31:   create if statement according to the Target of P;
32:   CombiningAlgorithm CA
33:   = P.CombiningAlgorithm;
34:   create Evaluation code block according to CA;
35:   for each Rule r in P do
36:     Function func = BuildRule(r);
37:     ccr.addFunction(func);
38:   end for
39:   return function for P;
40: end function
41: function BUILDRULE(Rule R)
42:   query into AS to find integer values for the attributes;
43:   create if statement according to the Target of R;
44:   return function for P;
45: end function
```

---

decision, there must exist a fault in the XACML policy. The request will be highlighted as a failed test in the generated report. By observing the evaluation trace of the test request, it is easy for the tester to locate the fault in the policy.

#### IV. IMPLEMENTATION AND EVALUATION

Based on the proposed approach, we implemented a prototype tool, XPTester, using Java version 1.7 under Linux (Ubuntu 12.10) with LLVM (version 2.9) and llvm-gcc (we used binary version for LLVM 2.9) [16], with KLEE [13] as the symbolic execution engine. In this section, we present the results of empirical experiments and give a comprehensive evaluation analysis.

Generally, there are two *effectiveness* measurements in evaluating an access control testing approach: the policy coverage ratios and the defect detection capabilities of the generated requests. There are also two other important aspects of *efficiency* that are often overlooked, number of test requests and time spent in generating them. The goal of this evaluation is to evaluate both the efficiency and effectiveness of our approach.

TABLE III: Experiment subjects

Policy Under Test	LOC	# CA	# PolicySet	# Policy	# Rule
continue-a	7138	2	111	266	298
SyntheticPolicy-1	12185	3	5	40	400
SyntheticPolicy-2	23785	3	5	40	800
WSO2	90	1	0	1	4
demo-5	86	1	1	1	3
demo-11	126	1	0	1	3
demo-26	100	1	0	1	2
Policy-EMC	125	1	6	3	3
IIC003Policy	105	1	0	1	2

In the experiments, XPTester was compared with the latest XACML policy testing tool X-CREATE [8]. X-CREATE generates XACML requests by assigning attribute values to the intermediate-request obtained by exploring the XACML Context Schema. For each XACML policy, the number of requests X-CREATE can generate is variable, and it is the testers' responsibility to decide how many requests to generate.

All our experiments were executed on a quad-core machine with an Intel Core (TM) i5-2400M 3.10GHz processor and 4GB memory size.

1) *Experiment Subjects*: TABLE III lists the subjects we tested in our experiment, including names of the policies, line of code (LOC) in the XACML file, numbers of type of Combining Algorithm(CA) appearing in the policies, Policy-Set, Policy and Rule. *continue-a* is used as experiment subject to evaluate *Cirg* [10], collected from a real-world conference management system. *SyntheticPolicy-1* and *SyntheticPolicy-2* are synthetic policies obtained from the benchmarks used by Xengine [15]. These two policies are huge in size and we used them to evaluate the effectiveness of XPTester to deal with large-scale policies. *WSO2* is a sample policy composed by WSO2, which is a leading company in service-oriented architecture solutions providers. *demo-5*, *demo-11* and *demo-26* are example policies used to illustrate Fedoras' usage of XACML policy. *Policy-EMC* is an example policy presented by EMC<sup>4</sup>. *IIC003Policy* is one of the test cases for XACML 2.0 conformance tests<sup>5</sup>.

2) *Evaluation Results*: We report the experiment results in Table IV. X-CREATE allows users to decide how many requests to generate [8] while XPTester generates fixed amount of requests. For each of the experimental subjects, we used XPTester to generate one set of requests and X-CREATE to generate two, one of which has the same number of the requests generated by XPTester while the other contains all the requests X-CREATE can generate. We found that the number of requests generated by X-CREATE could be far more than that generated by XPTester, especially when the size of the policy is big. We summarize the evaluation results as follows.

**Result 1:** XPTester is more time efficient to generate even a large number of test requests than X-CREATE.

We define **Time Consumed in Obtaining Requests (TCOR)** of an access control testing framework as the total time spent in obtaining a certain number of requests. Intuitively, the smaller the TCOR is, the more efficient the

---

<sup>4</sup><https://community.emc.com/docs/DOC-7410><sup>5</sup><https://www.oasis-open.org/committees/download.php/14877/ConformanceTests.html>

TABLE IV: Policy coverage and mutant killing ratios achieved by requests generated by XPTester and X-CREATE as well as the time spent in generating requests (TCOR). While evaluating the TCORs, we used the two sets of requests from XPTester and X-CREATE that have the same size.

Policy Under Test	XPTester				X-CREATE(Less Requests)				X-CREATE(Full Requests)		
	# Req	TCOR	Rule cov	Mut killing	# Req	TCOR	Rule cov	Mut killing	# Req	Rule cov	Mut killing
continue-a	229	9.33s	46.31%	37.18%	229	402.47s	22.48%	21.02%	870	46.31%	37.18%
SyntheticPolicy-1	440	17.64s	83.00%	80.07%	440	64.36s	27.00%	57.50%	1134	83.00%	80.07%
SyntheticPolicy-2	756	31.15s	87.75%	90.37%	756	114.10s	28.13%	67.91%	2142	87.75%	90.37%
WSO2	8	8.61s	100.00%	93.33%	8	17.55s	75.00%	83.33%	32	100.00%	93.33%
demo-5	6	10.54s	100.00%	78.57%	6	17.85s	100.00%	64.27%	42	100.00%	78.57%
demo-11	5	8.58s	100.00%	90.91%	5	15.92s	66.67%	86.11%	20	100.00%	90.91%
demo-26	4	9.10s	100.00%	87.50%	4	15.69s	100.00%	50.00%	8	100.00%	87.50%
Policy-EMC	8	9.68s	100.00%	52.63%	8	16.66s	66.67%	18.42%	24	100.00%	52.63%
IIIC003Policy	5	10.61s	100.00%	77.78%	5	15.50s	100.00%	77.78%	48	100.00%	77.78%

framework is in generating test requests. We found that the time of XPTester consumed in generating test requests is almost one tenth of that of X-CREATE on average. This is because X-CREATE generates test requests by exploring the XACML Context Schema and taking the attribute value combinations, both of which are time consuming processes. Especially, for policies with large number of attribute values, such as *continue-a*, the time X-CREATE spends in generating test requests exploded.

**Result 2:** XPTester is able to generate less test requests while achieving the same policy coverage as X-CREATE.

Regarding the policy coverage of requests generated by XPTester and X-CREATE, the results show that X-CREATE achieves less Rule coverage than XPTester when generating the same number of requests. For an unbiased comparison, we also experimented to have X-CREATE achieve the same rule coverage as XPTester did by generating a larger set of requests. It is worth noting that XPTester generated far less number of requests than X-CREATE. X-CREATE achieves certain degree of policy coverage by simply taking combinations of all attribute values that may lead to redundant requests.

**Result 3:** XPTester is more capable of detecting faults in XACML policies than X-CREATE with less test requests.

We carried out mutant testing to evaluate the fault detection capability of test requests. We referred to the mutation operation proposed by Martin [17] and automatically generated mutant policies for each subject. We observed that the mutation killing ratios of requests from XPTester and X-CREATE are the same when using X-CREATE to generate the maximum number of requests it could. However, if we utilized X-CREATE to generate the same number of requests as XPTester, its mutant killing ratios declined significantly compared to XPTester. This is intuitively understandable as fault detection capability associates with policy coverage ratio.

Our experiment results confirmed that XPTester is efficient in generating test requests that have high fault detection ability for XACML policies. The efficiency and effectiveness of XPTester are due to following facts. First, XPTester is useful in reducing the time spent in requests generation. Second, the generated requests cover the policy adequately and have strong fault detection capability. In addition, XPTester generates

relatively small number of requests while achieving the same policy coverage and fault detection capability comparing to the state-of-the-art tools, such as X-CREATE.

## V. RELATED WORK

There are many research projects on deploying and testing access control for mobile devices. El-Aziz and Kannan propose a framework that uses XACML to support access control over the data access of handheld devices [2]. Ulltveit-Moe and Oleshchuk study mobile security with location-aware role-based access control by implementing access control policy using XACML [3]. Abdunabi et al. propose a new spatio-temporal role-based access control model that considers time of access and location of the mobile device users [4]. To avoid modifying the operating system, Android specifically, while enhancing a security policy, Xu et al. propose a method that is able to repackage applications to enclose the policy to facilitate runtime monitoring of application behaviors [6]. An Android runtime security policy enforcement framework is proposed by Banuri et al. that validates the behavior of an application through its permission exercising patterns [18]. In terms of securing the scenario of Bring Your Own Device (BYDO), Rmando et al. propose a security framework for mobile devices to restrict the installation of applications to assure the security of the resources under protection [19]. These research demonstrates the need of access control for mobile security and XACML is becoming an important deployment candidate for policy specification and enforcement.

For security policy testing, Mouelhi et al. and Le Traon et al. propose a technique that transforms the test cases for functional testing into test cases for security policy testing [20], [21]. There are several techniques proposed and tools developed for XACML policy testing. Bertolino et al. propose a test strategy to generate XACML requests from the XACML Context Schema that leads to X-CREATE [8]. Martin and Xie develop Targen [9], in which the policy under test is converted into a tree structure representation. It generates XACML requests by combining all the possible combinations of attributes values satisfying the constraint along the tree's paths. X-CREATE and Targen, however, generate too many redundant test requests due to the randomness nature of their request generation approaches. Another tool called Cirg is

also developed by Martin and Xie [10]. Cirq utilizes the change-impact analysis capability of Margrave [22] to generate XACML requests. However, change-impact's efficiency depends on the size of the policy [10].

The work most similar to ours is proposed by Yu et al. [11], in which they also propose a security policy testing approach based on code generation. They utilize concolic testing [23] to generate test inputs for program module by first converting the policy under test into program module like JAVA code. The concolic testing tool they use is JCUTE [24]. After obtaining test inputs from JCUTE, the inputs are translated into access requests for testing purpose. While both leverages code generation, Yu's work has several limitations. First, the program module generation process fails to take into comprehensive consideration of the complex semantics of the policy language like various Functions defined in XACML. Second, as far as we know, JCUTE does not offer any programming interfaces thus the test inputs generation and translation must be done with manual effort, thus it cannot be completely automated. Finally, the tester has to audit the evaluation results of the test requests manually, which is a tedious and error-prone exercise.

## VI. CONCLUSION AND FUTURE WORK

This paper proposes an automatic approach to systematically test access control policies specified in XACML by employing symbolic execution techniques. First, XACML policy is converted into runnable code. Second, the code is symbolically executed to generate highly adequate and non-redundant test inputs. Final, the test inputs are then used to compose access requests which can be automatically evaluated to report defects with existing tools. We have implemented a supporting tool and conducted extensive experiments upon real-world policies. The experiments demonstrate the applicability, scalability, efficiency and effectiveness of our approach.

In the future, we will continue our work in three directions. First, we plan to extend our approach to support more access control policy languages. Second, the approaches for testing PEP or obligations are yet to be developed. Third, we will work on mobile security, for example, to test mobile apps running on mobile devices, whose access control policies are specified with XACML.

## ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China (No. 91118007, No. 61328491, No. 61170066), and by the National 863 High-Tech Program of China (No. 2012AA011205).

## REFERENCES

- [1] E. I. Tim Moses, *eXtensible Access Control Markup Language(XACML) Version 2.0*, OASIS, February 2005.
- [2] A. A. El-Aziz and A. Kannan, "Access control for healthcare data using extended xacml-srbac model," in *2012 International Conference on Computer Communication and Informatics (ICCCI)*. IEEE, 2012, pp. 1–4.
- [3] N. Ulltveit-Moe and V. Oleshchuk, "Mobile security with location-aware role-based access control," in *Security and Privacy in Mobile Information and Communication Systems*. Springer, 2012, pp. 172–183.
- [4] R. Abdunabi, I. Ray, and R. France, "Specification and analysis of access control policies for mobile applications," in *Proceedings of the 18th ACM symposium on Access control models and technologies*. ACM, 2013, pp. 173–184.
- [5] S. Arunkumar and M. Rajarajan, "Healthcare data access control using xacml for handheld devices," in *Developments in E-systems Engineering (DESE), 2010*. IEEE, 2010, pp. 35–38.
- [6] R. Xu, H. Saïdi, and R. Anderson, "Aurasium: Practical policy enforcement for android applications," in *Proceedings of the 21st USENIX conference on Security symposium*. USENIX Association, 2012, pp. 27–27.
- [7] M. Poulmenopoulou, F. Malamateniou, and G. Vassilacopoulos, "An access control framework for pervasive mobile healthcare systems utilizing cloud services," in *Wireless Mobile Communication and Healthcare*. Springer, 2012, pp. 380–385.
- [8] A. Bertolino, F. Lonetti, and E. Marchetti, "Systematic xacml request generation for testing purposes," in *2010 36th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2010, pp. 3–11.
- [9] E. Martin, "Automated test generation for access control policies," in *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*. ACM, 2006, pp. 752–753.
- [10] E. Martin and T. Xie, "Automated test generation for access control policies via change-impact analysis," in *Third International Workshop on Software Engineering for Secure Systems, 2007. SESS'07: ICSE Workshops 2007*. IEEE, 2007, pp. 5–5.
- [11] T. Yu, D. Sivasubramanian, and T. Xie, "Security policy testing via automated program code generation," in *Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research: Cyber Security and Information Intelligence Challenges and Strategies*. ACM, 2009, p. 13.
- [12] J. C. King, "Symbolic execution and program testing," *Communications of the ACM*, vol. 19, no. 7, pp. 385–394, 1976.
- [13] C. Cadar, D. Dunbar, and D. R. Engler, "Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs," in *OSDI*, vol. 8, 2008, pp. 209–224.
- [14] IBM, "Byod: Bring your own device," <http://www.ibm.com/mobilefirst/us/en/bring-your-own-device/byod.html>.
- [15] A. X. Liu, F. Chen, J. Hwang, and T. Xie, "Xengine: a fast and scalable xacml policy evaluation engine," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 36, no. 1. ACM, 2008, pp. 265–276.
- [16] C. Lattner, "The llvm compiler infrastructure," <http://www.llvm.org/>.
- [17] E. Martin and T. Xie, "A fault model and mutation testing of access control policies," in *Proceedings of the 16th international conference on World Wide Web*. ACM, 2007, pp. 667–676.
- [18] H. Banuri, M. Alam, S. Khan, J. Manzoor, B. Ali, Y. Khan, M. Yaseen, M. N. Tahir, T. Ali, Q. Alam *et al.*, "An android runtime security policy enforcement framework," *Personal and Ubiquitous Computing*, vol. 16, no. 6, pp. 631–641, 2012.
- [19] A. Armando, G. Costa, and A. Merlo, "Bring your own device, securely," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. ACM, 2013, pp. 1852–1858.
- [20] Y. L. Traon, T. Mouelhi, and B. Baudry, "Testing security policies: going beyond functional testing," in *The 18th IEEE International Symposium on Software Reliability, 2007. ISSRE'07*. IEEE, 2007, pp. 93–102.
- [21] T. Mouelhi, Y. Le Traon, and B. Baudry, "Transforming and selecting functional test cases for security policy testing," in *International Conference on Software Testing Verification and Validation, 2009. ICST'09*. IEEE, 2009, pp. 171–180.
- [22] K. Fislser, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz, "Verification and change-impact analysis of access-control policies," in *Proceedings of the 27th international conference on Software engineering*. ACM, 2005, pp. 196–205.
- [23] K. Sen, "Concolic testing," in *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*. ACM, 2007, pp. 571–572.
- [24] K. Sen and G. Agha, "Cute and jcute: Concolic unit testing and explicit path model-checking tools," in *Computer Aided Verification*. Springer, 2006, pp. 419–423.

# Reducing Test Cases with Causality Partitions

Haijun Wang<sup>1</sup>, Xiaohong Guan<sup>1</sup>, Qinghua Zheng<sup>1</sup>, Ting Liu<sup>1,\*</sup>, Xiangyang Li<sup>1</sup>, Lechen Yu<sup>1</sup>, Zijiang Yang<sup>2,3</sup>

1. MOE Key Lab. for Intelligent Networks and Network Security, Xi'an Jiaotong University, Xi'an, China

2. College of Computer and Technology, Xi'an University of Technology, Xi'an, China

3. Western Michigan University, Kalamazoo MI, U.S.A.

{hjiang, xhguan, tliu, xyli, lcyu}@sei.xjtu.edu.cn; qhzheng@mail.xjtu.edu.cn; zijiang.yang@wmich.edu

**Abstract**—Automatic test case generation using symbolic execution suffers from the problem of *path explosion*: the number of paths to be explored may grow exponentially with the scale of a program. We believe that different paths may exhibit some similar program behaviors, thus it is unnecessary to explore all of the paths to generate test cases. In this paper, a novel model of program causality is proposed to extract all kinds of influences among statements of a program, which consists of four types of program dependencies. Then, a heuristic approach based on program causality is developed to selectively explore program paths in symbolic execution. We have implemented a prototype of our approach within the symbolic execution engine of Java Pathfinder. Experiments on six public benchmarks show that our approach can significantly reduce the number of explored paths and the symbolic execution time so that to improve the performance of test case generation.

**Keywords**-Symbolic Execution; Causality Partition; Test Case Generation;

## I. INTRODUCTION

Automatic test case generation using symbolic execution [1-3] has recently regained prominence as a technique for systematically exploring the paths of a program. In practice though, symbolic execution suffers from the inherent problem of *path explosion* where the number of paths to be explored can increase exponentially with the scale of a program. While there are techniques for mitigating *path explosion*, e.g. by using compositionality [4], abstraction refinement [5], and parallelization [6], systematically exploring program paths remains a daunting task even for medium-sized programs.

In this paper, we propose a novel test case generation technique that selectively explores program paths based on program causality. The intuition is that the path with the same causal relationship with already explored paths does not reveal additional program behavior. Figure 1 shows a program that may raise *division-by-zero* exceptions at Lines 11 and 13. Table I gives all eight possible paths of the program. For brevity we only list branch statements of a path. As indicated by *Column 3*, paths  $\pi_5$  and  $\pi_7$  can raise the exception at Line

```
void Test(int x, int y, int m) {
1: int a=1;
2: int b=1;
3: int c=2;
4: int d=3;
5: int first_out, second_out;
6: if(x+y < 0)
7:   a=3;
8:   if(m > 2)
9:     d=4;
10:  if(x-y > 1)
11:   c=1/(a-b);
12:  first_out=c;
13:  second_out=1/(a-d);
}
```

Figure 1. A Running Program.

11, and paths  $\pi_3$  and  $\pi_4$  can raise the exception at Line 13. *Columns 4-8* mark the necessary paths for kinds of coverage criteria, respectively. For statement coverage  $T_{\text{Statement}}$ , path  $\pi_1$  is sufficient. For branch coverage  $T_{\text{Branch}}$ , two paths  $\pi_1$  and  $\pi_8$  are satisfiable. In order to achieve the path coverage  $T_{\text{Path}}$ , all eight paths have to be executed. As expected, statement and branch coverage require much fewer test cases than path coverage, at the cost of missing both exceptions at Lines 11 and 13. *Column 7* lists the paths that are covered by our approach. Note that with half of the paths to be explored, our approach can still catch both exceptions. Comparing  $\pi_5$  and  $\pi_7$ , we can observe that the decision at Line 8 makes no difference on the potential exception at Line 11. Therefore a testing that covers either  $\pi_5$  or  $\pi_7$  can report the exception at Line 11. Similarly, it is not necessary to explore both  $\pi_3$  and  $\pi_4$  for detecting the exception at Line 13. This paper addresses a more general question: without considering any particular type of fault, is it possible to achieve the same testing effect as path coverage without exploring all paths? By exploring four instead of eight paths, Table I shows the answer is positive. In our approach we exploit causal relationship to achieve the path reduction. However, the existing program dependence definitions are not sufficient for causal relationship. Therefore in this paper we introduce a new type of program dependence called **Correlation Dependence**. The last column in Table I shows why correlation dependence is indispensable, as without it the exception at Line 13 cannot be detected.

As illustrated by the example, the key insight of our approach is to prevent some irrelevant statements combination from generating redundant paths. We first extract causal dependencies of a program, which consist of control, data, potential dependence and newly proposed correlation dependence. Then, we conduct symbolic execution to selectively explore program paths through exploiting the

\* The corresponding author is Ting Liu (tliu@sei.xjtu.edu.cn)

The work was supported in part by National Science Foundation of China under Grant (91118005, 91218301, 61221063, 61203174, U1301254), National High Technology Research and Development Program 863 of China under Grant (2012AA011003), Key Projects in the National Science and Technology Pillar Program of China (2012BAH16F02) and the Fundamental Research Funds for the Central Universities.

TABLE I. TEST CASES FOR KINDS OF COVERAGE CRITERIA

No	Execution	Error	T <sub>Statement</sub>	T <sub>Branch</sub>	T <sub>Path</sub>	T <sub>Causal</sub>	T <sub>w/o</sub>
$\pi_1$	<6T,8T,10T>		√	√	√	√	√
$\pi_2$	<6T,8T,10F>				√		
$\pi_3$	<6T,8F,10T>	13			√	√	
$\pi_4$	<6T,8F,10F>	13			√		
$\pi_5$	<6F,8T,10T>	11			√	√	√
$\pi_6$	<6F,8T,10F>				√		
$\pi_7$	<6F,8F,10T>	11			√		
$\pi_8$	<6F,8F,10F>			√	√	√	√

causal dependencies. We have implemented a prototype of our approach within the framework of JPF-SE [1] and conducted experiments on six public benchmarks. The results validate the effectiveness of our approach. Compared with the original JPF-SE, our approach can reduce the number of explored paths by 46.28% to 98.84%, and achieve a speedup of 2.47X to 49.12X. We also compare our approach with FlowTest [8], where our approach can reduce the number of explored paths by 49.37% to 98.86%, and achieve a speedup of 1.05X to 61.40X. Additional experiments on the growth rates of the number of explored paths and usage time indicate that our approach can achieve orders of magnitude improvement.

Our main contributions are summarized as follows.

- We propose a novel approach to reduce test cases through exploiting the causality exhibited in program paths. Such reduction is sound.
- We extract causality in terms of control dependence, data dependence, potential dependence and newly proposed correlation dependence. This can help to comprehensively and systematically understand the causal dependencies among program statements.
- We have implemented a prototype of our approach within the framework of JPF-SE and conducted experiments on six public benchmarks. The results show that our approach can dramatically reduce the number of explored paths and the symbolic execution time. Additional experiments on growth rates of the explored path size and usage time indicate that our approach can offer orders of magnitude improvement.

## II. REDUCING TEST CASES WITH CAUSALITY PARTITIONS

### A. Relevant Definitions

Control flow graph (CFG) of a procedure is a directed graph that can be represented by  $CFG = \langle N, E \rangle$ , where  $N$  is the set of nodes and  $E \subseteq N \times N$  represents possible execution flow between the nodes.  $Br \subseteq N$  is the set of branches. Also, we use  $\bar{n} \in Br$  to represent the opposite branch of  $n \in Br$ .

**Definition 1 Control Dependence** is a map  $controlD: Br \times N \rightarrow \{T, F\}$  that returns true for a pair of nodes  $(n_i, n_j)$  if node  $n_i$  has more than one exit paths, and one of the exit paths must result in  $n_j$  being executed but the other exit path may result in  $n_j$  not being executed; Otherwise it returns false.

**Definition 2 Data Dependence** is a map  $dataD: N \times N \rightarrow \{T, F\}$  that returns true for a pair of nodes  $(n_i, n_j)$  if there exists a path  $\pi$  from  $n_i$  to  $n_j$  where  $n_i$  defines a variable  $v$ ,  $n_j$  uses the

variable  $v$  and any  $n_k$  ( $i < k < j$ ) on  $\pi$  does not redefine the variable  $v$ ; Otherwise it returns false.

**Definition 3 Potential Dependence** [9] is a map  $potentialD: Br \times N \rightarrow \{T, F\}$  that returns true for a pair of nodes  $(n_i, n_j)$ , if (1) there exists a path  $\pi$  from  $n_i$  to  $n_j$  where  $n_j$  uses a variable  $v$  and the definition of  $v$  occurs before  $n_i$ ; (2)  $n_j$  is not control dependent on  $n_i$ ; (3) a different definition of  $v$  could potentially reach  $n_j$  if  $n_i$  is evaluated differently; Otherwise it returns false.

We define **Traditional Dependence** as a map  $traditionalD: N \times N \rightarrow \{T, F\}$  that returns true for a pair of nodes  $(n_i, n_j)$  if  $controlD(n_i, n_j) \vee dataD(n_i, n_j) \vee potentialD(n_i, n_j)$ ; Otherwise it returns false.

**Definition 4 Correlation Dependence** is a map  $correlationD: N \times N \rightarrow \{T, F\}$  that returns true for a pair of nodes  $(n_i, n_j)$  if there exists a path  $\pi = \langle \dots n_i \dots n_j \dots n_k \dots \rangle$ , and  $n_i, n_j, n_k$  satisfy (1)  $traditionalD(n_i, n_k) \wedge (traditionalD(n_j, n_k) \vee correlationD(n_j, n_k))$ ; or (2)  $(traditionalD(n_i, n_k) \vee correlationD(n_i, n_k)) \wedge traditionalD(n_j, n_k)$ ; Otherwise it returns false.

We define **Causal Dependence** as a map  $causalD: N \times N \rightarrow \{T, F\}$  that returns true for a pair of nodes  $(n_i, n_j)$  if  $traditionalD(n_i, n_j) \vee correlationD(n_i, n_j)$ ; Otherwise it returns false. Table II lists all causal dependencies of the example.

TABLE II. DEPENDENCIES OF THE PROGRAM IN FIGURE 1

Type	Dependent Pairs
ControlD	(6T,7) (8T,9) (10T,11)
DataD	(1,11) (1,13) (2,11) (3,12) (4,13) (7,11) (7,13) (9,13) (11,12)
PotentialD	(6F,11) (6F,13) (8F,13) (10F,12)
CorrelationD	(1,2) (1,4) (1,6F) (1,8T) (1,8F) (1,9) (1,10T) (2,6T) (2,6F) (2,7) (2,10T) (3,10F) (4,6T) (4,6F) (4,7) (4,8F) (6F,8T) (6F,8F) (6F,9) (6F,10T) (7,8T) (7,8F) (7,9) (7,10T)

**Definition 5 A covered causal chain**  $c_\pi$  of path  $\pi$  is a sequence  $V = [n_1, \dots, n_m]$  such that

- $\exists n_i \in \pi, causalD(n_i, n_1)$ , and
- $\forall n_j \in V, traditionalD(n_i, n_j) \rightarrow n_i \in V$ , and
- $\forall n_j, n_{j+1} \in V, causalD(n_j, n_{j+1})$ .

**Definition 6 A discovered causal chain**  $d_\pi$  of path  $\pi$  is a sequence  $V = [n_1, \dots, \bar{n}_m]$  that is a **covered causal chain** of path  $\pi'$  generated by flipping the branch  $n_m$  to  $\bar{n}_m$  in path  $\pi$ .

### B. Computation of Causal Dependence

Since control, data and potential dependence are widely applied in the software engineering and can be obtained by the existing tools [8, 9], in this paper we focus on the computation of newly proposed correlation dependence.

We first compute the reachability of node pair  $(n_i, n_j)$  using Equations (1), (2) and (3). Similar to the data flow analysis via graph reachability [11], we use the node pair in the computation. Note that  $n_0$  represents start node of CFG, which has no incoming node pair. The outgoing node pairs of  $n_j$

consist of two parts: the node pairs generated at  $n_j$ , and the subtraction of the incoming node pairs by the node pairs killed at  $n_j$ . Given the incoming nodes  $N_j$  at  $n_j$ , obtained through data flow analysis [11], the node pair  $(n'_j, n_j)$  is generated at  $n_j$  through pairing every  $n'_j \in N_j$  with  $n_j$ . A node pair  $(n_p, n_q)$  is killed at  $n_j$  if either  $n_p$  or  $n_q$  is redefined by  $n_j$ . We consider  $n_p$  or  $n_q$  is redefined by  $n_j$ , if  $n_p$  or  $n_q$  is the definition statement and the variable they define is the same with the variable defined at  $n_j$ , or  $n_p$  or  $n_q$  is the conditional statement and they no longer directly or indirectly control  $n_j$ . Assuming  $N_{pre}$  is the set of nodes that are immediate predecessors of  $n_j$  in CFG, the incoming node pairs of  $n_j$  are the union of the outgoing node pairs of the nodes in  $N_{pre}$ .

$$in[n_0] = \{\}$$

$$out[n_j] = gen[n_j] \cup (in[n_j] - kill[n_j])$$

$$in[n_j] = \cup_{n_i \in N_{pre}} out[n_i]$$

We then compute the correlation dependence. For each node  $n_j$  with an incoming node pair  $(n_p, n_q)$ , if either  $traditionalD(n_p, n_j) \wedge causalD(n_q, n_j)$  or  $causalD(n_p, n_j) \wedge traditionalD(n_q, n_j)$ , it is considered that  $n_q$  is correlation dependent on  $n_p$ .

### C. Reducing Test Cases with Causality Partitions

**Overview.** Algorithm 1 gives a high level overview of our approach. It maintains two global data structures  $C$  and  $D$ .  $C$  is the set of covered causal chains and  $D$  is the set of discovered causal chains that are yet to be covered. The algorithm first starts symbolic execution with a random input (Line 2). Each subsequent path exploration is directed by a discovered causal chain  $d$  removed from  $D$  (Line 5). At Line 6, if  $d$  is satisfiable, we will obtain a new input such that  $d$  can be covered by the execution under the newly acquired input. When we obtain an input, we use the procedure *reducedPathExploration* to handle the sets  $C$  and  $D$  (Line 8), which are used to drive the future path explorations. The algorithm terminates when  $D$  becomes empty. The computation of the input based on a discovered causal chain  $d$  (Line 6) and the algorithm for procedure *reducedPathExploration* (Line 8) will be discussed later.

Table III gives a complete procedure of our approach in the example program. *Column 1* shows the discovered causal chains that are yet not covered and then used to guide further path explorations. The generated inputs and corresponding paths are given in *Columns 2 and 3*, respectively. *Columns 4 and 5* list the set of covered causal chains and discovered causal chains. Note that although we pick only *one* discovered causal chain in each iteration, the newly explored path may cover many previously discovered causal chains that are still not covered. Therefore it may reduce the size of  $D$  significantly. As illustrated in Table III, four paths can cover all possible causal chains instead of eight paths in standard symbolic execution.

Table IV explains why neither  $\pi_3$  nor  $\pi_4$  is explored without considering correlation dependence and hence the exception at Line 13 is missed. The exception at Line 13 can only be raised by the path that executes both 6T and 8F. Since there does not

---

### Algorithm 1 *reducedSymbolicExecution*(Program $P$ )

---

```

1:  $C = D = \emptyset$ 
2:  $input = randomInput()$ 
3:  $reducedPathExploration(P, input, C, D)$ 
4: while  $D \neq \emptyset$  do
5:    $d = D.remove()$ 
6:    $input = obtain\ an\ input\ based\ on\ d$ 
7:   if  $input \neq \emptyset$  then
8:      $reducedPathExploration(P, input, C, D)$ 
9:   end if
10: end while

```

---

exist any control, data or potential dependence between 6T and 8F, symbolic execution without correlation dependence will not combine 6T and 8F to generate a new path after exploring  $\pi = \langle 6T, 8T, 10T \rangle$ , which leads to incompleteness.

### Causal Chain Computation during Path Exploration.

Algorithm 2 focuses on computing the covered and discovered causal chains once a path  $\pi$  is obtained. We will explain the algorithm through Table V, which illustrates the procedures under the path  $\pi = \langle 1, 2, 3, 4, 6T, 7, 8T, 9, 10T, 11, 12, 13 \rangle$ . Let  $C_\pi$  and  $D_\pi$  be the set of covered and discovered causal chains of  $\pi$ , respectively. Initially, as shown in *row 1*,  $C_\pi = \{[1], [3]\}$  and  $D_\pi = \{\}$ . Both nodes 1 and 3 appear in  $\pi$  and they are not causal dependent on any other nodes in  $\pi$  (Line 2).  $D_\pi$  is empty as there is no node in  $\pi$  whose negated node is not causal dependent on any other nodes in  $\pi$  (Line 3). We illustrate Lines 5-15 in Algorithm 2 using *rows 6 and 13* in Table V.  $c_\pi.branch$  at Line 6 represents the sub sequence of  $c_\pi$  that consists of all its branches. In fact, the longer causal chain is generated by extending a node into the previous shorter causal chain. In *row 6*, the previous shorter covered causal chain  $c_\pi$  is  $[1, 2, 6T, 7, 10T]$  that is generated at *row 5* and  $EXT$  is  $\{11\}$ .  $EXT$  represents the set of extensible nodes. If a node  $n_j$  is causal dependent on the last node of previous shorter covered causal chain  $c_\pi$  and the set of nodes that are traditional dependent by  $n_j$  are all included in  $c_\pi$ ,  $n_j$  can be an extensible

TABLE III. SYMBOLIC EXECUTION WITH CAUSALITY

Causal Chain	Input	Path	$C$	$D$
None	(0,-2,3)	$\langle 6T, 8T, 10T \rangle$	[6T] [6T,8T] [6T,10T] [8T] [10T]	[6F] [6T,8F] [8F] [10F]
[6F]	(2,0,3)	$\langle 6F, 8T, 10T \rangle$	[6F] [6F,8T] [6F,10T]	[6F,8F]
[6T, 8F]	(0,-2,0)	$\langle 6T, 8F, 10T \rangle$	[6T,8F] [8F]	
[10F]	(1,0,0)	$\langle 6F, 8F, 10F \rangle$	[6F,8F] [10F]	

TABLE IV. SYMBOLIC EXECUTION WITHOUT CORRELATION DEPENDENCE

Causal Chain	Input	Path	$C$	$D$
None	(0,-2,3)	$\langle 6T, 8T, 10T \rangle$	[6T] [6T,10T] [8T] [10T]	[6F] [8F] [10F]
[6F]	(2,0,3)	$\langle 6F, 8T, 10T \rangle$	[6F] [6F,10T]	
[10F]	(1,0,0)	$\langle 6F, 8F, 10F \rangle$	[8F] [10F]	

**Algorithm 2** *reducedPathExploration*( $P$ , *input*,  $C$ ,  $D$ )

---

```

1:  $\pi$  = execute  $P$  under input
2:  $C_\pi = \{[n_j] | n_j \in \pi \wedge \neg \text{causalD}(n_i, n_j) \text{ for any other } n_i \in \pi\}$ 
3:  $D_\pi = \{[n_j] | \bar{n}_j \in \pi \wedge \neg \text{causalD}(n_i, n_j) \text{ for any other } n_i \in \pi\}$ 
4: while  $C_\pi \neq \emptyset$  do
5:    $c_\pi = C_\pi.\text{remove}()$ 
6:    $C = C \cup \{c_\pi.\text{branch}\}$ 
7:    $EXT = \{n_j | \text{causalD}(c_\pi.\text{lastNode}, n_j) \wedge \forall n_i (\text{traditionalD}(n_i, n_j) \rightarrow n_i \in c_\pi)\}$ 
8:   for each node  $ext \in EXT$  do
9:      $c'_\pi = c_\pi.\text{append}(ext)$ 
10:    if  $ext \in \pi$  then
11:       $C_\pi = C_\pi \cup \{c'_\pi\}$ 
12:    else
13:       $D_\pi = D_\pi \cup \{c'_\pi\}$ 
14:    end if
15:  end for
16: end while
17:  $D = (D \cup D_\pi.\text{branch}) - C$ 

```

---

node of  $c_\pi$  (Line 7). Note that  $c_\pi.\text{lastNode}$  at Line 7 refers to the last node of  $c_\pi$ . Since node 11 is causal dependent on branch 10T and the set of nodes  $\{2,7,10T\}$  that are traditional dependent by node 11 are all included in  $c_\pi$ , node 11 is an extensible node. Obviously,  $ext \in \pi$  and finally  $c'_\pi$  is added into  $C_\pi$ . In row 13, the previous shorter covered causal chain  $c_\pi$  is  $[3]$  and  $EXT$  is  $\{10F\}$ .  $c_\pi$  is appended as  $c'_\pi = [3,10F]$  and  $c'_\pi$  is added into  $D_\pi$  because  $ext \notin \pi$ .

**From Discovered Causal Chain to Input.** Given a discovered causal chain  $d$ , we will compute an input so that the execution under the newly acquired input can cover  $d$ . In this section we use an example to illustrate the computation.

Assume  $d = [10F]$  and we would like to obtain a path  $\pi$  of the program in Figure 1 such that  $\pi$  can cover  $d$ . Treating inputs as symbolic variables we start a symbolic execution. At the first conditional statement (Line 6) we allow symbolic execution to choose any branch because  $d$  does not specify a particular branch at Line 6. The fact that  $d$  does not specify Line 6 means that there is no statement in  $d$  that is causal dependent on Line 6. Therefore symbolic execution can take any branch at Line 6. Next symbolic execution encounters the conditional statement Line 8. Since the branch at Line 8 is also not specified by  $d$ , symbolic execution still executes randomly. Finally symbolic execution reaches the conditional statement Line 10. Since 10F is specified in  $d$ , symbolic execution takes the *false* branch and adds term  $x-y \leq 1$  into the path constraints  $\zeta$  to match the requirement of 10F in  $d$ . If  $\zeta$  is unsatisfiable, the procedure terminates and no input can be found to match  $d$ , otherwise symbolic execution continues. As there is no more conditional constraint in  $d$ , we pass  $\zeta$  to SMT solver. If  $\zeta$  is satisfiable, the solution forms an input that can cover  $d$ ; otherwise there is no valid path to cover  $d$ .

**Theorem 1.** Given a program  $P$ , our approach can cover all feasible causal chains of  $P$ .

**Example.** Assume a causal chain  $V = [1,4,6T,7,8F,13]$  of the program in Figure 1 is required to be covered, which can detect the exception at Line 13. As shown in rows 9 and 10 of Table V, the initial path  $\pi = \langle 1,2,3,4,6T,7,8T,9,10T,11,12,13 \rangle$  is able to cover the causal chain  $V' = [1,4,6T,7]$  and discover the

TABLE V. CAUSAL CHAINS OF  $\Pi = \langle 1,2,3,4,6T,7,8T,9,10T,11,12,13 \rangle$ 

No.	$c_\pi$	<i>ext</i>	$C_\pi$	$D_\pi$
1		1	[1]	
		3	[3]	
		2	[1,2]	
2	[1]	4	[1,4]	
		6F		[1,6F]
		8T	[1,8T]	
		8F		[1,8F]
		10T	[1,10T]	
		6T	[1,2,6T]	
3	[1,2]	6F		[1,2,6F]
		10T	[1,2,10T]	
		7	[1,2,6T,7]	
4	[1,2,6T]	7	[1,2,6T,7]	
5	[1,2,6T,7]	10T	[1,2,6T,7,10T]	
6	[1,2,6T,7,10T]	11	[1,2,6T,7,10T,11]	
7	[1,2,6T,7,10T,11]	12	[1,2,6T,7,10T,11,12]	
8	[1,4]	6T	[1,4,6T]	
		6F		[1,4,6F]
		8F		[1,4,8F]
9	[1,4,6T]	7	[1,4,6T,7]	
10	[1,4,6T,7]	8T	[1,4,6T,7,8T]	
		8F		[1,4,6T,7,8F]
		10T	[1,4,6T,7,10T]	
11	[1,4,6T,7,8T]	9	[1,4,6T,7,8T,9]	
12	[1,4,6T,7,8T,9]	13	[1,4,6T,7,8T,9,13]	
13	[3]	10F		[3,10F]

causal chain  $V'' = [1,4,6T,7,8F]$  through extending the node 8F to  $V'$ . As stated in our algorithm, the discovered causal chain  $V''$  then is used to direct symbolic execution to generate an input  $input = \langle x=0, y=-2, m=0 \rangle$  which can cover  $V''$ , as shown in Table III. Simultaneously, *input* can cover the required causal chain  $V$ .

## III. RELATED WORK

Symbolic execution [1-3] has emerged as a popular technique for testing real-world software applications. However, symbolic execution suffers from the inherent path explosion problem. Many techniques have been proposed to alleviate this problem, and they can be roughly divided into four categories. The first category addresses the path explosion problem, e.g., [5], through integrating abstraction to reduce the search space. In the second category (e.g. [4]), researchers generate method summaries to enable more efficient constraint solving. The approaches of the third category, e.g., [8, 9, 12, 13], adopt the notion of the path equivalence to avoid full path exploration. Finally the techniques in the fourth category, e.g., [14, 15], limit the analysis scope to avoid irrelevant path exploration.

Our approach shares similarities with the third category. The technique in [9] partitions the paths based on the program symbolic output. Two paths are placed in the same partition if the symbolic expressions connecting the input of a program with the output are the same. During symbolic execution only one path from each partition is explored. The main difference from our approach is that [9] ignores program crashes that do not come from the unexpected outputs, while our approach can avoid such situation. In work [13], a program is statically decomposed into some path families, where each path family contains several paths that share similar program behaviors.

TABLE VI. SYMBOLIC EXECUTION WITH AND WITHOUT CAUSALITY REDUCTION

Subject	LoC	Time(s)			#Explored Paths			#Infeasible Paths			#Avg. PC		
		Causal	Flow	JPF	Causal	Flow	JPF	Causal	Flow	JPF	Causal	Flow	JPF
WBS	231	196	539	519	4123	13840	13824	0	0	0	28.8	56.0	56.1
Jtcas	236	734	1906	1813	101	216	188	214	1169	1138	20.2	23.9	24.4
Totinfo	375	7946	36224	34998	37	225	204	306	3536	3151	54.2	93.8	93.7
Siena	1529	39	41	596	40	79	1728	104	104	471	7.2	6.7	20.2
Wc	1557	25	1535	1228	163	2626	2596	130	2630	2590	4.3	23.2	23.5
Sum	1257	37	720	696	17	1490	1464	23	1460	1458	6.5	17.8	18.0

Authors argue that a program can be analyzed at the granularity of path family instead of individual path. A significant difference from our approach is that [13] does not consider the interaction of statements that are correlation dependent. The technique in [12] partitions paths based on program states. Two program paths are equivalent if symbolic states of all live variables are the same.

The work closest to our approach is FlowTest [8], where the inputs of a program are partitioned into non-interfering blocks such that symbolically solving an input block while keeping other blocks assigned with concrete values can find the same set of assertion violations as symbolically solving for the entire inputs. Two inputs are considered of interfering, if they are jointly relevant to the same statement or they are relevant to the statements that have dependent relationship in some program path. By partitioning the program inputs, FlowTest can avoid full path exploration as our approach. In this paper, we have re-implemented FlowTest within the framework of JPF-SE and empirically compared it with our approach in Section IV. The main difference between two techniques is the different partition criteria. The technique FlowTest partitions the paths based on the dependencies of the program inputs, while our approach partitions the paths based on the dependencies of the program statements. Our approach enjoys finer granularity as if two statements are dependent, the inputs that are relevant to these two statements must be dependent; it may be not true for the opposite situation.

#### IV. EMPIRICAL STUDY

We implemented a prototype of our approach within the framework of JPF-SE [1] and used the constraint solver Z3 [16] to check the satisfiability of path conditions. To compute the causal dependence we used Indus [10], an inter-procedural analysis tool based on Soot for Java byte code. Our benchmark suite consists of six programs with lines of code ranging from 200 to 2000 (*Column LoC* in Table VI). WBS is a synchronous reactive component from the automotive domain. Its Java implementation is based on a Simulink model derived from the WBS case example presented in ARP 4761 [15]. Jtcas, Totinfo and Siena are obtained from the SIR repository [17, 18]. Jtcas is an aircraft collision avoidance system. Totinfo is used to compute the statistics of the given input data. Siena implements partial event notification architecture. WC and SUM are obtained from the GNU Coreutils suite [2]. WC calculates the number of bytes, words, and lines in a file. SUM checksums and counts the number of blocks in a file. Totinfo, WC and SUM are written in C and we manually translated them to Java.

We conducted two groups of experiments. In the first group, we compared our approach against standard symbolic execution JPF-SE and FlowTest proposed in [8]. We reported the symbolic execution time, the number of explored paths and other data on six benchmark programs. In the second group, we examined the growth rates of the explored path size and usage time with respect to the number of input parameters, through using our approach and other two peer techniques. The benchmark programs used in the second group are WC and SUM, because both programs can take different numbers of input parameters and produce different numbers of explored paths.

##### A. Reducing Test Cases with Causality Partitions

Table VI gives the experiment results of our approach and other two peer techniques. We set the number of input parameters of WC and SUM to be 5 and 7, respectively. The columns labeled with *Causal*, *Flow* and *JPF* denote the results obtained from our approach, FlowTest and standard symbolic execution JPF-SE, respectively. The column *Time(s)* gives the usage time of three approaches in second.

It can be observed that the speedup achieved by our approach is significant, ranging from 2.47X to 49.12X compared with JPF-SE, and from 1.05X to 61.40X compared with FlowTest. These results indicate the effectiveness of our approach. It is well known that the cost of symbolic execution is dominated by SMT solver callings. From this perspective, there are three main reasons that can explain the speedup achieved by our approach. The first reason is due to the reduction in the number of explored paths. This can be confirmed by the column labeled with *#Explored Paths*. The number of *#Explored Paths* is proportional to the number of SMT solver callings that are satisfiable, and our approach reduces the number of explored paths by 46.28% to 98.84% and by 49.37% to 98.86% compared with JPF-SE and FlowTest, respectively. The second reason is due to fewer numbers of SMT solver callings that are unsatisfiable. This can be confirmed by the column *#Infeasible Paths*. Our approach reduces the number of *#Infeasible Paths* by 0 to 98.42% and by 0 to 98.43% compared with JPF-SE and FlowTest, respectively. In this paper, if the path conditions  $\xi_i$  are unsatisfiable, any path conditions  $\xi_j$  prefixed by  $\xi_i$  are not counted in *#Infeasible Paths* because  $\xi_j$  cannot be generated by our approach and other two peer techniques. The third reason is that the average number of constraints in the path conditions is fewer in our approach. Although not guaranteed, in general SMT solver is more efficient with fewer numbers of constraints. This can be explained by the fact that the path conditions in our approach must have causal relationships, while in JPF-SE and FlowTest

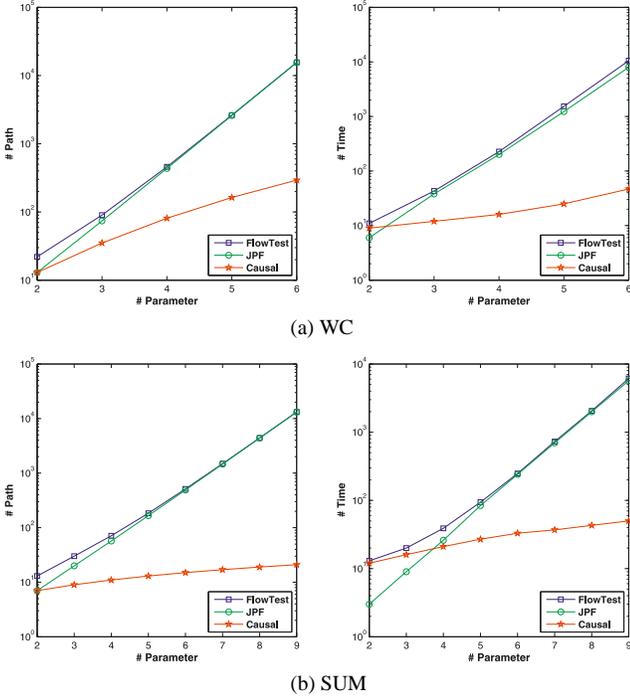


Figure 2. Growth of the explored paths and usage time on WC and SUM.

all path conditions encountered during a path exploration have to be considered. This can be confirmed by the data presented in the column labeled with *#Avg.PC* that shows the average number of constraints in the path conditions.

### B. Growth Rates of Explored Path Size and Usage Time

Figure 2 depicts the number of explored paths and usage time over different numbers of input parameters that range from 2 to 6 for WC and from 2 to 9 for SUM.

It can be observed that in JPF-SE both the number of explored paths and usage time grow exponentially with the number of input parameters, and there is no observable improvement in FlowTest. In both programs, FlowTest explores the paths without reduction because the input parameters of two programs are mutually dependent. On the other hand, the growth rates are much slower in our approach. In Figure 2(a), we can see that the number of explored paths and usage time in our approach still grow exponentially with the number of input parameters, but the exponents in our approach are smaller than other two peer techniques. In Figure 2(b), the number of explored paths and usage time in our approach become to grow in the order of polynomial magnitude and it greatly slows the growth rates. In program SUM, the program statements that handle input parameters are independent. Therefore, with additional input parameters our approach only additionally explores the program paths that are relevant to the added input parameters, without considering the program paths that are relevant to the previous input parameters. In addition, it can also be observed that when the number of explored paths is trivial, our approach uses a little longer time than JPF-SE. This is due to the overhead of the causal chain computation. Based on the growth rates of the

explored path size and usage time, our approach has potential to offer orders of magnitude improvement over standard symbolic execution.

## V. CONCLUSION AND FUTURE WORK

In this paper, we present a novel test case generation technique that selectively explores program paths based on program causality, which can generally alleviate the path explosion problem of symbolic execution. In addition, our reduction is sound, which means our approach can cover all feasible causal chains of a program. This enables us to obtain a concise test suite. We also empirically compare the efficiency of our approach with standard symbolic execution JPF-SE and another improved symbolic execution FlowTest.

Moreover, our approach can be extended for regression testing if we only focus on the program dependencies relevant to the changed statements. We will address this problem in the future work.

## REFERENCES

- [1] Pasareanu, C. and N. Rungta. Symbolic PathFinder: symbolic execution of Java bytecode. ASE, 2010. Antwerp, Belgium: ACM.
- [2] Cadar, C., D. Dunbar, and D. Engler. KLEE: unassisted and automatic generation of high-coverage tests for complex systems programs. OSDI, 2008. San Diego, California: USENIX Association.
- [3] Godefroid, P., N. Klarlund, and K. Sen. DART: directed automated random testing. PLDI, 2005. Chicago, IL, USA: ACM.
- [4] Anand, S., and N. Tillmann. Demand-driven compositional symbolic execution. TACAS, 2008. Budapest, Hungary: Springer-Verlag.
- [5] Anand, S, Pasareanu, C, Visser, W. Symbolic execution with abstraction. Int. J. Softw. Tools Technol. Transf., 2009. 11(1): p. 53-67.
- [6] Siddiqui, J.H. and S. Khurshid. Scaling symbolic execution using ranged analysis. OOPSLA, 2012. Tucson, Arizona, USA: ACM.
- [7] Staats, M. and C. P. Parallel symbolic execution for structural test generation. ISSTA, 2010. Trento, Italy: ACM.
- [8] Majumdar, R. and R.-G. Xu. Reducing Test Inputs Using Information Partitions. CAV, 2009. Grenoble, France: Springer-Verlag.
- [9] Qi, D., H.D.T. Nguyen, and A. Roychoudhury. Path exploration based on symbolic output. FSE, 2011. Szeged, Hungary: ACM.
- [10] Ranganath, V.P., et al., A new foundation for control dependence and slicing for modern program structures. ACM Trans. Program. Lang. Syst., 2007. 29(5): p. 27.
- [11] Reps, T., S. Horwitz, and M. Sagiv. Precise interprocedural dataflow analysis via graph reachability. POPL, 1995, ACM: San Francisco, California, USA. p. 49-61.
- [12] Boonstoppel, P., C. Cadar, and D. Engler. RWset: attacking path explosion in constraint-based test generation. TACAS, 2008. Budapest, Hungary: Springer-Verlag.
- [13] Santelices and Harrold. Exploiting program dependencies for scalable multiple-path symbolic execution. ISSTA, 2010. Trento, Italy: ACM.
- [14] Taneja et al. eXpress: guided path exploration for efficient regression test generation. ISSTA, 2011. Toronto, Ontario, Canada: ACM.
- [15] Person S., G. Yang, Rungta, N. Directed incremental symbolic execution. PLDI, 2011. San Jose, California, USA: ACM.
- [16] Moura, L.D., Bjorn N. Z3: an efficient SMT solver. TACAS, 2008. Budapest, Hungary: Springer-Verlag.
- [17] Hutchins, M., Foster, H. Goradia, T, Ostrand, T. Experiments of the effectiveness of dataflow- and controlflow-based test adequacy criteria. ICSE, 1994. Sorrento, Italy: IEEE Computer Society Press.
- [18] Bohme, M., Oliveira, B, Rochoudhur, A. Partition-based regression verification. ICSE, 2013. San Francisco, CA, USA: IEEE Press.

# How to Enhance the Creativity of Software Developers: A Systematic Literature Review

Reshma Hegde  
Microsoft Dynamics AX  
Microsoft Corporation  
Fargo, USA  
rehegde@microsoft.com

Gursimran Walia  
Computer Science Department  
North Dakota State University  
Fargo, USA  
gursimran.walia@ndsu.edu

**Abstract**—Success during software development depends on the creativity of software engineers. Knowledge plays a very important role in enhancing the creativity of software developers. Knowledge is available in different forms like repository knowledge (experiences of past projects) and community knowledge (gained through communication among software engineers). To help software engineers be more creative and successful, a systematic literature review was undertaken to find how knowledge influences creativity. The systematic literature review reports the various knowledge sources and how these can be accessed by developers to be more creative, and the methods used to access the knowledge sources.

## I. INTRODUCTION

Knowledge plays a significant role in the creativity of software engineers and creativity of software engineers defines the success of projects. Creativity refers to “*the ability to produce new and original ideas and things*” [1]. Graham Wallas described the stages of the creative process as, the *preparation stage*; where data is collected to solve a problem, the *incubation stage*; where the brain works unconsciously on the problem with existing knowledge, the *illumination stage*; where ideas start emerging and links are established between known facts, and the *verification stage*; where ideas are worked on so that it can be communicated to others [8]. The creative process is described as rapid idea generation phase followed by periods of incubation and reflection before these ideas are refined or new ideas are created [16].

Software development is conceptually a complex, knowledge intensive and cognitive activity. Effective software development relies on the knowledge collaboration and on the creativity of software developers [9]. Creative thinking requires the ability to integrate *internal* (stored in head) and *external* knowledge (stored across different artifacts and developers) for performing a task. Creative thinkers search for new ideas by manipulating existing knowledge to see different problems, opportunities and solutions [4]. On that, previous researchers have conducted studies to analyze the role of various forms of knowledge (*prior knowledge, analogies* etc.) on the creativity during software development. This includes case studies [6, 13], experiments [12, 7], surveys and interviews [10] with students and professionals from various domains (not just limited to SE field). Their research found that knowledge plays a positive role and can help professionals to be more creative in their work. Knowledge in different forms helps software engineers to come up with as many ideas as possible to solve a problem. The results also show that prior knowledge helps engineers get new ideas/inspiration for the current problem [5].

Further, knowledge in the form of *examples* helps engineers improve their understanding of the problem, and to add innovation to their solution or to already exist ones [10]. Similarly, knowledge in the form of *analogies* helps engineers see the problem from different viewpoints and generate more ideas to solve their problem. The results from an empirical study showed that designers came up with more ideas with the help of analogies (as compared to without analogies) [7]. Similarly, sharing of knowledge among peer engineers makes the engineers see a problem in a different perspective which triggers creative ideas. In an experiment, software engineering and social workers were provided an existing object and tools to share their ideas and connect with other peers. The results showed that subjects who shared their knowledge generated more ideas and developed an innovative product [17].

While the above results provide some evidence that using “*knowledge*” can improve creativity, a subset of previous research [5,7,10,17] narrowed down the focus on different forms of knowledge (e.g., prior knowledge or analogies) that had an impact on the creativity of software engineers. It was found that information was scattered among different papers and each provided information on only one form of knowledge, how that knowledge type can be accessed to enhance creativity, and what problems are faced [5]. Hence, it was imperative to collect this scattered information and organize them in a single study to be able to better understand the effect of knowledge (in different forms) on creativity of software engineers. To realize this, a systematic literature review was conducted.

A *systematic literature review* is a formalized, repeatable process in which researchers systematically search a body of literature to document the state of knowledge on a particular subject. This approach is more commonly used in medical field to document high-level conclusions drawn from a series of detailed studies [3]. To be effective, a systematic review was driven by an overall goal : *Identify all the sources and forms of knowledge that helps improve creativity of software engineers, the ways that the knowledge is accessed, and their limitations.*

## II. MOTIVATION

Our software companies (like Microsoft) want employees to be creative and come up with new innovations. Creativity among software engineers has become very important for the long-term success of the company. In order to understand what triggers or enhances creativity among software engineers, an ad hoc literature review identified that among several factors (i.e., intellectual abilities, knowledge, thinking, personality,

motivation and environment) **knowledge** has the most impact on the creativity among software engineers [13].

Software engineering is knowledge intensive and collaborative task. Its success depends on the collaborative knowledge and skills of the software engineers involved in the process [9]. Software engineering knowledge is scattered across different resources (e.g., artifacts, code, documents, peers, lessons learned etc.) and continually flows through the entire software development ecosystem. From this perspective, knowledge plays an important role in triggering the creativity of software engineers. The main motivation is to help software engineers understand the importance of knowledge and properly utilize the knowledge to have a creative edge. This work aims to help software engineers understand the benefits of different forms of knowledge (e.g., prior knowledge, examples and analogies) and include them in their day to day software development activities, and to help motivate the software developers to store and share their experiences with their peers and generate creative ideas to solve their problems.

### III. SYSTEMATIC LITERATURE REVIEW

This section describes the process used for performing a systematic literature review of, *role played by knowledge in enhancing the creativity of software professionals*. This section provides a high-level description of the review protocol. More details of the review protocol can be referred in [19].

The systematic review is based on guidelines established by Kitchenham et al. [2, 3]. The purpose of performing a systematic literature review is similar to that of performing any scientific experiment. Procedures are established, followed, and reported on so that other researchers are capable of replicating the work. Following a systematic review process also provides a high degree of control over the type and quality of reference works that will be included in the review and helps to provide support for the conclusions of the literature review. In accordance with the guidelines established by Kitchenham [2-3], the review protocol specified the research questions to be addressed, established a list of databases, conference proceedings, journals, etc. from which primary sources were selected, and established criteria for including sources, extracting data from the included studies, and evaluating their quality. The subsequent steps closely mirror those of any other experiment in that the protocol is executed; the results are analyzed to address the research questions, and are discussed.

**Research Questions:** A high-level research question (*What is the role played by knowledge in creativity of professionals in software development organization?*) was decomposed into three more specific RQ's and related sub-questions. A list of these RQ's is available in Table I. **RQ#1** gathers and analyzes the evidence from the literature to investigate the extent to which the knowledge has an effect on creativity. **RQ#2** gathers and analyzes different ways in which knowledge is accessed

TABLE I. RESEARCH QUESTIONS AND MOTIVATION

Research Question # and Description
RQ1. Is there any evidence that knowledge influences creativity in Software Engineering?
RQ2. Is there any evidence that the Knowledge access methods identified in literature enhances the creativity of software professionals?
RQ3. What are the ways to improve the current knowledge access methods so that it can support the creative thinking process of software professional?

and utilized to be creative. It also gathers all the shortcomings of the knowledge access methods. **RQ#3**, a meta-question gathers additional evidence along with the information from RQ's 1 and 2 to suggest improvements of the knowledge access methods to support creativity in software development.

**Source Selection and Search:** Initially, an ad hoc review was performed in order to assist with the development of search strings and to provide a list of potential conference proceedings and journals to be manually searched. Selection criteria were developed to establish a list of initial databases to be searched and conference proceedings or journals to be searched manually. References from primary sources were also included if they were relevant. The source list which was searched included these Databases: *ACM Digital Library, IEEE Explorer, APA PsycINFO, ScienceDirect*.

**Study Execution:** After executing search strings on all databases and manually searching through selected sources, 6696 papers were found, that were initially screen for inclusion based on their *title* and *abstract*. After this, 43 papers remained and were read in their entirety to evaluate their *quality* and *applicability*. After reading all of these papers, 16 remained.

**Study Inclusion and Exclusion Criteria:** The papers chosen for inclusion in the literature review are all based on empirical research in order to provide greater confidence that the results are not anecdotal. A set of inclusion /exclusion rules were created so that only the most relevant studies were considered for the systematic review (as shown in Table II).

**Quality Assessment:** Quality assessment was performed to assess the study design, bias, validity, and generalizability of results for the final list of papers. Each paper was evaluated using a quality assessment checklist in accordance with the guidelines published by Kitchenham et al. The results of the quality assessment were good and can be found in [19].

**Data Extraction and Synthesis:** Once all of the papers had undergone quality assessment, data extraction was performed on all papers. A data extraction form (see Table III) was developed to ensure consistent extraction across all papers. This data \was synthesized to answer the RQ's (in Table I).

**Validity Threats:** Most of the sources found during the review focused on the relationship between knowledge and creativity in the requirements and design stages of software

TABLE II. INCLUSION AND EXCLUSION CRITERIA

Inclusion Criteria	Exclusion Criteria
Publications that are related to Knowledge and creativity in software engineering. Publications that reported the results on creativity in Software Engineering. Publications that contained empirical results. Publications that are related to creativity and knowledge which were non software engineering, but could be applied to software engineering.	Publications that are not in English. Publications that were short or mini reports. Publications that were not related to any of our research questions. Publications that contain unclear or ambiguous results.

TABLE III. DATA EXTRACTION FORM

Identifier	Description
<b>Main RQ's</b>	The research question(s) that the paper answers.
<b>Motivating study</b>	The previous study from which the study was inspired if any.
<b>Type of study</b>	The type of study (e.g., survey, experiment etc).
<b>Findings</b>	The main findings of the paper
<b>Validity threats</b>	Threats in the findings or study techniques.
<b>Why paper is interesting?</b>	Highlights of the paper/most interesting finding or technique.
<b>Knowledge management</b>	The knowledge management or access technique discussed in the paper.
<b>Improvement</b>	Improvements for the current study if applicable.
<b>Empirical evidence</b>	The empirical evidence for the question(s).
<b>Relation with creativity</b>	Any information that indicates that knowledge improves or influences creativity.

engineering. There was no literature that focused on the creativity during the testing stage of software development. We anticipate that this relationship (Knowledge vs. Creativity) would hold true for other stages of software development, due to the nature of testing as a knowledge intensive activity as well. However, there is no empirical evidence to underscore this claim. Also, the sources which reported the effect of knowledge on the creativity during the design included designers that were not limited to SE background. However, the results were consistent across SE and non-SE studies.

#### IV. REPORTING THE REVIEW

This section reports the results organized by each RQ using the information extracted and synthesized from all the papers.

##### A. Question 1: Does knowledge influences creativity?

A review of the literature indicates that knowledge impacts creativity during software construction. Knowledge (available in various forms) helps software engineers come up with a larger number of solutions to their problems (some of which may be original). While most of the empirical studies focused on the relationship between design knowledge and the creativity, the relationship and the resulting effects can be applied to any development stage since each stage involves knowledge transfer of some sort. The review also uncovered studies from the *Psychology* and *Human Cognition* whose findings and results were generalized to the SE domain.

The review identified many forms of knowledge that can influence the creativity of software professionals. These include: a) **Prior knowledge**– useful information (ideas that could be extracted from design documents, magazines, sketches, notes, lessons learnt etc.) from previous projects. The previous projects could be one's own projects or other's projects; b) **Analogies**– are the entities which look similar to the entity under construction. This could belong from the same domain or other domains; c) **Dynamic knowledge** in tools – pertains to the knowledge that is embedded into the tool and is available to the engineers at any time or when user demands. Details of these knowledge forms and the empirical results on their effect on an individual developers' creativity follows:

Cognitive Psychology has often cited the use of the **prior knowledge** as an important ingredient to create novel solutions. Humans are able to use the prior information (e.g., examples of other's work) to learn produce new ideas and solutions. Similar

to the psychology field, prior knowledge helps software engineers to generate more (and new) ideas or be creative. Specifically, software engineers during the early phase of design heavily depend on the prior knowledge (form their own and their peer's prior work) in order to produce new ideas. Prior knowledge represents conceptual ideas, lessons and representations captured or collected when solving a problem Bailey et al., reported the results from interview with 14 design engineers and an online survey with 28 additional participants that showed that the reuse of prior knowledge is highly valued and an important step during early phases of design [5]. In early phases, people are still looking for directions rather than actual solutions. They provide inspiration to new and creative ideas. Prior knowledge can be well defined solutions to recurring problems, case based solutions, design decisions and reasons behind those decisions, design history for a particular item. The results showed that designers tended to access previous work or project to get inspiration. Results showed that designers stored artifacts like notes and sketches from their current project as they were sure that they will need them in their future projects; however designers mostly prefer to analyze other designer's work more than their own past work to gain new ideas and inspiring solutions [5].

**Examples** (of prior knowledge) are helpful in enhancing the creativity of software engineers. Examples are any kind of material, product, prototype or digital artifact that contributes directly or indirectly to the design. Examples can come from the designer's own work or external sources like other designer's work, web, blogs, magazines etc. Herring et al. conducted an interview with designers to understand the use of examples in creative design. The results showed that examples helped designers in idea generation phase. Examples also helped the designers to understand the problem better and helped them compare similar products to see the novelty in their idea. During the evaluation phase, examples were used as references to see how their design has evolved from where they started [10]. Therefore, prior knowledge plays an important role in providing a starting point and inspiration for new ideas.

**Analogies** are another form of knowledge which helped designers to come up with creative and innovative solutions by analyzing the design solution from a different domain. Analogies help compare two similar entities, find similar feature and integrate them. This can lead to creative solutions. As pointed out by Johan Hoorn, people come up with creative solutions when they combine similar entities together, compare them and look closely to their intersecting features and integrate them. He explains creative process as entities being associated, selected, integrated and adapted until optimal solution is found [11].

Nathalie Bonnard reported the results from a series of experiments to show that creativity during the design phase can be improved with the help of analogies [7]. In one of the experiments, subjects were asked to design a chair for cyber café and were provided with sources to make analogies. Two analogies provided belonged to the same domain and were nomadic stool and rocking chair. The other two were from different domain and were canoe and logotype. The results showed that the group that used analogies was able to draw features from many more analogies than the unguided group,

who did not have sources to make analogies. The guided group came up with significantly more number of sources and more ideas to generate [7]. In another experiment, professional designers and students were asked to design a renewed torch. They were asked to find a new gesture to use the torch and move away from the traditional switch. Very minimal information about the problem was given to the participants, but additional information or analogies were provided upon request. The results showed that professional designers asked more questions to understand the problem better and visualize the problem from various perspectives. Professional designers applied these various viewpoints into their problem solving at an early stage. Overall, these results showed that analogies help designers to extract more details about the problem in hand and in coming up with more ideas to solve the given problem [7].

The engineer's knowledge base consists mainly of scientific principles, formulae and rules. The scientific principles are linked to the real world, where creative products will be used. This linking or applying the principles to the real world is done by the engineer. If we want to enhance the creativity and creative output of an engineer, we should be able to build the capability of applying the scientific principles to the real problems. The information technology (IT) tools can gather this knowledge and devise rules and make it available to the engineers during their work of solving problems. This is called the **dynamic knowledge**, which is available in the tool and can help designers. Example of tools like CAD helps designers analyze and design solutions. It helps in rapid prototyping. Simulation tools help engineers run their design in its environment and predict how it behaves, identify any shortcomings and fix them. This helps the engineer in trying different things and be creative. Tools with knowledge built into them can help engineers be creative [14].

#### B. Question 2: Knowledge access methods vs. Creativity

The review identified two major categories of knowledge access that can influence creativity and are listed below.

**Repository** – Repository is a centralized place where all prior knowledge is stored. This knowledge can be pulled by users as needed or can be provided to the users dynamically with the help of tools, as they work on problems. In this category, searching relevant information is very challenging as there is huge amount of data. The search mechanism used in the repository is not efficient at finding the relevant information inhibiting the creativity of software engineers.

**Collaboration** – Software engineers exchange and share their ideas and information with other engineers. Knowledge and idea sharing helps software engineers to explore areas which they are otherwise unaware of, and helps them to see different viewpoints, opens a new channel of thought, and provides feedback to improve on one's idea and knowledge. The main problem during the collaboration is finding the right people and tools to collaborate and share knowledge with.

Knowledge sharing through collaboration was the most commonly reported method that software engineers used to access the knowledge which in turn triggers creative ideas and their development. Ardiaz et al. performed a study with subjects using the tools named *Wiki ideas* and *Creativity*

*Connector* for knowledge sharing and collaboration. These tools allow large number of users to participate in brainstorming sessions to share their knowledge which can enhance creativity. Fifty participating subjects in the study were divided into two equal groups, one composed of people from similar background and the other group had people from different backgrounds based on their degrees. The goal for the groups was to create an innovative product or service from an existing object. The results showed that these tools helped users to generate a large number of ideas by sharing knowledge and through the feedback provided by other users. It also provided user interface to enable users to see the ideas online and provide feedback to them. The results showed that the heterogeneous group produced greater number of ideas as compared to homogeneous group [17]. Collaboration between peers (irrespective of the peers' background) can help software engineers to be more creative and generate a larger number of ideas. Therefore, it is important for software engineers to collaborate with other knowledgeable peers and exchange feedback and suggestions.

YunWen Ye [9] explains that knowledge collaboration in software development happens in two axes. In *technological axis*, tools like books, manuals and repositories are used to enhance the insufficient knowledge in the heads of software developers. In *social axis*, collaboration happens with other knowledgeable peers to share their proposed ideas and improve their solutions. Engineers collaborate to solve a problem by filling in the missing knowledge in each other's head. When other knowledgeable peers review the ideas from different perspectives, creativity (multiple ideas) can emerge or evolve.

Computers play two roles to support knowledge collaboration: a) they act like repositories and provide useful information to the developers on their work, and b) they provide a platform for software developers to share new ideas and solutions [9]. Kao et al. conducted an experiment to show that people came up with new ideas and understood concepts better when they share their ideas and concepts with others, as opposed to working alone. The experiment was conducted with thirty two subjects who initially formed their concept maps (which consisted of the concepts and links between different concepts that they had learnt in the class). The concept map represented how well they understood the concepts in the class. Subjects exchanged these maps with other students, reviewed them and noted down any information (or creative cross links) that was missing in their maps. Next, the subjects reworked their concept maps by incorporating the information gained from other concept maps. The results showed that the revised maps were better than the original ones. More importantly, when individuals share their concepts and ideas with others, it gives them a new perception to concepts and helps them discover ideas which they could not find on their own [12]. Based on these results, during collaboration software engineers generate more creative ideas compared to working alone.

Tarik Baykara also reports that among several factors that can drive creativity among the people in a software organization (e.g., intellectual abilities, knowledge, thinking styles, personality, and motivation), the ability to provide a **dynamic work environment** for the engineers to share their ideas and knowledge is most relevant factor. This was

concluded by performing qualitative evaluation and through assessment of creativity observed during all the project phases on ten completed research and development projects in Turkish's scientific and technological council. The results from the case study showed that there is a strong correlation between knowledge sharing and the creativity in all project types. It is important for an organization to promote knowledge sharing between their employees to allow them be more creative [13]. Similarly, Edmonds et. al., reports the results from a retrospective study on bicycle design which showed that for a designer to be creative, he should have access to diverse and dynamic knowledge. Based on the results from the study, it was concluded that **dynamic knowledge** should be made available to the designer depending on the problem.

A system like KSS (Knowledge support system) has been proposed by several authors that can provide the knowledge or information to the designers during the design process. The KSS includes the design history in the form of predecessor designs, design rules and knowledge editors and provides access to all the information and expertise. This system was proposed after concluding that embedding knowledge into the tools can provide knowledge dynamically to the user [6].

Similar to the above study, Bales et al. conducted an observational study to see how designers organize tangible information in their work space and how they utilize this information in their work. The results from the study showed that the **prior knowledge** and the **domain knowledge** are keys to creative and successful design. They also observed the designers individual and collaborative work spaces and showed that the tangible information is moved between these workspaces as design evolves and refines. Based on these results, the authors concluded that, for any tool to support creativity, it has to provide multiple workspaces and ability to move the information between these work spaces to share their ideas with others and get feedback to refine their solutions [15].

On similar lines Joshua et al., conducted a study to see how software tools can help the designers in being creative. The tool used in the study provided personal and collaborative workspaces for sharing ideas and work. People were asked to work on individual ideas, and then move it to the collaboration space and work collaboratively on the idea. The results of the study showed that the tool helped design multiple solutions for the problem and also work on them in parallel. The tool also helped the teams share their ideas with others, get feedback and work in collaboration which led to creative solutions [16].

**Limitations:** Based on these results, there is ample evidence to show that the knowledge access and sharing methods help improve the creativity of software engineers, especially during the software design. However, there were few limitations with the knowledge access method that inhibits the creative power of software engineers. For example, when using the repositories, searching for prior knowledge is cumbersome due to plethora of information stored in them. It is challenging to finding the right information efficiently. Software engineers also expect to have a visual search to help them locate the right information easily. Knowledge users also felt that the information was stored in an inconsistent manner and wanted some kind of consistency so that they can easily make their

way to the right information they are looking for [5]. Similarly, during the knowledge collaboration, the most difficult problem is to find the right people to collaborate with. It is reported that people are not always aware of whom to contact to get help when they are in problem or get feedback from. At times when people find the right people to collaborate with, others may be busy with their work and would not completely participate in collaboration or they might deny participating [9]. Therefore, it is important for an organization to create work environment for the engineers to be able to share their ideas and knowledge and collaborate in order to support creative software development.

### C. Question 3: Improving the creative thinking process

Multiple studies suggest that the knowledge access methods need to be improved for supporting creativity [5, 4, 9, and 18]. Multiple studies reported knowledge stored in repositories as one of the main sources of knowledge used by software engineers to solve problems. While prior knowledge gained from repositories is very useful, it is not being used effectively as searching is very difficult and in few occasions important information from the data is missing from these repositories. The search mechanisms used in existing repositories are not very powerful thereby making searching for the right things extremely difficult. This is the reason why software engineers seldom use prior knowledge stored in repositories.

One way to improve this could be to make search more powerful with introducing visual search. In this kind of search, a person can visually see the data through which he is searching, the user can modify the search based on their needs, and the user can see how the data is organized so that he/she can search the data in an effective way. Once the search provides the right information in an efficient manner, software engineers will start using the repositories to enhance their creative thought process. Regarding the incompleteness of the information stored in the repositories, it is important to collect the story behind the knowledge being stored. For example when a design was chosen over alternate designs, storing the reason as to why it was chosen over others is a good approach. This would help the person looking at this knowledge better understand the original thought behind the decision and help relate it to his or her current problem [5].

Based on the results reported in IV.A, **analogies** are an important form of knowledge which triggered creativity among software engineers. Maiden et al., [4] conducted a workshop to show that though using analogies during the requirements engineering generated more creative ideas, it had inherent limitations. The workshop targeted collecting requirements for a future air traffic management system for managing departures from major European airports. Twenty one professionals from the departure management and scheduling departments in the UK and France attended the workshop. One of the main lessons learned from this workshop was the realization that in order to support the use of analogies during requirement stage, people need to be trained on how to use analogies and to be able to extract useful information from analogies to transfer knowledge from one domain to generate new ideas and use it in the targeted domain. Therefore, usage of analogies should be supported by providing relevant trainings so that people know how to extract knowledge from analogies and generate creative

ideas. While this workshop was focused on the requirements phase, the results are applicable at other phases.

Based on the results reported in IV.B, sharing knowledge or collaboration is one of the common ways to promote creativity. Andrawina [18], reports the results from a survey that organizational factors like top management's support to knowledge sharing and technology factors like computer network and electronic bulletin boards, discussion groups to collaborate and share knowledge influence positively and encourage people to share knowledge and apply them to be innovative. The survey recommended that organizations should have management who appreciates and rewards knowledge sharing behaviors among their employees and provide help required for knowledge collaboration. This would help organization and its people be innovative and creative [18].

Additionally, the results reported that the two major knowledge access methods were repository and collaboration. The data in the repository does not contain dynamic contextual data whereas the collaboration gives access to dynamic and contextual knowledge. The best approach would be to integrate both types of access methods by providing access to both the repository knowledge and the contextual knowledge. Software tools that can combine both these approaches would act as single point of contact for a software developer to search for information when he/she wants feedback or help with his or her problem. Such tools would increase the knowledge access and in turn help software engineers be more creative [9].

## V. DISCUSSION OF FINDINGS AND FUTURE WORK

The knowledge types reported in literature that impacts the creativity of software engineers were broadly classified as: *prior knowledge*, *analogies*, and *dynamic knowledge*. Prior knowledge provided useful information about previous projects and helped software engineers in the initial stages of idea generation. They provided a starting point or motivation to start with ideas to solve the problem. Analogies helped software engineers link two objects from same or different domain and create new ideas. Tools provided knowledge when required by the software engineer and help software engineers concentrate on the original problem. Knowledge is accessed using different ways. The most common knowledge access techniques found during the review was searching for prior knowledge in repositories, internet, wikis, books and magazines. Another technique was collaboration, where software engineers shared their ideas and thoughts with others. They build on each other's ideas and provide constructive feedback. They covered each other's knowledge gap as a group.

Knowledge is very useful for software engineers and it makes them more creative and but not all software engineers were utilizing the available knowledge. The reasons for this were that searching for the right information in large repositories was cumbersome and when the information was found relevant information but would be missing in the repositories. Searching huge database and finding the right information was difficult as the search techniques were not very powerful. Gaining new knowledge via collaboration was difficult too as finding the right experts to share the ideas and get inputs was difficult and when right people were found they were busy in their work and had minimal or no time to

collaborate. To improve searching and make it easier the knowledge should be stored in a consistent way, searching techniques should be improved and powerful visual techniques should be added. We should try to combine repository knowledge with collaboration tools so that software engineers can find all the knowledge in one location. Software engineers should be trained on analogy making and utilizing analogies to extract useful information and come up with new ideas. Management of software industries should support and enhance knowledge sharing behavior among employees.

## REFERENCES

1. A. K. Aurum, F. Daneshgar and J. Ward, "Investigating Knowledge Management practices in software development organizations – An Australian experience. Information and Software Technology", Information and Software Technology, Vol. 50, Issue. 6, pp. 511–533, May 2008.
2. B. A. Kitchenham, "Procedures for undertaking systematic reviews", Technical report, Computer Science Department, Keele University, 2004.
3. B. A. Kitchenham and S. Charters, "Guidelines for performing Systematic Literature Reviews in Software Engineering", EBSE Technical Report, Keele University and Durham University Joint Report, July 2007.
4. N. Maiden, S. Manning, S. Robertson and J. Greenwood, "Integrating Creativity Workshops into Structured Requirements Processes", Designing interactive systems, 2004.
5. M. Sharmin, B.P. Bailey, C. Coats and K. Hamilton. "Understanding Knowledge Management Practices for Early Design Activity and Its Implications for Reuse", Conference on Human Factors in Computing Systems, pp. 2367-2376, 2009.
6. L. Candy and E. Edmonds, "Creative design of the Lotus bicycle: implications for knowledge support systems research", Design Studies, Vol. 17, Issue, 1, pp. 71-90, 1996.
7. N. Bonnardel, "Creativity in Design Activities: The Role of Analogies in a Constrained Cognitive Environment", Creativity and Cognition conference, pp. 158-165, 1999.
8. L. Gabora, "Cognitive Mechanisms Underlying the Creative Process", Creativity and Cognition conference, pp. 126-133, 2002.
9. Y. Ye, "Supporting Software Development as Knowledge –Intensive and collaborative activity", Foundations of software engineering conference, pp. 15-22, 2006.
10. S.Herring, C. Chang, J. Krantzler and B. Bailey, "Getting Inspired! Understanding how and why examples are used in creative design Practice", Conference on Human Factors in Computing Systems, pp. 87-96, 2009.
11. J.Hoorb, "A Model for Information Technologies that can be creative", Creativity and Cognition conference, pp. 186-191, 2002.
12. G. Kao, S. Lin and C. Sun, "Breaking concept boundaries to enhance creative potential: Using integrated concept maps for conceptual self-awareness", Computers & Education, Vol. 51, Issue. 4, pp. 1718-1728, 2008.
13. T. Baykara, "Dynamics of 'Technological creativity' as a decision in knowledge creation process", Technology management for the global future, Vol. 2, pp. 951-956, July 2006.
14. T.Kappel, "Creativity in Design: The contribution of information technology", Engineering Management, 1999.
15. C. J. Bales and E. Y. Do, "Managing information in a creative environment", Creativity and Cognition conference, pp. 353-354, 2009.
16. J. Hailpern, E. Hinterbichler, C. Leppert, D. Cook and B. Bailey, "TEAM STORM: Demonstrating an interaction model for working with multiple ideas during creative group work", Creativity and Cognition conference, pp. 193-202, 2007.
17. O. Ardaiz, M. L. Acedo and M. T. Acedo, "Wikiideas and creativity connector: Supporting group ideational creativity", Article No. 31, WikiSym, 2008.
18. L. Andrawina, "Relationship between Knowledge Sharing and Absorptive capacity Moderated by Organizational and Technology Factors: a Conceptual Model", IEEE International Conference of Industrial engineering and engineering management, pp. 1865-1869, December 2009.
19. R. Hegde, G. S. Walia, "Developing a Software Tool to Enhance the Creativity during Software Development using the Results from the Literature Review", Department of Computer Science, North Dakota State University, Fargo, ND, Tech. Rep. NDSU-CS-TR-14-002, Mar. 2014

# Knowledge Transfer between Senior and Novice Software Engineers: A Qualitative Analysis

Davi Viana and Tayana Conte  
USES Research Group  
Universidade Federal do Amazonas  
Amazonas - Brazil  
{davi.viana, tayana}@icomp.ufam.edu.br

Cleidson de Souza  
Instituto Tecnológico Vale and Universidade Federal do  
Pará  
Pará - Brazil  
cleidson.desouza@acm.org

**Abstract**— Software development is a knowledge intensive activity. Software engineers need to gather domain knowledge to be able to successfully deliver a software system. In particular, novice software engineers need to acquire enough knowledge to perform their tasks. This means that knowledge transfer to novice software engineers must be quickly and effectively performed to facilitate the onboarding process. One way to understand the knowledge transfer process is by analyzing the software development context and the involved team members. Such analysis enables the development team to determine key aspects that can influence knowledge acquisition by novice software engineers. This can also allow the identification of possible strategies that minimize the effort employed by the team members during this process. This paper presents a qualitative study about knowledge transfer in a small software organization from the point of view of software practitioners. Our results suggest that software developers have several knowledge sources and novice engineers learn when they observe organizational procedures and when the tasks have detailed guidelines. In addition, we identified that developers carry out most steps of the knowledge creation process defined by Nonaka and Takeuchi. We believe our results can support other software organizations to improve the sharing of knowledge and learning practices.

**Keywords-component:** *Onboarding, Knowledge Transfer, novice software engineers, qualitative analysis.*

## I. INTRODUCTION

In software development activities, dealing with knowledge can be particularly critical, since these activities depend heavily on knowledge and individuals [1, 2, 3]. Novice software engineers must have knowledge about: (1) technologies; (2) software engineering (SE) methods; and, (3) the organization's internal processes [4]. Nevertheless, carrying out activities that involve such amount of knowledge is not trivial [5]. In order to act according to the market's demands, knowledge must be analyzed and shared among all employees [2, 6].

Nonaka et al. [7] classify knowledge into two types: tacit and explicit. Tacit knowledge is based on the person's experience and is hard to formalize. On the other hand, explicit knowledge, also called codified knowledge, is considered to be transferable using a formal and semantic language. Nonaka et al. [7] also state that explicit knowledge can be represented in documents and databases. Furthermore, explicit knowledge must be adequately structured so that other team members are able to use it. Despite the type of knowledge, tacit or explicit, it is necessary to manage this knowledge since knowledge is

considered to be the main competitive asset of a software organization [3].

According to Wang et al. [8], traditional training models often draw on limited resources that are insufficient to support current training efforts. This question can be more severe when organizations need to train and transfer basic organizational knowledge to novice software engineers. Therefore, it is important to understand how novice software development practitioners learn the relevant knowledge on their organizations. Knowledge creation and transfer are inseparable from learning in an organization; both knowledge activities are in fact the consequence of learning [9].

The goal of this paper is to analyze, through a qualitative empirical study, how novice software engineers gain knowledge during the initial activities they perform in a software organization. We also analyze which factors influence such knowledge transfer.

In order to better understand such factors, we performed a qualitative study based on interviews. Such study was performed in a software development organization that encourages knowledge sharing in the work environment. Such sharing was important because some of the senior engineers were going to leave the organization. When talking about novice practitioners we refer to beginners who are learning how to perform assignments in a team. The lack of defined software processes makes knowledge transfer and sharing more complex in the studied organization. Thus, the observation of knowledge transfer practices in this organization is important because we can use our findings to verify the nature of such practices in a real context.

Our results show that software developers in this organization have several knowledge sources including the web, the intranet, training material, the source code, expert team members and the software documentation. However, novice practitioners usually prefer to use informal conversations, which are allowed in the organization's environment. In other words, our results illustrate a real process of knowledge transfer and learning software development activities. The understanding of how software knowledge transfer occurs can support the creation of new strategies for improving novice engineers' capabilities (e.g. stimulus for using the knowledge sources).

The rest of this paper is structured as follows: Section 2 presents related work regarding knowledge transfer in SE.

Section 3 describes the defined research method, the study settings and study context. Then, in Section 4 we discuss our findings, a comparison between our results and findings from literature, and a discussion of our results and the knowledge creation theory defined by Nonaka and Takeuchi [10]. Finally, Section 5 presents our conclusions and future work.

## II. RELATED WORK

Software development is a knowledge intensive activity. For instance, software engineers need to gather domain knowledge to be able to successfully deliver the software system. In addition, novice software engineers need to acquire enough knowledge in order to perform their assignments. Furthermore, not all team members have enough knowledge for performing their role [11]. In this sense, it is necessary to stimulate the transfer of knowledge among software development members. According Joshi et al. [12], knowledge transfer occurs when knowledge is broadcasted from one entity (e.g., an individual) to others.

Many researchers aim to propose and analyze strategies for transfer the all necessary knowledge in a software organization. Aurum et al. [13] performed a qualitative study in two Australian organizations and concluded that team members believe in the utility of sharing knowledge. However, the teams' ability of using Knowledge Management (KM) mechanisms was limited. Furthermore, the authors observed that team members created informal networks, and when supporting such networks with KM, it was necessary to formalize the practices of knowledge sharing. Finally, they identified that the tools, techniques and methodologies used in the organizations were inadequate for an effective knowledge management and transfer.

In order to analyze novice practitioners, Begel and Simon [4] performed a case study at Microsoft. The authors analyzed all aspects of novice practitioners' jobs: coding, debugging, designing, and engaging with their team; and analyzed the types of tasks in which they engaged. In that research, novice practitioners showed high capability in the functional and technical job aspects, but lack of preparation and training in the social and communication aspects they faced at a daily basis [4]. The authors believe that novice practitioners' naivety caused stress and poor productivity during their first months.

Regarding tools for sharing and transferring software knowledge, Amescua et al [14] describes the usage of wikis for storing software engineering best practices. Such research demonstrated that wikis facilitate an effective learning environment. They help users learn an agile process through mechanisms for knowledge sharing, and provide a repository with useful artefacts. Moreover, the authors concluded that novice software engineers could develop software products more independently.

One way to integrate both tacit and explicit knowledge is presented by Sandhawalia and Dalcher [15]. They propose a framework to integrate both types of knowledge and to facilitate the flow of knowledge to address unstructured situations in software projects. They verified that knowledge flow depends on the organizational area.

Although the literature presents numerous researches on knowledge sharing in software engineering, it is important to observe and to analyze how knowledge transfer occurs during project execution with undefined software processes. Such analysis can provide insights to support a successful knowledge transfer and learning by novice software engineers.

## III. RESEARCH METHOD AND STUDY SETTING

One way to investigate the software developers' point of view is to use qualitative methods. According to Seaman [16], the use of qualitative methods allows the researcher to consider human behavior and thoroughly understand the studied object. Therefore, by performing a qualitative study, we hope to obtain a more adequate understanding of the process of knowledge transfer when performed by novice practitioners.

We collected data for this qualitative study using three interviews with a semi-structured questionnaire. First of all, we prepared a questionnaire with open questions about knowledge sharing and the organizational environment. Such questionnaire was applied in the first interview with an expert software engineer. After that, we transcribed and analyze the interview using grounded theory techniques [17]. This analysis helped us adapt the questionnaire to the organizational context and according to the novice software engineers' perspective. Finally, we conducted and analyzed two additional interviews with novice engineers and drew our conclusions. More details about the organization, data collection and analysis methods are presented below.

### A. Study Context

The interviews took place in a software organization called Beta located in Manaus, a city in the North of Brazil. Such organization needs to share knowledge in its different projects in order to increase the new employees' abilities. Beta is a small (10 employees) software development company within the customs control agency and develops software for supporting such agency in the management of goods flow in the Free Economic Zone of Manaus.

Beta adopted agile method in the past. However, it currently has an undefined software process. Team members receive the assignments according to the customs agency needs. Due to the lack of a defined software process, the observation of knowledge transfer practices in this organization is important because we can verify the nature of such practices.

### B. Data Collection

For the interviews, semi-structured questionnaires with open questions were prepared. We prepared two types of questionnaires: one for the expert and another one for novices. Table I shows some of the questions asked to novices.

TABLE I. EXTRACT OF QUESTIONNAIRE FOR NOVICE ENGINEERS.

Knowledge transfer activities identification	How do you identify important knowledge to share in your project?
	How do team members share knowledge in your organization?
Knowledge characterization	When you identify important project knowledge, which information do you try to get?
	How can this knowledge help you during the software project?

Our interviews took place in the interviewees' offices at the Beta organization and the researcher did not rush the interview so that the interviewee could express his/her opinion effectively.

### C. Data Analysis

To analyze the data obtained through the interviews, we used Grounded Theory (GT) techniques [17]. GT is a qualitative research method that uses a set of systematic data collection and analysis procedures to generate, prepare, and validate substantive theories on essentially social phenomena, or on wide social processes. The essence of the GT Method is that the substantive theory emerges from the data, that is, it produces a theory derived from systematically collected and analyzed data. Although the purpose of the GT Method is the construction of substantive theories, its use does not necessarily need to remain restricted only to researchers who have this research goal. According to Strauss and Corbin [17], the researcher may use only some of its procedures to meet his/her research goals. That is our case: we did *not* use all the process suggested by GT, but focused on the coding stages because those were the relevant ones for our study.

The coding process can be split into three stages: open, axial, and selective coding [17]. Open coding involves the breakdown, analysis, comparison, conceptualization, and the categorization of the data. In the early stages of open coding the researcher explores the data with a detailed examination of what one deems as relevant due to the intensive reading of the texts. In the open coding stage the incidents or events are grouped in codes via the incident-incident comparison. In the next step, axial coding, the codes are grouped according to their properties forming concepts that represent categories. These categories are analyzed and subcategories are identified aiming to provide more clarification and specification. Finally, the categories and subcategories are related to each other, and the causal relationships between the categories were determined.

We conducted two periods of data collection and analysis. Each analysis was held after each data collection phase. After each analysis, another researcher verified the identified codes and the created categories in order to audit the coding process. Our informants (i.e., the practitioners within the analyzed organization that we interviewed) are representative since they were: (a) practitioners in charge of the organization's knowledge activities (e.g, sending relevant knowledge through e-mail) and (b) novice engineers within the organization. Therefore, we have confidence in the accuracy and validity of the data we collected, and as a consequence of our results.

## IV. STUDY RESULTS

In this section we discuss the results from our empirical study. Furthermore, we compare the findings from the qualitative analysis with the ones found in literature.

### A. Knowledge transfer at the Beta Organization

As mentioned before, Beta develops software within a customs control agency. Besides the lack of defined software process, there is not a formal way to transfer knowledge within the organization. Thus, the main way to analyze how knowledge transfer happens in practice is by verifying this

process among senior and novice developers. In fact, practitioners have a commitment to transfer the organizational knowledge to novice practitioners because they believe such people will be the future expert members.

Developers at the Beta organization have several knowledge sources available: the Web, the intranet, training material, the source code, expert members and the software documentation. However, novice practitioners usually ignore the software documentation as a knowledge source and prefer informal conversations, which are allowed in the organization's environment. Novice engineers learn when they observe organizational procedures and when the assignments they are supposed to perform have detailed guidelines. Expert team members create such detailed guidelines due the lack of novices' expertise. Thus, both types of knowledge (tacit and explicit) are emphasized at the Beta organization.

Before beginning their assignments, novice practitioners had to participate in diverse trainings focusing on programming languages, database and business context. Such trainings are important for the onboarding process of novice engineers in the organization by providing basic required skills for novice's assignments.

After trainings, when novices start working, they receive new assignments every day. These assignments contain guidelines that they have to follow in order to successfully carry them out. A senior team member sends these assignments and guidelines by e-mail.

*"We do not have a well-defined schedule, (...) I provide novices with what they need to do [their assignments]. Also, I give guidelines about what their tasks are. In the afternoon, after lunch, I talk to them again and I verify if they have any doubts and so on [until they finish the assignment]" – Senior Team Member Interviewee.*

*"I arrive and check my e-mail, because the senior member sends my assignments. So, if there is something I have to do, I start doing it. I am doing the assignments according to the instructions from the guidelines. If I have any doubts, I talk to the senior team member. – Novice Interviewee 2*

During our study, we noticed that e-mails contained a lot of information (knowledge) regarding organizational assignments, such as programming functions that support the custom agencies' routines. Additionally, we noticed that the most detailed knowledge is shared face to face according to the organization's needs.

*"- How do you share relevant knowledge among the project team?"*

*- I think that when demands are appearing. For example, when there is a new function that has to be developed, the senior member says: 'we do this in a certain way, because the framework was defined a long time ago.' The senior member explains how the framework works. So it is more a tacit knowledge." - Novice Interviewee 1*

Another way to obtain knowledge is through observations. Novice developers observe how other team members perform their assignments. When doing so, they perceive how the business process occurs. In this way, they acquire tacit knowledge.

*"- How do you obtain this knowledge?"*

- I obtain knowledge through the observation of the organizational process and by analyzing what each part of the software does. Regarding programming, I observed the source code too.” - Novice Interviewee 2

We noticed that the senior developer provided a complete assignment guideline because he did not have high confidence regarding the novice engineer's performance and expertise. Additionally, we noticed that novice engineers acquired more knowledge (about programming techniques and the business domain) when they performed the assignments by themselves. Although initial information had been provided, in most cases, it was necessary to search for additional knowledge sources that helped the assignments' success.

Novice practitioners pointed out the following knowledge sources: (a) the web; (b) the intranet; (c) books; (d) training material; (c) expert team members; and (d) software source code. The following quotations present examples of such sources:

“Before asking someone, I try to solve my problem. If I do not know how to solve it, I consult some software program that I implemented. Most of the time, we develop, however we do not keep all in mind. Then, if I haven't solved my problem yet, I search the web. Lastly, I ask for help from expert team members.” - Novice Interviewee 1

“(…) The software has some similar functions that I can use as examples. So, I develop based on such examples (…) If I have any doubts, I ask him [senior member] too (…). We have C# books that show some examples” - Novice Interviewee 2

Our informants told us about another knowledge source, the software documentation (e.g., requirements specification, UML diagrams, and software plans). The senior member stated that novice engineers are aware of this documentation. However, it is out of date and novice engineers did not mention its utilization.

“...The software documentation started, if I am not mistaken, in 2008. Then, it is probably out of date...” - Senior Team Member Interviewee.

During the execution of the assignments, novice engineers ask expert team members for help. We verified that all team members are in the same office. Thus, we concluded that the office configuration facilitates face to face interaction. Nevertheless, there are not knowledge transfer records. Then, the senior managers cannot know what knowledge was transferred.

The usage of the source code as a knowledge source is

possible due to the organizational code standard. Such standard recommends the creation of relevant comments in the source code. We notice that code standard is very important, because it helps maintain the knowledge about the software when some team member stops working in Beta, for example.

“Usually, we create comments in the source code. We explain the code step-by-step. The organization has a standard for software development. Anyone who knows such standard can read the source code. This standard facilitates a lot.” – Senior Team Member Interviewee

One novice practitioner stated that some knowledge sources (e.g., books and the Web) are less used as the days pass. The most common source of knowledge is the expert team members. Fig. 1 shows the relations between the codes about knowledge transfer that we identified regarding the results presented above. In order to perform the open and axial coding of the interviews, we used the ATLAS.ti software.

We can see that the “knowledge transfer to novice engineers” category has three main associations: knowledge sources; observations of organizational procedures; detailing of assignments' descriptions for novice practitioners. Each one contributes for the knowledge sharing process.

Regarding evaluation of the knowledge acquired by novice software engineers, the senior member evaluates this informally through the observation of the quality of the releases. This evaluation helps the senior team member increase the complexity of the activities assigned to the novice developers. Such increase will ensure that the novice practitioners acquire new knowledge and gain additional experience regarding the software and the organizational business model. While analyzing these results, we noted some aspects that can influence the sharing of knowledge. Table II presents such aspects and their description.

### B. Comparison Findings

In this section, we discuss our current results compared to previous literature results and prior results described in our own previous work [18]. Some aspects reported by such publications are important and contribute for a better comprehension of our findings. The comparison of our results with these publications aims to verify how literature in the field relates to our emerging data [19]. Regarding our prior investigation, we performed another qualitative study about the life cycle of lessons learned in a software organization [18].

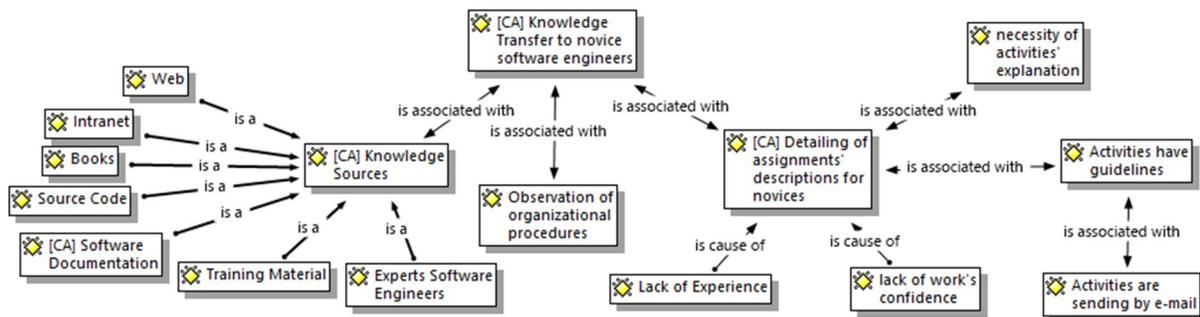


Figure 1. Associations related to Knowledge Transfer.

TABLE II. KNOWLEDGE TRANSFER INFLUENCE ASPECTS

	Item	Observations
Positive Aspects	The usage of code patterns	The code pattern facilitates the understanding regarding software functionalities. The code pattern includes code comments. When practitioners need to know about software, they can consult the source code.
	Initial training	Trainings help novice practitioners gain knowledge regarding business process and technologies used within the organization.
	Learning proactivity	The novice practitioners' behavior helps the experts not to lose time in performing their assignments.
	Organizational Environment	The work office allows the knowledge transfer because all team members are in the same place and the experts are always available for helping novice practitioners.
	Learning by doing	The organization motivates novice practitioners to do the assignments, even if they perform them wrong. It is important for them to perform the assignments by themselves without any prior help.
Negative	Lack of time to update software documentation	Expert engineers do not have time to update the documentation and this can be a reason for the lack of consulting of such knowledge source by novice practitioners.
	Lack of information transfer among experts team members	This aspect can make some important information or knowledge be missed during the knowledge transfer for novice software engineers.

We consider lessons learned as a kind of knowledge artifact, i.e., an artifact that stores knowledge. In this study we observed some similar knowledge sources and practices. To avoid possible misunderstanding, we call "Alpha" the organization in our prior study [18]. Such comparisons give us more confidence on our results.

At Beta, senior members have a great commitment for improving the novice engineers' skills in order to integrate novice practitioners into the software team. Furthermore, we identified that knowledge is mainly transferred by face-to-face interaction. Knowledge transfer occurs during project development and in an informal way. This shows that the knowledge is handled more tacitly. Joshi et al. [12] stated that communication is a primary mechanism for transferring knowledge. The authors state that face to face communication is more significant than virtual communication.

In our previous research, we detected that, at Alpha, the knowledge is handled explicitly. Alpha adopted agile development methods and had an organizational culture to explicit its lessons learned. Nonetheless, they did not mention the use of code as knowledge source. Also, the software documentation in these projects is constantly updated [18]. At Beta, the software documentation was not updated. Yet, the source code had been standardized and it was used as a knowledge source. In future investigations, we plan to verify to what extent the source code and software documentation assist the activities related to knowledge acquisition.

Training is important because they allow one to access important knowledge [20]. Such trainings are important to address some issues, as lack of soft and technical novices' skills [21]. In our findings, novices and senior team members emphasize the importance of training because it presented an overview about the used technologies and the organizational business process. However, training does not cover complete knowledge about all the aspects of the organization.

At Beta, all team members are allocated in the same office. This environment facilitates tacit knowledge sharing. However, team members do not record this knowledge. Tsai and Cheng [22] stated that organizational climate (as the feeling conveyed in a group by the physical layout and the way in which members of the organization interact with each other) contributes positively to sharing both types of knowledge.

The interaction between tacit and explicit knowledge creates the SECI model (Socialization, Externalization, Combination, and Internalization) [10]. Such model supports the creation, exploration and maintenance knowledge. The Socialization is the process of sharing experiences, in which the person shares tacit knowledge directly with others through observations and collaboration. The Externalization converts tacit knowledge into explicit knowledge, thus allowing it to be shared with the project team. In the Combination process new explicit knowledge is created based on a combination of different explicit knowledge. The internalization process makes the conversion of explicit knowledge into implicit and finally maybe tacit knowledge. We compared the knowledge treatment practices in Beta regarding the SECI model [10], we noticed some activities related to each process, according to Figure 2.

The socialization process is highlighted due to the organizations' characteristics presented before, they are in the same office and novices can always talk to senior members. We can point out that an organization that converts tacit knowledge in explicit knowledge through detailed e-mails about work activities and commented source code can help explicit relevant knowledge about the software. Regarding the internalization process, novice practitioners receive the explanation of assignments by e-mail, they can consult similar software source code, intranet, books and Web. Nonaka and Takeuchi [10] state that explicit knowledge is shared in all organization and it is converted into tacit knowledge by individuals. This knowledge conversion is related to "learning

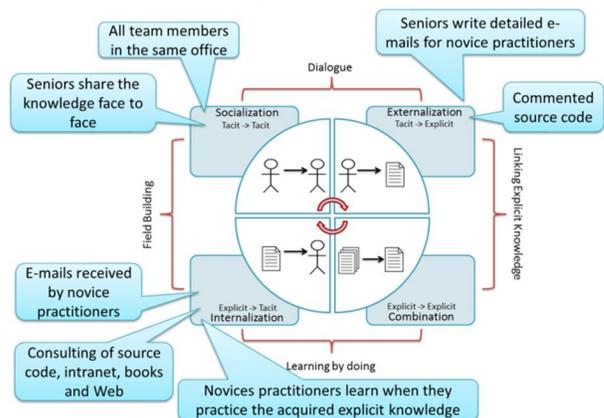


Figure 2. Our findings and SECI Model.

by doing". Thus, we verified that when novice practitioners have finished their assignments, they are learning something new. Finally, our results do not allow us to identify any activities related to Combination process.

#### V. FINAL REMARKS AND FUTURE WORK

In this paper, we discussed knowledge transfer in a small software organization. The lack of usage of a defined software process allowed us to analyze knowledge sharing in the field, i.e, in the context of the Beta organization. A qualitative study was conducted seeking to understand knowledge transfer among senior and novice software engineers. In addition, it was possible to observe tacit and explicit knowledge dissemination.

We noticed that novice practitioners had an initial contact with the necessary knowledge to perform their assignments through training provided by senior engineers. However, our results suggest that they gained more knowledge during the actual software development activities. Also, as mentioned above, a successful knowledge sharing was important for the organization because some of its senior engineers were leaving. The informal development process contributed to an informal way of knowledge transfer. In addition, we observed that Beta adopted parts of the SECI model proposed by Nonaka and Takeuchi [10]. Our results can be used by other software organizations as recommendations for improving their knowledge sharing activities. Additionally, it is necessary to encourage the complete usage of tacit and explicit knowledge in order to create benefits for organizations.

This study has some limitations regarding its organizational context. We carried out interviews with novice and senior team members. We covered knowledge transfer among novices as a whole. However, we know that as a qualitative study, we cannot generalize our results to other companies. Despite that, we believe our study is important because each qualitative study contributes to advancing the state of art in a research area, providing evidence and hypothesis that can be later tested using quantitative methods. In short, each study helps to build a body of knowledge about the sharing of knowledge. Many studies have been performed in academic settings, while little studies have been conducted with novices on corporate contexts. It is important to observe what happens in a real software development context in order to contribute for improving the organizational practices.

We plan to perform additional qualitative data collections aiming to understand more organizations and other contexts. This will allow us to gain a better understanding about knowledge transfer and identify support strategies to organizational learning in software companies.

#### ACKNOWLEDGMENT

The authors would like to thank the team members of Beta organization. We would also like to thank Luis Rivero for his remarks on this paper. Also, we would like to thank the financial support granted by FAPEAM RHTI - Doctorate; Project N.021/2011 - FAPEAM Universal Amazonas; and FAPEAM process PAPE 032/2013

#### REFERENCES

[1] 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, Elsevier, 2008, pp. 1055 - 1068.

[2] K. Schneider. "Experience and Knowledge Management in Software Engineering". Springer, Berlin, 2009

[3] M. Levy and O. Hazzan. "Knowledge Management in Practice: The Case of Agile Software Development", in: Proc. of 2nd Intern. Workshop on Cooperative and Human Aspects on Software Engineering (CHASE) - ICSE Workshop. Vancouver, 2009, pp. 60-65.

[4] A. Begel and B. Simon. Novice software developers, all over again. In: Proceedings of the Fourth international Workshop on Computing Education Research. ACM, 2008. p. 3-14.

[5] M. Gomes, S. Carolyn, V. Basili, and Y. Kim. "A Prototype Experience Management System for a software Consulting Organization", in: 13th Conference on Soft. Eng. and Knowledge Eng. SEKE'01. Buenos Aires, 2001, pp. 29-36.

[6] S. Lee and K. Steward, "A Process-based approach to improving knowledge sharing in software engineering", in: 24th Conference on Soft. Eng. and Knowledge Eng. SEKE'12. San Francisco, pp. 700-705.

[7] I. Nonaka, R. Toyama, and N. Konno, "SECI, Ba and Leadership: A Unified Model of Dynamic Knowledge Creation", Long Range Planning, Elsevier, 2000, pp. 5-34.

[8] M. Wang, H. Jia, V. Sugumaran, W. Ran, and J. Liao, "A Web-Based Learning System for Software Test Professionals", IEEE Transactions on Education, IEEE, 2011, pp.263-272.

[9] K. Mehra and S. Dhawan. "Study of the process of organisational learning in software firms in India". Technovation, Volume 23, Issue 2 pp. 121-129.

[10] I. Nonaka and H. Takeuchi. "The Knowledge-Creating Company". 17th ed. Oxford University Press. 1995.

[11] T. Chau, F. Maurer, and G. Melnik. "Knowledge sharing: agile methods vs. Tayloristic methods" In: IEEE Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises. Linz, 2003. pp. 302-307.

[12] K. Joshi, Sa. Sarker, Su. Sarker, "Knowledge transfer among face-to-face information systems development team members: examining the role of knowledge, source, and relational context" in: Proceedings of the 37th Annual HICSS, 2004, pp.1-11

[13] A. Aurum, F. Daneshgara and J. Warda. "Investigating Knowledge Management practices in software development organisations - An Australian experience". IST Journal, 2008, pp. 511-533.

[14] A. Amescua, L. Bermón, J. García, and M. Sánchez-Segura. "Knowledge repository to improve agile development processes learning". IET Software, pp.434-444.

[15] B. Sandhawalia and D. Dalcher. "Knowledge flows in software projects: An empirical investigation". Knowledge and Process Management, 2010, pp. 205-220.

[16] C. Seaman, "Qualitative Methods", in: Guide to Advanced Empirical Software Engineering, Shull et al. (eds.): Springer, 2008.

[17] A. Strauss and J. Corbin. "Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory". 2 ed. London, SAGE Publications. 1998.

[18] D. Viana, J. Rabelo, T. Conte, A. Vieira, E. Barroso, and M. Dib, "A qualitative study about the life cycle of lessons learned," 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE). San Francisco, 2013, pp.73-76.

[19] R. Hoda, J. Noble, and S. Marshall. "Using grounded theory to study the human aspects of software engineering." In: Proceedings of Human Aspects of Software Engineering (HAoSE '10), Nevada, 2010. p. 5-6.

[20] I. Newman. "Observations on relationships between initial professional education for software engineering and systems engineering-a case study". In: Proceedings of 14th Conference on Software Engineering Education and Training (CSEE&T). North Carolina, 2001. pp. 172-181.

[21] V. Kulkarni, C. Scharff, and O. Gotel. "From Student to Software Engineer in the Indian IT Industry: A Survey of Training". In: Proceedings of 23rd IEEE Conference on Software Engineering Education and Training (CSEE&T). Pittsburgh. 2010, pp.57-64.

[22] M. Tsai and N. Cheng. "Programmer perceptions of knowledge-sharing behavior under social cognitive theory". Journal of Expert Systems with Applications, Volume 37, Issue 12, 2010, pp. 8479-8485.

# A Knowledge & Competencies Checklist for Software Project Management Success

What software project managers need to know and do to be effective

Lawrence Peters SCI Limited, Auburn, Washington, USA, and

Universidad Politecnica de Madrid, Software Engineering Department, Madrid, Spain  
ljpeters42@gmail.com

**Abstract**—By their very nature, software projects require someone to coordinate and direct the efforts of the development team – the software project manager. Although this role has traditionally been viewed as a technical one, there is growing evidence from many sectors that technical skills are only part of what a software project manager needs to be effective. This paper delineates these knowledge areas and notes how they impact the software project.

**Keywords;** software engineering; software project management; essential knowledge areas

## I. INTRODUCTION

For several decades, the software engineering community has attempted to improve its ability to deliver quality results on time and within budget. To that end, several dozen programming methods and techniques have been developed, published and put into use [1] with disappointing results showing only a one source line per programmer per month improvement in productivity over a 30 year period [2]. Given such a poor return on investment, it appears that the “software problem” does not lend itself to technical solutions. Turning our attention away from the knowledge needed to write software leads us to conclude that perhaps the management of software projects may be a beneficial adjunct to the technical activities involved in software development. There is growing evidence the knowledge of how to manage software development may enhance our success rate.

## II. SUCCESSFUL SOFTWARE PROJECTS

A study of successful software projects was recently published [3]. It sought to help software project managers’ chances of success by identifying the characteristics that successful software projects had in common. This was accomplished by analyzing published papers that reported on successful software projects. The research found a common set of factors were present at the start of these projects and other factors were present when the projects had completed. None of these factors were technical in nature but one, “competent software project manager,” lends credence to the proposition that the software project manager plays a critical role in the conduct of successful software projects [4, 5]. This raises the question of just what knowledge and skills must a software project manager possess to be competent thereby increasing their chances of success? While it can be argued that perceptions of “success” vary from stakeholder to stakeholder,

project to project and often changes during the conduct of a project, the authors of the nearly 600 published reports studied felt they had been successful and reported on it in various journals [3]. Keep in mind that today, success goes beyond meeting requirements, schedule and costs.

## III. EFFECTIVE SOFTWARE PROJECT MANAGERS

Some years ago, researchers noted that people from a broad range of backgrounds chose software engineering as their profession [6]. One of their findings was there was something about developing software that their personality was compatible with. In other words, to some extent, they were born to it. Such is not the case for software project managers. Although many have been selected from the ranks of high performing software engineers to lead software projects, this may not be a cost effective practice. Table 1 lists the five primary functions needed to manage software projects [7, 8, 9, 10]. Note the strong presence of interpersonal and communication skills. These five primary functions comprise the “traditional” (control) focused view of project management. While these skills are necessary, they are not sufficient to successfully conduct a software project.

TABLE I. THE FIVE FUNCTIONS OF PROJECT MANAGEMENT

Category	Description
Plan	-Tasks and subtasks -Synchronized with the project schedule -Shows how each event will be achieved -Done in collaboration with the team
Schedule	-List of dates of project events -Supported by the Project Plan -Done in collaboration with all involved
Control	-Continuous monitoring and analysis (e.g. using Earned Value Management) -Apply corrective action to achieve plan
Staff	-Acquire skilled, knowledgeable team with compatible personalities and complementary skills -Resolve intra-team conflicts
Motivate	-Review/evaluate team and individuals -Help develop individuals’ career goals -Engage team to perform well on project

#### IV. SO WHAT KNOWLEDGE AND COMPETENCIES DO SOFTWARE PROJECT MANAGERS NEED TO BE SUCCESSFUL?

The five basic functions (Table I) represent the foundation for software project managers upon which to develop the competencies they need to be successful. The fact that perceptions of success vary during the course of a software project is only one aspect of the complexities a software project manager must deal with. To better understand the competencies needed to complement the basic five functions, it may be useful to identify what knowledge and skills a prospective software project manager must possess in order to, potentially, bring a software project to a successful conclusion. In a recent study related to this issue [11], thousands of advertisements worldwide for project managers were analyzed to identify what knowledge and skills industry sought in project managers. The companies placing the advertisements ranged from multinational conglomerates to specialized contracting firms. In order to determine if there were differences between industries, the advertisements were separated into six industry categories. Significantly, the software industry was the only one that required significant domain specific skills. All the advertisements required communication skills, management skills and other, non-technical knowledge and competencies. From a recently published work [3] which analyzed what factors successful software projects had in common, we know what factors improve the chances a software project will be classified as successful. One significant one was a “competent” software project manager. Some of the knowledge categories critical to being a “competent” software project manager are listed in Table II. Notice that these knowledge areas are “soft” in that they are not related to strongly technical topics. This often presents a problem to strongly technical people who are put into leadership roles when their previous success had been dependent upon solving problems by obtaining the “correct” answer. Once they are in project management, “correct” answers are neither obvious nor readily identifiable and there may be several. This causes many to leave the ranks of management after a relatively short time [12]. Adding to the difficulties software project managers experience is the fact that the path into software project management is rarely clearly spelled out and new managers are rarely trained prior to assuming their new position [12, 13].

#### V. WHAT MAKES A SOFTWARE PROJECT MANAGER COMPETENT?

The space available here is not sufficient to completely answer this question but a few key points can be made:

- First and foremost, the person must want to be a software project manager for the “right” reasons. The fact that the pay may be better, the benefits better and other attractive perquisites come with being a software project manager can cause some people to make the transition into software project management. These represent the “wrong” reasons often resulting in frustration, disillusionment, and a succession of failed projects. Stated simply, the “right” reasons include the acquisition and use of

authority to benefit the company, stakeholders and advance the profession.

- Mastery of the functions in Table I.
- Being prepared for the transition from technician to manager. This includes realizing that the software project manager is more of a coach than a technical contributor. Not realizing this often drives people who make the transition to head back into the technical realm [12].
- Receiving adequate training in not only the five functions of software project management (see Table I) but the other, complementary subjects as well (Table II).

#### VI. CLOSING COMMENTS

Software Engineering has had a relatively short existence as compared with other engineering fields. During the 50+ years it has existed, it has accomplished a great deal. However, the profession is still plagued by a lack of certainty with respect to project costs, delivery dates and the quality of what is delivered. Today, there is a growing recognition that our predisposition with technology as being the remedy for software project uncertainties and failures may be misplaced. We are beginning to recognize the importance of the software project manager’s role as a major determinant of project “success.” Increasing the “soft” knowledge and competencies that software project managers bring to bear on a software project can only improve software engineering’s performance with respect to certainty of delivery dates, content and cost control. Software project management is, primarily, about dealing with people, their idiosyncrasies, cultural preferences, value systems differences, age, work ethic, individual value systems and lifestyles. This makes software project management more like conducting an orchestra than commanding troops

TABLE II. KNOWLEDGE AND COMPETENCIES CRITICAL TO SOFTWARE PROJECT MANAGEMENT SUCCESS [8,9,10,11,12]

<i>Category</i>	<i>Description</i>
Estimating	-Use of various estimation methods -Use of Reference Class Forecasting
Communications	-Written expression (e.g. status reports, Proposals) -Presentation skills
Personnel Management	-Ability to evaluate and direct the actions of individuals and teams -Labor law
Negotiation Collaboration	-Obtain consensus among stakeholders
Cultural Sensitivity	-Awareness & response to the effects of culture(s) represented on the development team -Sensitivity and response to value system differences
Accounting	-Cost allocation, general and administrative and overhead expenses

## REFERENCES

- [1] Rico, D. F., "Short history of software development methods," posted on the web, 2010.
- [2] Jensen, R., "Don't forget about good management," CrossTalk, p. 30, August 2000.
- [3] Ghazi, P., Moreno, A., and Peters, L., "Looking for the Holy Grail of software development," IEEE Software, January/February, 2014, pp. 96-92.
- [4] Boehm, B., Software Engineering Economics, Prentice-Hall, Englewood Cliffs, N.J., 1981.
- [5] Weinberg, G., Quality Software Management, Volume 3: Congruent Action, Dorset House Publishing, New York, NY, pp. 15-16
- [6] Zawacki, R. A. and Couger, J. D., "Evaluating performance appraisal systems for IS personnel," Proceedings of the ACM SIGCPR Conference on Management of Information Systems Personnel, April, 1988, College Park, Maryland, pp. 144-147.
- [7] Peters, L., "Successfully Managing Software Projects," 2-day professional development seminar, Software Consultants International Limited, 2014.
- [8] Peters, L., Getting Results from Software Development Teams, Microsoft Press – Best Practices Series, Redmond, Washington, 2008.
- [9] Peters, L., Managing Software Projects: On the Edge of Chaos (working title), eBook, Software Consultants International Limited, to be released September, 2014.
- [10] Kerzner, H. R., Project Management: A Systems Approach to Planning, Scheduling and Controlling, 11<sup>th</sup> edition, John Wiley & Sons, 2013, New York, N. Y.
- [11] Chipulu, M., Neoh, J.G., Ojiako, U. and Williams, T., "A Multidimensional Analysis of Project Manager Competences," IEEE Transactions on Engineering Management, Q3 2013.
- [12] Tarim, T. B., "Managing technical professionals: When to know to transition from technology manager to individual contributor!," IEEE Engineering Management Review, Vol. 41, No. 4, Fourth Quarter, December, 2013, pp. 3-4.
- [13] Katz, R., "Motivating Technical Professionals Today," IEEE Engineering Management Review, Volume 41, Number 1, March, 2013, pp. 28-38.

# Identifying strategies on god class detection in two controlled experiments

José Amancio M. Santos  
State University of Feira de Santana  
Technology Department  
Bahia, Brazil  
Email: zeamancio@ecomp.ufes.br

Manoel G. de Mendonça  
Fraunhofer Project Center for Software & Systems Eng.  
Federal University of Bahia  
Bahia, Brazil  
Email: mgmendonca@dcc.ufba.br

**Abstract—Context:** “Code smell” is commonly presented as indicative of problems in design of object-oriented systems. However, some empirical studies have presented findings refuting this idea. One of the reasons of the misunderstanding is the low number of studies focused on the role of human on code smell detection. **Objective:** Our aim is to build empirical support to exploration of the human role on code smell detection. Specifically, we investigated strategies adopted by developers on god class detection. God class is one of the most known code smell. **Method:** We performed a controlled experiment and replicated it. We explored the strategies from the participant’s actions logged during the detection of god classes. **Result:** One of our findings was that the observation of coupling is more relevant than the observation of attributes like LOC or complexity and the hierarchical relation among these. We also noted that reading source code is important, even with visual resources enhancing the general comprehension of the software. **Conclusion:** This study contributes to expand the comprehension of the human role on code smell detection through the use of automatic logging. We suggest that this approach brings a complementary perspective of analysis in discussions about the topic.

**Keywords:** Code smell, god class, controlled experiment

## I. INTRODUCTION

Object-Oriented (OO) design challenges are a key aspect in Software Engineering (SE). A poor design can lead to future problems when evolving the code. Important works address the problem from different perspectives. Fowler [1] presented several scenarios where the code may indicate a bad design. They refer to these cases as “code smells”. A code smell could be a sign that one should refactor the code to improve it. Lanza and Marinescu [2] focused on OO metrics to characterize bad design. They refer to bad design as “disharmonies”. These terms are used to define potential design problems. In this paper, we adopt the term code smell, or simply smell, to refer to those design problems.

Despite the conception of code smell be widely accepted as indicative of bad design, some empirical studies have presented findings in other directions. Sjøberg et al. [3], for example, investigated the relationship between code smell and maintenance effort. They noted that none of the investigated code smells were significantly associated with increased maintenance effort. Macia et al. [4] investigated the relationship between code smells and problems that occur with an evolving system’s architecture. In their study, they noted that many of the detected code smells were not related to architectural

problems. Zhang et al. [5] performed a systematic mapping study on the subject and declared that “...we do not know whether using Code Bad Smells to target code improvement is effective”. In general, the authors agree that more empirical studies are necessary to better understanding the smell effect [3], [5], [6]. Specifically, the role of humans has been little studied [7]. The studies focused on human aspects commonly address agreement and decision drivers (ex. [6], [8], [9]).

The evaluation of the human aspects is not a simple task because of the extensive number of context variables that affect the human perception of code smell. In this work, our general aim is to build more empirical studies evaluating human aspects and code smells. Specifically, we have focus on the strategies adopted on *god class* detection, one of the most known code smell. Evidence of the relevance on the topic are other studies with similar aim [9], [10], [11], [6]. We explored the strategies of god class detection based on actions logged in a non-intrusive way in two controlled experiments (an original experiment and a replication).

The rest of the paper is structured as follows. Section II address some concepts and summarizes prior empirical studies that address the subject. Section III presents the planning and execution of the experiments. Section IV and V contains the results and a discussion about them. Section VI discusses the threats to the validity of the study. Lastly, Section VII presents our conclusions and proposed future works.

## II. CONTEXT AND RELATED WORKS

God class is a central concept in our study. The term was coined by Lanza and Marinescu [2] to refer classes that tend to centralize the intelligence of the system. Lanza and Marinescu presented an heuristic for god class identification based on metrics. The authors also presented the code smell brain class, in a similar way: complex classes that tend to accumulate an excessive amount of intelligence. The main difference is that a god class accesses directly many attributes from other classes. Another code smell with similar characteristics was presented by Folwer [1], but in this case subjectively. He defined the code smell large class, which is a class that try to do too much. These three code smells have a similar concept.

We presented to participants of our experiments a set of support questions to guide them. The questions were extracted from another empirical study [6] and concentrate the general idea of the three code smells. Some examples are: “Does the

class have more than one responsibility ?” and “Does the class have functionality that would fit better into other classes ?” We did not define formally the god class concept in this paper because we focused on the strategies, not in the correctness on detection. From now we will adopt only the term God Class.

#### A. Studies with similar aim of ours

Schumacher et al. [6] investigated the way developers detect god class, then compared these results to automatic classification. They built on and extended Mäntylä and Lassenius’s [9] work. Schumacher et al. study was done in a professional environment, with two real projects and two participants of each project. The participants were introduced to the god class smell in a short presentation and were then asked to detect them in specific code pieces. During this task, they received the list of questions (the same we adopted) to help with the identification of god classes.

Schumacher et al. used a “think-aloud” protocol (recorded as audio) and data collection forms. Coding was carried out to identify drivers and answers from the data collection form were used to evaluate time and agreement. Their main findings were: (1) there was a low agreement between the participants; and, (2) “misplaced method” was the stronger driver for god class detection. Related to the evaluation of automatic detection, their main findings were: (1) an automated metric-based pre-selection decreases the effort needed for manual code inspections; and, (2) automatic detection followed by manual review increases the overall confidence.

A study with aims similar to Schumacher’s was presented by Mäntylä et al. in [11] and [10]. Through a survey, they asked participants about 23 smells and used a scale from 1 (lack) to 7 (large presence) to evaluate the presence of smells in a piece of code. They received 12 completed questionnaires from 18 sent to developers in a small software company. In one of the findings the authors declare: “*the use of smells for code evaluation purposes is hard due to conflicting perceptions of different evaluators*”.

#### B. Studies with similar strategy of analysis of ours

We found only one work where the researchers used log of actions to try identify strategies of detection. Carneiro et al. [12] presented a software visualization tool with support to concerns. They investigated how the tool, named SourceMiner, helped developers on code smell detection. We will detail the SourceMiner thereafter because it is the tool that we adopt in this paper. Carneiro et al. also logged the participant’s actions, but how they had focus on the presentation of the tool and the support to concern, their observations on this topic were restricted.

Another empirical study that adopted log to understand performing of a Software Engineering task was presented by Wang et al. [13]. In this case the authors investigated how developers perform feature location tasks. Their analysis was based on 76 hours of full-screen videos of 38 developers working on 12 feature-location tasks. They adopted log to analyze the videos: they developed a tool to create and maintain a log of each developer’s work while they watched the videos. They also detached that there are few works evaluating strategies adopted by developers performing Software Engineering tasks.

### III. THE FINDING GOD CLASS EXPERIMENT

In this section we present settings of two controlled experiments: we named *Finding God Class (FinG)*. FinG aimed to address a set of context-aspects on god class detection, such as agreement, effort, strategies, and others. We had already partial results in [14]. In this paper we focused only on the investigation of the strategies adopted by the participants. We will call FinG 1 the original experiment, and FinG 2 the second experiment, which is a replication of FinG 1. In the following sections, the only difference among the FinG 1 and FinG 2 settings is the experimental unit.

#### A. The Experimental Planning

1) *Experimental Unit*: FinG 1 involved 11 undergraduate Computer Science students from the Federal University of Bahia (UFBA), in Brazil. All students were enrolled on the Software Quality course offered in the first semester in 2012. The participation in the experiment was voluntary. FinG 2 involved 23 graduate Computer Science students also from the Federal University of Bahia. All students were enrolled in the Experimental Software Engineering course offered in the second semester in 2012. Due to the special circumstances, all students of the course were (or had been) professionals. In this case, the participants received grades for their participation.

2) *Tools*: We adopted four software tools in the experiment<sup>1</sup>: (i) The Eclipse Indigo IDE; (ii) Usage Data Collector (UDC), an Eclipse plug-in for collecting IDE usage data information (interactions between participants and Eclipse can be accessed by the log of UDC). This tool is embedded in the Eclipse Indigo IDE; (iii) Task Register plug-in, a tool we developed to enable participants to indicate what task was being done at a given moment. This information was also registered in the UDC log. What all the participants had to do was to click on an item of the “Task Register” view to indicate when they were starting or finishing a task. The Figure 1-F shows the “Task Register” view; and (iv) SourceMiner, an Eclipse plug-in that provides visual resources to enhance software comprehension activities [12], [15].

We detach the SourceMiner tool because our analysis of the strategies on god class detection was based on it. The tool has visual resources that make easier the design comprehension of the programs. The SourceMiner has five visual metaphors, divided in two groups. The first group is formed by three coupling views. These views show the different types of coupling, as direct access to attributes or method calling, for instance. Moreover, they show the direction of the coupling. The coupling views are based on radial graphs, matrix of relationships, and tabular view (Figures 1 B, D and E). The second group is formed by two hierarchical views. These views associate LOC, complexity and number of methods of classes to area, width and length of rectangles. The treemap view (Figure 1 A) shows the hierarchy of package-class-method. The more internal rectangles represent methods. The area and color are used to highlight the attributes of LOC or cyclomatic complexity. The red boards represent classes and the yellow boards represent packages. The Polimetric view (Figure 1 C)

<sup>1</sup>Eclipse IDE - <http://www.eclipse.org/downloads/>; UDC - <http://www.eclipse.org/epp/usagedata/>; Task Register - private; SourceMiner - <http://www.sourceminer.org/>

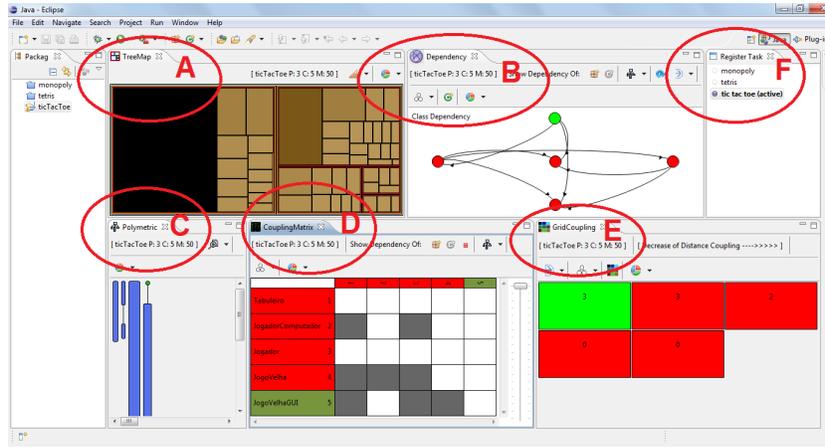


Fig. 1. Views of the SourceMiner and the Task Register plug-in

shows the hierarchy between classes and interfaces. The wider rectangles represent classes with more number of methods. Rectangles of greater length represent classes with larger LOC.

3) *Software Artifacts*: Six programs were used in the experiment. All of them implement familiar applications or games in Java. Chess, Tic Tac Toe, Monopoly and Tetris implement known games. Solitaire-Freecell (Solitaire) is a framework for card games with Solitaire and Freecell. Jackut implements a simple social network application, in the line of Facebook or Orkut. Table I characterizes the used programs in terms of the number of packages, number of classes and NLOC. It is possible to note that Monopoly is the software with higher NLOC: 2682. Due this, we consider all them simple programs. We chose simple programs to make easy their comprehension by participants because we were interested in the strategies, not in the evaluation of the difficulties on god class detection.

TABLE I. SOFTWARES USED AS STUDY OBJECTS

Software	Chess	Jackut	Tic	Monopoly	Solitaire	Tetris
Packages	5	8	2	3	6	4
Classes	15	19	5	10	23	16
NLOC	1426	978	616	2682	1758	993

4) *Design*: The both FinG 1 and FinG 2 experiments were run in a laboratory at UFBA. Participants had about 2 hours to carry out the task. Each participant worked at a separate workstation. At each workstation, we set up the Eclipse IDE fitted with the SourceMiner, the Task Register and UDC plugins. Each Eclipse with SourceMiner had three of the six programs in their workspace. The workstations were divided into two groups. In FinG 1 there were six participants in the group 1 and five participants in the group 2. In FinG 2 there were 13 participants in the group 1 and 11 participants in the group 2. We present the distribution in the Table II.

TABLE II. DISTRIBUTION OF THE PARTICIPANTS BY GROUP

Group	Programs	Participants of FinG 1 (ID)	Participants of FinG 2 (ID)
1	Chess, Jackut and Solitaire	14, 21, 32, 35 and 44	1, 3, 4, 7, 9, 13, 15, 17, 19, 21, 23, 25 and 27
2	Monopoly, Tetris and Tic Tac Toe	13, 15, 25, 31 and 41	5, 6, 8, 10, 12, 14, 16, 18, 20, 22 and 26

## B. Execution

1) *Preparation*: Participants of the both FinG 1 and FinG 2 experiments were trained on the god class concept and on the SourceMiner tool. We performed the two training in different weeks. In order to guarantee some expertise using the SourceMiner to detect god class, we performed, as part of the training, a practical exercise in the lab asking them to search god class using the SourceMiner. All participants of both experiments performed the exercise.

2) *Data*: We used the UDC plugin to log participant's actions while the experiment was running. UDC is a framework for collecting usage data on various aspects of the Eclipse workbench. It gathers information about the kinds of activities that the user is doing in the IDE (i.e. activating views, editors, etc.). The Task Register (Figure 1-F) was used to enrich the UDC log with higher level information. Figure 2 shows a clipping of the UDC log annotated by the Task Register plug-in. The first column ("task") does not exist in the original UDC log. It was added by the Task Register. We highlighted columns that we were interested in. The first column ("task") indicates the program for which the participant was doing the god class detection task. Columns "what", "kind" and "description" describe the actions. For example, the first line represents: user activated the Package Explorer view. As a result, we have sequences of actions for each participant and for each program.

## IV. RESULTS

To investigate the strategies adopted by the participants we explored their preferences for the views of SourceMiner during the detection of god classes. We used the UDC log of actions to do this. Some participants read more and used fewer views, others did the opposite. To evaluate these aspects we calculated the ratio between the number of classes investigated for each program and the use of views and reading. We counted the number of actions related to reading (activation of the Compilation Unit Editor), activation of hierarchical views (Polimetric or Tree Map), and activation of coupling views (Dependency, GridCoupling or CouplingMatrix). For example, the participant 14 of FinG 1 did not activate the Compilation Unit Editor (i.e. read the code), activated some of the coupling

task	what	kind	bundleId	bundleVersion	description	time
jackut	activated	view	org.eclipse.jdt.ui	3.7.2.v20120109-1427	org.eclipse.jdt.ui.PackageExplorer	1362837914406
jackut	opened	view	org.gesa.sourceminier	2.17.0	org.gesa.sourceminier.view.paradigms.CouplingMatrixView	1362837914438
jackut	activated	view	org.gesa.sourceminier	2.17.0	org.gesa.sourceminier.view.paradigms.DependencyView	1362837943313
jackut	opened	editor	org.eclipse.jdt.ui	3.7.2.v20120109-1427	org.eclipse.jdt.ui.CompilationUnitEditor	1362838004980
jackut	activated	editor	org.eclipse.jdt.ui	3.7.2.v20120109-1427	org.eclipse.jdt.ui.CompilationUnitEditor	1362838005011
jackut	activated	view	org.eclipse.jdt.ui	3.7.2.v20120109-1427	org.eclipse.jdt.ui.PackageExplorer	1362838006790

Fig. 2. Clipping of the user UDC log

views 41 times and activated some of the hierarchical views 39 times. He/she did this for the programs Chess, Jackut and Solitaire. The total number of classes for these three programs is 57. We defined the ratios for the participant 14 as 0/57, 41/57 and 39/57 for reading, use of coupling views and use of hierarchical views, respectively.

The Figure 3 (shown on next page) shows in a bar graph the ratio using views for all participants of FinG 1. The ratio for the use of reading is in red. In green we show the ratio for the use of coupling views. In blue the ratio for the use of hierarchical views. We grouped participants with similar strategy. The lines above the bars group name them. In the Figure 4 we show the results for FinG 2.

From the Figures 3 and 4 we identified a set of strategies. For example, in the Figure 3 we noted that the first three participants (id's 14, 21 and 42) had similar focus on the use of hierarchical and coupling views and they did not read the source code or used very few reading. The first participant of FinG 2, in the Figure 4, adopted the same strategy. We called it Strategy 1. In the Table III we show all strategies identified. All them were identified according to high focus or none/few use on the views or reading. For example, for the Strategy 3, the participants had focus on coupling views, and none or few use of reading. It is important to note that we did not define a threshold for high or none focus because we compared preferences between views considering participants individually. We disregarded the number of clicks on the views because we did not analyze the effort.

After identifying the strategies we counted the number of participants with focus on each type of views or reading. We also counted the number of participants with few or none use of each type of view or reading. We show the result in the Table IV. For the Strategy 1, there were three participants of the FinG 1 (id's 13, 21 and 42) and one participant of the FinG 2 (id 4). Then in the Table IV we registered that there were four participants with focus on coupling and hierarchical views, and four participants with few or none use of reading. The two last lines in the table show the total of participants with focus or with few or none use of views or reading and the percentage considering the total of participants of both FinG 1 and FinG 2 experiments (35).

## V. DISCUSSION

From the Table IV we noted that the god class detection was mainly focused on coupling views. Almost 63% of the participants adopted strategies based on coupling views. Once coupling was preferred instead hierarchical attributes, we suggest that on god class detection to observe coupling is more relevant. As the essence of the god class concept is related to classes with many roles, we linked the idea of participants on identifying many roles with the coupling among

TABLE III. STRATEGIES ADOPTED

N	Strategy	Participants of FinG 1 (ID)	Participants of FinG 2 (ID)
1	Similar focus on coupling and hierarchical/ few or no reading	14, 21, 42	4
2	Similar focus on coupling and hierarchical		12, 18, 19
3	Focus on coupling/ few or no reading	31, 41	5, 10, 20
4	Focus on reading/ few hierarchical	15, 25	7, 8, 16
5	Focus on coupling		3, 15, 17, 22
6	Similar focus on reading and coupling		13, 27
7	Focus on reading/ few coupling		14, 25
8	Focus on hierarchical/ few reading		26, 23
9	Similar focus on coupling, hierarchical and reading	44	9, 21
10	Focus on coupling/ few or no hierarchical		6
11	Focus on reading		1
12	Focus on hierarchical/ few coupling	35	
13	Only reading/no views	32	
14	Focus on hierarchical	13	

TABLE IV. NUMBER OF PARTICIPANTS FOCUSING ON VIEWS

Strategy	Reading		Coupling		Hierarchical	
	High focus	Few/ none	High focus	Few/ none	High focus	Few/ none
1		4	4		4	
2			3		3	
3		5	5			
4	5					5
5			4			
6	2		2			
7	2			2		
8		2			2	
9	3		3		3	
10			1			1
11	1					
12				1	1	
13	1			1		1
14					1	
<b>Total</b>	14	11	22	4	14	7
<b>% (out of 35)</b>	40.0	31.4	62.9	11.4	40.0	20.0

classes. Hierarchical attributes, as LOC or complexity, were less adopted.

Another interesting finding is that, even with visualization resources, 40% of the participants had focus on reading in their strategies. For us, this is an evidence that some reading is necessary on god class detection. Our conjecture is that the

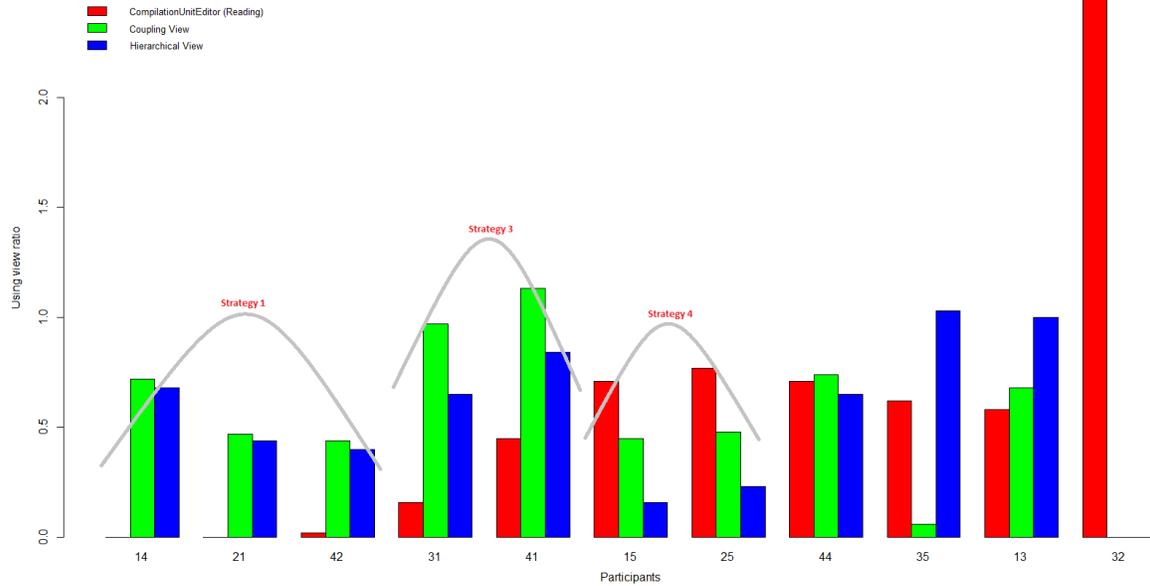


Fig. 3. Ratio of using reading, coupling and hierarchical views for FinG 1

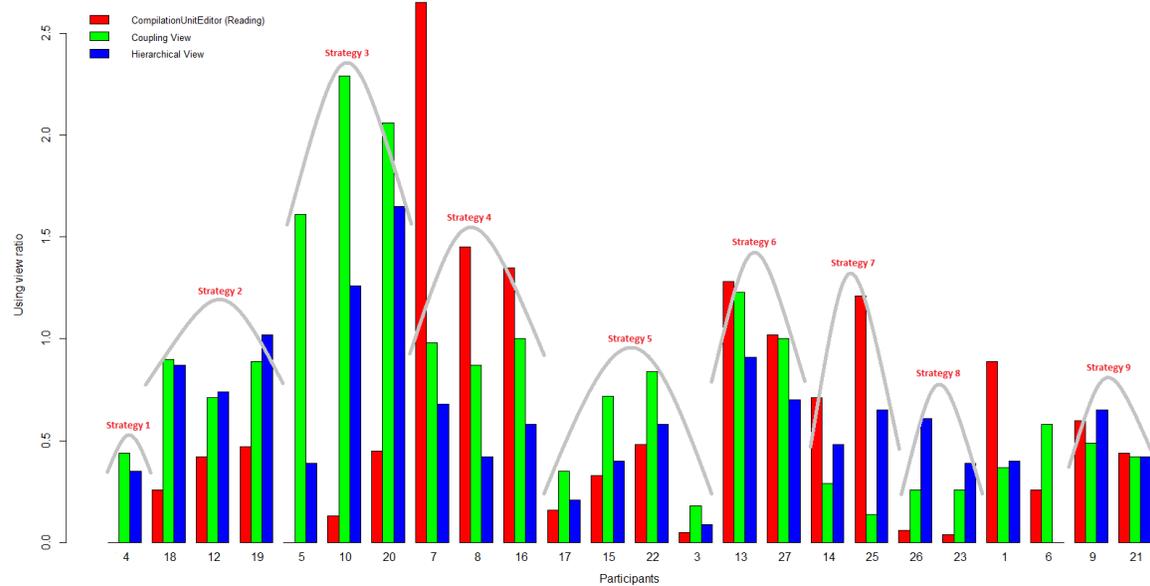


Fig. 4. Ratio of using reading, coupling and hierarchical views for FinG 2

reading is used to identify context aspects about the classes. For example, in some cases, to identify if a method is in according with the role of the class, reading source code is important. On the other hand, the few or none use of reading had highest percentage among the few use of views, 31.4%. We consider that the value is not high, but it can be an evidence that visual resources are important because make possible for developers filter specific classes to read.

An interesting observation is that our analysis presents a non usual perspective investigating code smell empirically. Papers addressing how developers detect code smell usually depends on the human answers, in interviews [6], questionnaires [9] or surveys [11], [10]. We adopt a non usual technique. The main benefits of the use of log are the high volume of data

and the absence of disparity between the participants responses and the “reality” [16], [17]. We highlight the importance of the use of complementary strategies of data collection to analyze similar problems from different perspectives. Due to this focus and the space constraints, we did not present analysis of other variables (such as the impact of the experience) in this paper, which is possible by the setting of FinG.

## VI. THREATS TO VALIDITY

Our analysis of threats was based on Wohlin et al. [18].

*External validity.* Our first threat fits in the “interaction of selection and treatment” subcategory and is related to fact that the participants in FinG 1 were undergraduate students.

Although this aspect can be considered a problem for generalization, we found similar strategies in both FinG 1 and FinG 2 experiments, which make this threat weaker. Other threat to external validity fits in the “interaction of setting and treatment” subcategory. In this case, the threat is the type of programs. We adopted simple softwares. However, we argue the impact of this threat is small because we consider that the strategy is more dependent of the participant than of the complexity of the software.

*Internal validity.* A subcategory of the internal validity is “maturation”. Participants could be affected because they do the same task over three programs so they may learn as they go and work faster. On the other hand they could be affected negatively because of boredom. We consider maturation a weak threat because the experiment was performed in 1 hour, on average. We consider this a reasonable period of time to do a task in a balanced way.

*Conclusion validity.* In the “reliability of measures” category we should report that the logged information represents the actions of the participants only indirectly. They represents actions of the Eclipse IDE. For example, if a developer changes the perspective in the Eclipse, some views are activated by the tool and these actions are registered in the log. To mitigate this aspect, we investigated the logging to evaluate actions in detail and eliminated lines clearly related to Eclipse actions. Moreover, these registers occurred for all participants and did not affect the general conclusion. In the “reliability of treatment implementation” we have to consider if that the visualization tool was appropriated for the identification of useful attributes on god class detection. We are confident that this threat did not have impact on our results because of the discussion occurred during the training based on exercise. Lastly, our findings were based on the analysis of the log, and we did not present inferential testing.

## VII. CONCLUSION AND FUTURE WORKS

In this work was investigated how developers detect god class. More specifically, we explored strategies adopted by developers detecting god class. To do this we performed two controlled experiments. The setup of the experiments was based on the use of a software visualization tool fitted in the IDE Eclipse. We logged actions performed by participants using the visual resources grouped in three categories: use of Compilation Editor of the Eclipse (or reading code); use of hierarchical views (focused in attributes as LOC or complexity); or use of coupling views.

Our first finding was that coupling attributes are more relevant than LOC or complexity on god class detection. We also noted that, even with visual resources, that make possible to enhance the general comprehension of the design, reading of the source code remains important. We suggest that the reading is important for developers to evaluate if the methods are in accordance with the role of the classes. We also noted that the visual resources helped participants to filter candidate god classes to read. We consider our empirical results as an initial and complementary approach to investigate the real impact of code smell on software development.

To evolve our study in this topic, we will explore other context aspects of the experiments, as effort and decision

drivers. We also intent replicate the experiments focusing in other types of code smell. To mitigate some limitations we will replicate the experiments with similar setting addressing more complex software. To support replication we provide the experimental package, available in the site of FinG 1<sup>2</sup> and FinG 2<sup>3</sup>. The packages contain forms, data and software.

## ACKNOWLEDGMENT

This work was partially supported by the National Institute of Science and Technology for Software Engineering (INES)<sup>4</sup>, funded by CNPq, grants 573964/2008-4.

## REFERENCES

- [1] M. Fowler, *Refactoring: improving the design of existing code*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.
- [2] M. Lanza, R. Marinescu, and S. Ducasse, *Object-Oriented Metrics in Practice*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005.
- [3] D. Sjöberg, A. Yamashita, B. Anda, A. Mockus, and T. Dyba, “Quantifying the effect of code smells on maintenance effort,” *Software Engineering, IEEE Transactions on*, vol. 39, no. 8, 2013.
- [4] I. Macia, J. Garcia, D. Popescu, A. Garcia, N. Medvidovic, and A. von Staa, “Are automatically-detected code anomalies relevant to architectural modularity?: An exploratory analysis of evolving systems,” in *Proc. of the 11th AOSD*, 2012, pp. 167–178.
- [5] M. Zhang, T. Hall, and N. Baddoo, “Code bad smells: A review of current knowledge,” *J. Softw. Maint. Evol.*, vol. 23, no. 3, Apr. 2011.
- [6] J. Schumacher, N. Zazworka, F. Shull, C. Seaman, and M. Shaw, “Building empirical support for automated code smell detection,” in *Proc. of the 4th ESEM*, 2010.
- [7] M. V. Mäntylä and C. Lassenius, “Subjective evaluation of software evolvability using code smells: An empirical study,” *Empirical Softw. Engg.*, vol. 11, no. 3, pp. 395–431, 2006.
- [8] L. Moonen and A. Yamashita, “Do code smells reflect important maintainability aspects?” in *Proc. of 28th ICSM*, 2012, pp. 306–315.
- [9] M. V. Mäntylä and C. Lassenius, “Drivers for software refactoring decisions,” in *Proc. of 5th ISESE*, 2006, pp. 297–306.
- [10] M. Mäntylä and C. Lassenius, “Subjective evaluation of software evolvability using code smells: An empirical study,” *Empirical Software Engineering*, vol. 11, no. 3, pp. 395–431, 2006.
- [11] M. V. Mantyla, J. Vanhanen, and C. Lassenius, “Bad smells humans as code critics,” in *Proc. of the 20th ICSM*, 2004, pp. 399–408.
- [12] G. Carneiro, M. Silva, L. Maia, E. Figueiredo, C. Sant’Anna, A. Garcia, and M. Mendonça, “Identifying code smells with multiple concern views,” in *Proc. of the 1th CBSOFT*, 2010.
- [13] J. Wang, X. Peng, Z. Xing, and W. Zhao, “An exploratory study of feature location process: Distinct phases, recurring patterns, and elementary actions,” in *Proc. of 27th ICSM*, 2011.
- [14] J. A. Santos, M. Mendonça, and C. Silva, “An exploratory study to investigate the impact of conceptualization in god class detection,” in *Proc. of the 17th EASE*, 2013, pp. 48–59.
- [15] G. F. Carneiro and M. G. Mendonça, “Sourceminer - a multi-perspective software visualization environment,” in *Proc. of the 15th ICEIS*, 2013.
- [16] J. Singer, S. E. Sim, and T. C. Lethbridge, “Software engineering data collection for field studies,” in *Guide to Advanced Empirical Software Engineering*, F. Shull, J. Singer, and D. I. K. Sjöberg, Eds., 2008.
- [17] H. K. Jonathan I. Maletic, “Expressiveness and effectiveness of program comprehension: Thoughts on future research directions,” in *Frontiers of Software Maintenance, FoSM*, 2008, pp. 31–37.
- [18] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*. Springer Berlin Heidelberg, 2012.

<sup>2</sup>wiki.dcc.ufba.br/LES/FindingGdbClassExperiment2012

<sup>3</sup>wiki.dcc.ufba.br/LES/FingTwo

<sup>4</sup>www.ines.org.br

# An Empirical Study to Evaluate a Domain Specific Language for Formalizing Software Engineering Experiments

Marília Freire<sup>a,b</sup>, Uirá Kulesza<sup>a</sup>, Eduardo Aranha<sup>a</sup>, Andreas Jedlitschka<sup>c</sup>,  
Edmilson Campos<sup>a,b</sup>, Silvia T. Acuña<sup>d</sup>, Marta N. Gómez<sup>e</sup>

<sup>a</sup>Federal University of Rio Grande do Norte, Department of Informatics and Applied Mathematics, Natal, Brasil

<sup>b</sup>Federal Institute of Rio Grande do Norte, Natal, Brasil

<sup>c</sup>Fraunhofer Institute for Experimental Software Engineering, Kaiserslautern, Germany

<sup>d</sup>Universidad Autónoma de Madrid, Madrid, Spain

<sup>e</sup>Universidad San Pablo-CEU, Madrid, Spain

{marilia.freire,edmilsoncampos}@ppgsc.ufrn.br,{uira,eduardo}@dimap.ufrn.br, andreas.jedlitschka@iese.fraunhofer.de,  
silvia.acunna@uam.es, mn.gomez@usp.ceu.es

**Abstract**—The research about the formalization and conduction of controlled experiments in software engineering has reported important insights and guidelines for conducting an experiment. However, the computational support to formalize and execute controlled experiments still requires deeper investigation. In this context, this paper presents an empirical study that evaluates a domain-specific language proposed to formalize controlled experiments in software engineering. The language is part of an approach that allows the generation of executable workflows for the experiment participants, according to the statistical design of the experiment. Our study involves the modeling of eight software engineering experiments to analyze the completeness and expressiveness of our domain-specific language when specifying different controlled experiments. The results highlighted several limitations that affect the formalization and execution of experiments. These outcomes were used to extend the evaluated domain-specific language. Finally, the improved version of the language was used to model the same experiments to show the benefits of the improvements.

## I. INTRODUCTION

The conduction of controlled experiments in software engineering has increased over the last years [1] [2]. Experiments and their replication are considered essential to produce scientific evidences and to enable causal effect analysis, improving the knowledge about the benefits and limitations of new and existing software engineering (SE) methods, theories and technologies. In addition, it also promotes the sharing of this knowledge inside the SE community. Consequently, controlled experiments can accelerate the development and evaluation of effective scientific innovations produced by the academy or the industry.

Over the last decade, the community discussed how to better support the application of controlled experiments in the SE research area. These studies have proposed guidelines to report controlled experiments [3], conceptual frameworks to guide the replication of controlled experiments [4], and environments/tools to support their conduction and replication

[5] [6] [7]. Although these studies brought important insights and outcomes related to the conduction of controlled experiments, few of them [7][8] [9] [10] [11] have proposed to formalize the planning, execution and analysis of controlled experiments. In addition, the existing approaches that aim at formalizing the specification of controlled experiments – such as ontologies or domain-specific languages – and/or at providing support for their execution and analysis have not been evaluated. Therefore, this subject still requires deeper investigation.

In this context, this paper describes an empirical study conducted to evaluate a domain specific language that provides support to the modeling and execution of SE controlled experiments proposed by our research group [8] [12]. In our study, we have modeled several controlled experiments with the objective of analyzing the completeness and expressivity of the domain-specific language (DSL) and supporting environment of our approach. These criteria were analyzed based on the specification of eight different experiments and considering fundamental experimental aspects documented by the experimental software engineering community [1] [2] [3]. We present the results of our analysis and illustrate how they have been used to improve the evaluated domain-specific language.

This article is organized as follows. Section II describes the study settings. Section III discusses the evaluation results, as well as the new extensions proposed for the evaluated DSL. Sections IV and V present, respectively, the threats to validity and related work. Finally, Section VI presents conclusions and directions for future work.

## II. STUDY SETTINGS

This section presents the study settings in terms of: its main goal and research questions (Section II.A), the approach being evaluated (Section II.B), the study methodology (Section II.C), and the evaluation criteria (Section II.4).

### A. Study Goal and Research Question

**Table 1: Experiments modeled in our study.**

Experiment	Institution-Country
Testing [17]	UFPE-Brazil
Human Factors [14]	UAM-Spain
Requirements [15]	UPM-Spain
SPrL [20]	UFRN-Brazil
MDD	UPV-Spain
SPL [21]	PUC-Rio-Brazil
CFT [18]	Fraunhofer-IESE-Germany
PBR [19]	University of Maryland-USA

The main goal of this study is to validate the experimental domain specific language with respect to the modeling of SE controlled experiments from the perspective of the experimenters. To achieve this goal, a research question (RQ) was defined: Are the DSL abstractions of the approach adequate to model the different aspects of SE controlled experiments?

In order to answer this question, our study investigated how the different specification aspects of controlled experiments could be addressed by our DSL. Different and complementary criteria were adopted to analyze the completeness and expressiveness of the domain-specific language during the modeling of a set of existing experiments.

**B. Evaluated Domain Specific Language**

Our study focuses on the assessment of a DSL that is part of a model-driven approach for supporting the formalization and execution of experiments in software engineering [8] [12]. The approach consist of: (i) a DSL, called ExpDSL, used to describe the process and statistical design of controlled experiments; (ii) model transformations that allow the generation of workflow models that are specific for each experiment participant and according to the experiment’s design; and (iii) a workflow execution environment that guides and monitors the participant activities during the experiment execution, including the gathering of participants feedback during this process.

ExpDSL is a textual DSL that is composed of four main parts/views: process view, metrics views, experimental plan view, and questionnaire view. Each view allows defining the experiment aspects as follows: (i) *Process View* – allows defining the activities of data collection from the experiment participants. It is similar to a software process language, where we can define the activities, tasks, artifacts, and roles involved in the process; (ii) *Metric View* – defines the metrics that have to be collected during the experiment execution. Each metric intercepts process activities or tasks in order to quantify observations of interest (dependent variables, etc.) in the study; (iii) *Experimental Plan View* – defines the experimental plan by identifying the factors to be controlled (treatments, noise variables, etc.) and the statistical design to be used (treatments, participants and experimental material allocation); and (iv) *Questionnaire View* – defines questionnaires to collect quantitative and qualitative data from participants of the

experiment. These questionnaires can be applied before or after all the activities of the experiment process.

```

ExpDSLv1
Goals {
A G1 "Analyze test execution effort, for the purpose of evaluating the
effect of two different test case design techniques for SPL (GT vs. ST)"
}
Hypotheses{
B H0 "The number of terminated CRs using ST is the same compared with the
GT" null from G1
H1 "The number of terminated CRs using ST is the same compared with the
GT" alternative from G1
}
DesignType LS - Latin Square {
C Parameter LPS_complexity = "low"
Parameter Participants = "undergraduating students"

Factor TestTechnique isDesiredVariation True
Level Generic
Level Specific

D Factor Feature isDesiredVariation False
Level F1
Level F2
Factor Subject isDesiredVariation False
Level Subject1
Level Subject2

Internal Replication 4

E Link SpecificTecFeature1_Tests to TestTechnique.Specific Feature.F1
Link SpecificTecFeature2_Tests to TestTechnique.Specific Feature.F2
Link GenericTecFeature1_Tests to TestTechnique.Generic Feature.F1
Link GenericTecFeature2_Tests to TestTechnique.Generic Feature.F2
}
Process SpecificTecFeature1_Tests {
F Activity SP1_F1_1 description "Insert a new member - P1" to SP1_F1_2 Role
"Subject" {
Task ExecuteTCSP1_F1_1 description "Execute the test case SP1_F1_1"
artefacts artefact TesteCaseSP1_F1_1 description "Test case
details SP1_F1_1" type input
Task ReportCRSP1_F1_1 description "Describe TC SP1_F1_1 results"
artefacts artefact CR_SP1_F1_1 description "Reporting the CR for
Test Case SP1_F1_1" type output
}
...
G Metric ReportedCR1 relates SpecificTecFeature1_Tests {
description "Valid Terminated CR"
artefacts CRs
SpecificTecFeature1_Tests.SP1_F1_1.ReportCRSP1_F1_1.CRSP1_F1_1
SpecificTecFeature1_Tests.SP1_F1_2.ReportCRSP1_F1_2.CRSP1_F1_2
SpecificTecFeature1_Tests.SP1_F1_3.ReportCRSP1_F1_3.CRSP1_F1_3
SpecificTecFeature1_Tests.SP1_F1_4.ReportCRSP1_F1_4.CRSP1_F1_4
SpecificTecFeature1_Tests.SP1_F1_5.ReportCRSP1_F1_5.CRSP1_F1_5
SpecificTecFeature1_Tests.SP1_F1_6.ReportCRSP1_F1_6.CRSP1_F1_6
}
...
H Questionnaire ExperimentFeedback type Post {
Q1_Scholling {
description "What is your scholling:"
type SingleChoice
Alternatives
Option "Undergraduate student"
Option "Graduate"
Option "Graduate student"
Option "Master"
}
}
}

```

Figure 1: ExpDSL fragment before extensions

Fig. 1 shows the fragments of the “Testing” experiment specification [13] using the original version of our DSL (ExpDSLv1). It is a controlled experiment that compares two different black-box manual test design techniques: a generic technique and a product specific technique. The study evaluates those techniques from the point of view of the test execution process. In the specification, we can see an Experimental Plan View (Fig. 1- A,B,C,D,E), a Process View fragment (Fig. 1 – F), a Metric View fragment (Fig. 1 - G) and a Questionnaire View fragment (Fig. 1 – H). Section III presents and discusses those fragments in the context of our study.

```

ExpDSLv2
...
Hypotheses {
H0 "The number of terminated CRs using ST is the same compared with the
GT" null from G1 {
A H01 NumberofCR(TestTechnique.Specific)=NumberofCR(TestTechnique.Generic)
}
H1 "The number of terminated CRs using ST is the same compared with the
GT" alternative from G1 {
H11 NumberofCR(TestTechnique.Specific)>=NumberofCR(TestTechnique.Generic)
}
}

DesignOfExperiment = LS - Latin Square {
...
B Dependent Variable TestExecutionTime "Spent time for each test in the
Test Suite" TimeExecution1 TimeExecution2 TimeExecution3 TimeExecution4
C Dependent Variable NumberofCR "Number of terminated CR" ReportedCR1
ReportedCR2 ReportedCR3 ReportedCR4

Factor TestTechnique isDesiredVariation True
{ Generic, Specific }
Factor Feature isDesiredVariation False
{ F1, F2 }
Factor Participants isDesiredVariation False
{ Participant1, Participant2 }

D Statistical Analysis Technique H10.H101 H11.H111 : ANOVA
...
H20.H201 H21.H211 : Others
...
}
...
E Process SpecificTecFeature1_Tests to TestTechnique.Specific Feature.F1{
Role Participant
Activity SP1_F1_1 description "Insert a new member - P1" to SP1_F1_2 {
F Task ExecuteTCSP1_F1_1 description "Execute the test case SP1_F1_1"
artefacts TCSP1_F1_1 description "Test SP1 Feature 1" type input
Task ReportCRSP1_F1_1 description "Report CR to test case SP1_F1_1"
var ReportedCR1.CR
}
...
F Metric ReportedCR1 relates SpecificTecFeature1_Tests
...
G { description "CR Reported during test execution for Specific Technique
- Feature 1"
collectedData CR: text
}
...

```

Figure 2: ExpDSLfragment after extensions

The experiment modeled in ExpDSL is submitted in our approach to a set of transformations (model to model and model to text) to generate workflow models for the participants according to the statistical design of experiment (DoE). Finally, these workflow models can be executed in a workflow engine (web application), guiding and monitoring the participants during the experiment. For additional details of our approach, please refer to [12].

C. Study Methodology

Our study was organized in three main phases: (i) the selection and specification of different experiments using the evaluated version of ExpDSL herein called ExpDSLv1; (ii) the evaluation of each modeled experiment through the study criteria (Section II.D); (iii) the analysis, discussion, and proposal of improvements for the ExpDSL and the approach considering the study results.

We selected 2 quasi-experiments and 6 controlled experiments with different statistical designs, executed and documented by the software engineering community. We have also considered experiments from different research groups and software development areas (requirements, model-driven, software product lines, testing, human factors). Finally, the selection also took into consideration the availability of

information about the experiment planning and conduction. Table 1 lists the experiments used in our study. The specification of the experiments in ExpDSL is available at: <http://migre.me/hqP6N>. Our team – the authors of this paper – modeled different aspects of the experiments using the ExpDSLv1 based on their available documentation. For most of the experiments, we have also interacted with the researchers that conducted them.

In the last phase of our study, we analyzed the collected results for each specified experiment, and evaluated how the completeness and expressiveness criteria were tackled by the approach. During this analysis, we investigated for each criterion: (i) the reasons for the (non)adequate specification of each specific aspect of the experiment; and (ii) which improvements can be applied to the DSL to address a better modeling of the identified aspects. Finally, we implemented these new improvements and re-modeled the same experiments using the improved version of ExpDSL (herein called ExpDSLv2) to highlight the achieved results. Fig. 2 shows fragments of new or modified elements in ExpDSLv2.

D. Adopted Criteria

Our study adopted three main assessment criteria used for evaluating DSLs [13]:

**Completeness:** All concepts of the domain can be expressed in the DSL.

**Expressiveness:** The degree to which a problem solving strategy can be mapped into a program naturally. In particular in this study, we evaluated the orthogonality aspect of expressiveness [13] that states that each DSL construct is used to represent exactly one distinct concept in the domain.

For the completeness analysis, we investigated how the different chosen experiments can be properly specified using the ExpDSLv1. The following aspects were assessed during the specification: Goals, Hypotheses, Subhypotheses, Design of experiment (DoE), Independent variables (controlled variable), Dependent Variables, Metrics, Measurement Instruments, Characterization/ Contextualization, Data Collection Procedure, Experimental roles, Statistical Analysis Technique, and Questionnaires. Our analysis considered three levels regarding the completeness of the specification: (i) supported– it was possible to specify the aspect; (ii) partially supported –the specification required adjustments to the modeling of the experiment aspect; and (iii) not supported – the DSL cannot specify that specific experiment aspect.

The expressiveness analysis involved verifying if each ExpDSLv1 construct was used to specify exactly one concept during the specification of the chosen experiments. Thus, if there are constructs being used to specify different abstractions of the experiments, our analysis consisted on the identification of such scenarios, which indicate a lack of expressiveness of the DSL.

III. ANALYSIS OF THE RESULTS

A. General Results

The modeling of controlled experiments in the study revealed that the investigated DSL successfully addressed most

of the evaluated criteria. The completeness analysis showed that 60% of experimental aspects of the modeled experiments were supported, 15% partially supported, and 25% not supported. On the other hand, the expressiveness criteria revealed that there was only one ExpDSLv1 construct, the Metric element, which was used to specify three different concepts of the specified experiments.

Regarding the completeness of modeled experimental aspects – about 60% – were successfully satisfied. On the other hand, the results also show that a high percentage of experimental aspects was only partially supported (15%) or not supported (25%) by our approach. This demanded the investigation of how we could improve the ExpDSLv1 and our approach to allow their adequate specification. There were a few cases where the experiment aspect was not specified because it was not part of the experiment (n/a – not applicable).

Only three concepts were not supported for all modeled experiments, which are the dependent variable, measurement instruments, and the statistical analysis technique. These results motivated to propose ExpDSLv1 extensions to include: (i) a *DependentVariable* element that relates this concept with hypotheses and metrics, allowing the traceability among them and facilitating the conduction of meta-analysis in the future; and (ii) a statistical analysis element that allows documenting the statistical test which can be used for each experiment hypotheses.

There were also some scenarios where experimental aspects (such as metrics and experimental roles) have been only partially satisfied during the DSL completeness assessment. The metric element, for example, only allows specifying metrics regarding execution time for activity/task, and metrics that are quantified based on the produced artifacts. This result has also motivated the extension of ExpDSL to specify a new kind of metric related to any collected data informed by the user during an activity/task execution of the experiment process. Section III.C shows how we have improved the expressiveness of the language by extending the metric concept. It also discusses the lack of expressiveness of ExpDSLv1 to specify the design of experiment (DoE) and statistical analysis technique, and how language users can deal with that.

Next sections detail the results of the study regarding the modeling of different experimental aspects by presenting limitations of our approach and proposing new improvements.

## B. Completeness Results and DSL Improvement.

This section describes the main results of our completeness study considering the different experimental aspects that were modeled in ExpDSLv1. For the more affected elements, we show and discuss the obtained results when modeling the experiments. In addition, we also describe how those results have been used to propose improvements and new extensions for the ExpDSL (ExpDSLv2).

### 1) Hypotheses

ExpDSLv1 allows the definition of statistical hypotheses for the experiment through the Hypothesis element. In our study, it was possible to model all hypotheses of the experiments using such language element. Fig. 1 – (B) shows the hypothesis specification for the “Testing” experiment.

To enable automatic analysis of hypotheses after that experiment execution, we propose a more formal hypotheses definition. This allows specifying an expression that defines the hypothesis relating the factor/levels with a dependent variable. Fig. 2 - (A) shows the new formalization of hypotheses after the language extension. Each textual hypothesis can be decomposed in one or more formal hypotheses, which associate the Factor or Treatment with the dependent variable.

There are two additional benefits provided by this DSL extension: (i) it contributes to the automation of the statistical hypotheses analysis; and (ii) it supports the analysis of variables used in different experiments (meta-analysis). Besides, it allows the hypotheses specification in a more formal way, associating dependents variables and treatments.

### 2) Design of Experiment

ExpDSLv1 supports three experimental DoE: (i) completely randomized design (CRD); (ii) randomized complete block design (RCBD); and (iii) Latin square (LS). For this reason, it was not possible to model the designs used in the Human Factors and Requirements experiments, because both experiments used a simple prospective ex post facto quasi-experimental design [14] [15]. The same issue happens to the PBR experiment, which adopts a factorial design in blocks of size 2. The wrong choice of one of the existing supported DoE of our DSL could cause a problem during the model transformation, because the distribution of treatments will not correspond to the real design of these experiments.

For allowing the modeling of experiments that do not follow any of the supported DoE in ExpDSLv2, we extend the language to include the “Other” option. This change has a direct impact in the experiment configuration. Without knowledge about the experimental design, transformations of models specified in ExpDSLv2 are not able to automatically randomize and allocate treatments to participants and experimental materials. Therefore, the experimenter has to configure this information manually using the execution environment using the ExpDSLv2. Only after this manual configuration, the execution environment instantiates the workflow correctly to start the experiment execution.

The choice of the “Other” DoE also makes difficult automatic validations of ExpDSLv2 specification based on statistical design rules and assumptions. Our DSL editor can check, for example, if the requirement of the Latin Square DoE is satisfied: existence of two variables to block. Therefore, the “Other” option increases language flexibility but limits specification validations.

### 3) Dependent Variable

In ExpDSLv1, we can define metrics related to dependent variables but there is no explicit ways to specify them. Because of that, we have extended the ExpDSLv1 to include the modeling of dependent variable that can be mapped to: (i) metrics that are associated to the execution time of specific activities or tasks of the experiments; or (ii) metrics that are associated to fields (of process activities/tasks) and that will be informed by the participants or researchers (number of defects found, etc.) during the experiment execution.

Fig 2 – (B) shows the dependent variable specified for the “Testing” experiment. There are two response variables in this experiment: the number of valid terminated CR (*NumberofCR*), and the time to execute the tests (*TestExecutionTime*). The *TestExecutionTime*, for example, has a description and is associated to the metrics *TimeExecution1*, *TimeExecution2*, *TimeExecution3*, and *TimeExecution4*. These metrics are defined in the experiment specification in the Metrics View.

#### 4) Metrics

All metrics of the experiments were modeled by using one of the two kind of existing metrics in ExpDSLv1: those related to activities/tasks execution time and those associated to an artifact. Participants inform the value of a metric by using a text box provided in a web form related to the participant workflow. Another option is to define an *ArtifactMetric* element, forcing the participant to send a file whose content was only a value (text or number), for example. All these options represent workarounds that do not allow associating the informed metric value with the experiment hypotheses or other specific element. Fig. 1- (G) shows the definition of the metric Reported CRs used in the “Testing” experiment. In this experiment, the participant will be asked to upload a file whose content represents the collected CRs for each reported task. The specification shows the metric defined to the process related to the Specific Technique and Feature 1 (*SpecificTecFeature1\_Tests*). The metric name is “ReportedCR1”. The output artifacts related to the CR reporting are listed in the artifacts attribute of this metric (artifacts “CR SP1\_F1\_1”, “CR SP1\_F1\_2, etc.).

In our study, the specification of different experiments using such strategy indicated the need to provide support to inform the metric value associated to an activity/task in the experiment. Such support contributes to not only explicitly identify the metric, but also associate it with the experiment hypotheses thus contributing to the analysis phase. Therefore, we extended the ExpDSLv1 creating a new Metric type named *DataMetric* that can be used to specify metrics associated to artifacts and that can be collected by the experiment subjects and/or researchers. This adaptation also provides an association between a metric and a text or number variable – the *collectedData* attribute. It has to be defined during the metric definition, and it is also used and collected in the correspondent activity/task of the experiment process.

Fig. 2 - (G) shows the specification of the “ReportedCR1” metric in ExpDSLv2. It has a description and a *collectedData* text attribute named “CR”. This “CR” *collectedData* represents a data that will be gathered during the experiment execution, according to the process specification. Fig. 2 – (G) also illustrates that this “ReportedCR1” metric is associated to the “Number of CR” dependent variable. This will improve the traceability between dependent variable, metric and associated activity/tasks fields, facilitating their collection during the experiment execution and their usage during the experiment analysis.

#### 5) Data collection procedure

In ExpDSLv1, data collection procedures are specified as a process in the Process View, containing activities, tasks, roles and artifacts. Our DSL does not currently support loops and

conditionals paths but almost all the experiment could be modeled as sequential procedures. The semantic related to link the process to the correspondent treatment combination, Link element in ExpDSLv1 (Fig. 1 – E), was moved to the process definition (Fig. 2 – E), so the Link element was removed from the DSL. This element was not used in the quasi-experiment definitions (Human Factors and Requirements).

#### 6) Statistical Analysis Technique

There was no support for defining analysis techniques in the experimental DSL. As the choice of the statistical test depends on the experiment design (DoE), we extended ExpDSLv1 to allow the selection of the statistical analysis technique from a list. Although this information is currently used only to document the experiment, our workflow engine is being extended to support the automatic application and analysis of the statistical tests of the experiments after the collection of the data of interest during the analysis phase. Fig. 2 – (D) shows the statistical analysis techniques chosen for the “CFT” experiment: McNemar to test hypothesis H1, and Wilcoxon to test hypotheses H2 e H3. The current list of possible techniques for testing hypotheses is: Chi-2, Binomial Test, t-test, F-test, McNemar Test, Mann-Whitney, Paired t-test, Wilcoxon, Sign test, ANOVA, Kruskal-Wallis and, for any other test, “Others”. More than one type can be specified for each hypothesis.

#### C. Expressiveness Results

This section describes the analysis of the expressiveness criterion. We have mapped each ExpDSLv1 element/construction regarding the domain concept in order to evaluate if each language construction is associated to only one distinct concept in the domain.

For ExpDSLv1 we observed that the *ArtifactMetric* element was used to model metrics related to artifacts, questionnaires, and user data (data collected during the task/activity execution). This means that distinct concepts in the domain were expressed using the same abstraction of the DSL. In order to overcome this limitation, we created the *DataMetric* element in ExpDSLv2, which provides different ways to specify these three domain concepts: metric related to artifacts, questionnaires, and collected data. Fig. 2 - (G) shows a metric modeled as a *collectedData* element in ExpDSLv2. This *defectNumber* variable is referenced in each activity where the data have to be collected. It means that during the experiment process execution the user will be asked to inform this value during that specific activity.

We also observed an expressiveness limitation in ExpDSLv2 related to the language extension that defines the “Other” type to refer any other type of Design of Experiment (DoE) besides CRD, CRBD, and Latin Square. It has improved the completeness of the language (Section III.B), but on the other hand, it affects the expressiveness of the DSL, because this element can be used to express different concepts of the domain (factorial design, quasi-experimental designs, within subjects design, and so on). In order to improve the language coverage, we opted to accept this expressiveness decreasing because it allows the specification and guided execution of a higher number of experiments. The same problem of expressiveness happens to the *AnalysisTechniqueType* element of ExpDSLv2, which also has the option to the “Other” type.

#### IV. THREATS TO VALIDITY

One threat to the validity of this study is related to the choice of the experiments to be modeled. This choice defines for which types of experiments the conclusions are valid, restricting the extent to which the results can be generalized. We control this threat by selecting real experiments from different areas and with different DoE.

Another threat to validity of this study is called *Reliability* [1], which is concerned with to what extent the data and the analysis are dependent on the specific researchers. We control this threat by modeling and validating the experiments models with experimental software engineering researchers from Fraunhofer IESE and Universidad Politecnica de Madrid (UPM), which were not involved in the DSL development.

#### V. RELATED WORK

Whereas some existing research work explores the development of automated environments to support the conduction of controlled experiments [16], most of them do not describe in details how the specification of the different experimental aspects is addressed. Only a few research works have presented ways to formalize experiments in software engineering. Garcia et al. [10] present an ontology to describe experiments that have predicates to generate a plan for the experiment. Siy and Wu [11] present an ontology that allows defining an experiment and checking some constraints regarding validity threats to the experimental design, which are extracted from a set of experiments. Cartaxo et al. [9] present a graphic domain specific language to model experiments and generate a textual plan of it. However, all these research works do not provide an environment or tool support that allow interpreting and executing the controlled experiments based on their specifications. In addition, they have not conducted empirical studies to evaluate the completeness and expressivity of their specification mechanism – ontology or DSLs – through the modeling of different experiments, as we have presented in this paper.

#### VI. CONCLUSIONS AND FUTURE WORK

This paper presents an empirical study of evaluation and improvement of a DSL that supports the formalization of controlled experiments. In our study, several experiments reported by the software engineering community were specified using the DSL. These specifications were evaluated against the completeness and expressiveness criteria. The results of our study demonstrated the value of the approach, but on the other hand exposed a series of improvement opportunities. These improvements were discussed and addressed in a new version of the domain-specific language also evaluated in this paper.

As part of future work of this research, we are preparing two new studies to be conducted: (i) the first one is a survey that will be performed and applied to experts from the experimental software engineering community to collect feedback about the DSL; and (ii) the other one involves the usage of the improved version of the experimental DSL to conduct controlled experiments by other research groups in order to assessing the usability and performance of the approach. Finally, we also plan to extend the study presented in this paper to consider other existing criteria adopted in DSL evaluations [13].

#### ACKNOWLEDGMENTS

This study is supported by the program “Ciência sem Fronteiras”, a joint initiative from Ministério da Ciência, Tecnologia e Inovação (MCTI) and Ministério da Educação (MEC) of Brazil, through CNPq and CAPES. It is partially supported by the National Institute of Science and Technology for Software Engineering (INES), funded by CNPq, grants 573964/2008-4 and 552645/2011-7, and by FAPERN, CETENE and CAPES/PROAP.

#### REFERENCES

- 1 Wohlin et al. *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, Boston/Dordrecht/London, 2012.
- 2 Juristo, Natalia and Moreno, Ana M. *Basics of Software Engineering Experimentation*. Kluwer Academic Publisher, Madrid, 2010.
- 3 Jedlitschka et al. Reporting Experiments in Software Engineering. In *Guide to Advanced Empirical Software Engineering*. Springer Science+Business Media, 2008.
- 4 Mendonça et al. A Framework for Software Engineering Experimental Replications. *IEEE ICECCS* (2008).
- 5 Hochstein et al. An Environment for Conducting Families of Software Engineering Experiments. *Advances in Computers*, 74 (2008), 175–200.
- 6 Sjøberg et al. Conducting Realistic Experiments in Software Engineering. *ISESE* (2002).
- 7 Travassos et al. An environment to support large scale experimentation in software engineering. *13th IEEE ICECCS* (2008), 193-202.
- 8 Freire et al. A Model-Driven Approach to Specifying and Monitoring Controlled Experiments in Software Engineering. In *PROFES*. LNCS, Paphos, Cyprus, 2013.
- 9 Cartaxo et al. ESEML: empirical software engineering modeling language. (New York 2012), Workshop on Domain-specific modeling.
- 10 Garcia et al. An Ontology for Controlled Experiments on Software Engineering. (San Francisco 2008), SEKE.
- 11 Siy, H. and Wu, Yan. An Ontology to Support Empirical Studies in Software Engineering. (Fullerton 2009), ICCEL.
- 12 Freire, M. A.. A Model-Driven Approach to Formalize and Support Controlled Experiments in Software Engineering. (Baltimore 2013), Proceedings of the ESEM Doctoral Symposium (ESEM 2013).
- 13 Kahraman, G. and Bilgen, S. A framework for qualitative assessment of domain-specific languages. *SoSym Journal* (November 23, 2013), 1-22.
- 14 Acuña et al. How do personality, team processes and task characteristics relate to job satisfaction and software quality? *IST*, 51, 3 (2009), 627–639.
- 15 Aranda et al. In Search of Requirements Analyst Characteristics that Influence Requirements Elicitation Effectiveness: a Quasi-experiment. (Madrid 2012), INTEAMSE.
- 16 Freire et al. Automated Support for Controlled Experiments in Software Engineering: A Systematic Review. In SEKE (Boston/USA 2013).
- 17 Accioly et al. Comparing Two Black-box Testing Strategies for Software Product Lines. *SBCARS* (2012).
- 18 Jung et al. A Controlled Experiment on Component Fault Trees. In *Conference on Computer Safety, Reliability and Security (SafeComp)* (Toulouse 2013).
- 19 Basili et al. The Empirical Investigation of Perspective-Based Reading. ( 1996).
- 20 Aleixo et al. Modeling Variabilities from Software Process Lines with Compositional and Annotative Techniques: A Quantitative Study. (Paphos 2013), PROFES.
- 21 Cirilo et al. Configuration Knowledge of Software Product Lines: A Comprehensibility Study. (Porto de Galinhas 2011), Workshop on Variability & composition.

# Identifying Threats to Validity and Control Actions in the Planning Stages of Controlled Experiments

Amadeu Anderlin Neto and Tayana Uchôa Conte

Instituto de Computação  
Universidade Federal do Amazonas (UFAM)  
Manaus, AM, Brazil  
{neto.amadeu, tayana}@icomp.ufam.edu.br

**Abstract**—During the planning phase of an experiment, we must identify threats to validity in order to assess their impact over the analyzed data. Furthermore, actions to address these threats must be defined, if possible, so that we can minimize their impact over the experiment’s results. In this paper, we propose the Threats to Validity Assistant (TVA), which is a tool to assist novice researchers in identifying and addressing threats to validity of controlled experiments in Software Engineering. The TVA was evaluated through a controlled experiment which aimed at measuring the effectiveness and efficiency in identifying threats to validity and actions to address them. Sixteen subjects were asked to identify threats to validity and control actions related to an empirical study reported in literature. Our results indicate that the subjects who used TVA were significantly more effective and efficient than the subjects who did not use it.

**Keywords** – *validity, controlled experiment, empirical study*

## I. INTRODUCTION

Empirical studies have long been applied to provide confidence in assertions about what is true or not true in the Software Engineering (SE) domain [12]. According to Wohlin et al. [14], empirical research is responsible for the maturation of SE.

There are growing numbers of empirical studies to evaluate different SE practices and techniques [9]. New technologies should be compared with existing technologies through experimentation. Thus, we can collect evidence related to the performance of a technology (e.g., benefits, limitations, costs, risks). However, conducting experiments is a complex task [8] because researchers can cause changes in the experiment environment which could influence its outcomes and introduce threats that may compromise its validity.

According to Biffi and Halling [4], every empirical study presents threats to validity (TTVs). These TTVs must be identified and addressed [11]. The identification and mitigation of TTVs are critical activities that require a lot of effort from researchers. TTVs are potential risks that may arise during the planning and execution of empirical studies [14]. Thus, experiments must be carefully planned and executed.

Researchers in Empirical Software Engineering (ESE) have proposed guidelines, checklists, and summaries to support the identification and mitigation of TTVs [7]. However, none of these present relationships between TTVs and control actions (CAs).

Novice researchers have difficulty in identifying and addressing TTVs during the experiment planning. Thus, such problem motivated us to build a conceptual model presenting relationships between TTVs and CAs [1]. The proposed conceptual model shows which CAs can be applied to address a TTV. In addition, it also shows the new TTV that may occur when applying a CA to address a certain TTV.

As a way to facilitate the use of the conceptual model, we developed a tool, the Threats to Validity Assistant (TVA). The main idea is to assist novice researchers in identifying and mitigating TTVs. Therefore, we aim at enabling researchers to: (a) reduce the effect of the TTVs that can compromise the results of empirical studies; and (b) increase the degree of confidence in the obtained conclusions by enhancing the quality of the empirical studies.

The scope of this work focused only in TTVs of controlled experiments. Kampenes [8] defines controlled experiment as “*a randomized experiment or a quasi-experiment in which individuals or teams (the experimental units) conduct one or more software engineering tasks for the sake of comparing different populations, processes, methods, techniques, languages, or tools (the treatments)*”.

In this paper, we present a controlled experiment which evaluated the TVA. This experiment was conducted in order to measure the effectiveness and efficiency in identifying TTVs and CAs. The quantitative results indicated that the experimental group had better performance than the control group.

This paper is organized as follows. Our approach to assist the identification and mitigation of TTVs is described in Section II, while its evaluation follows in Section III. Then, we present the experiment’s results in Section IV and, in Section V, we explore its TTVs. Finally, Section VI concludes the paper and describes future work.

## II. THE APPROACH TO IDENTIFY AND ADDRESS TTVs

We followed a few stages to develop our approach which assists the identification and mitigation of TTVs. In the following subsections, we describe the conceptual model and the TVA tool.

### A. The Conceptual Model

The conceptual model shows TTVs and CAs which were described in experiments reported in literature. We selected

these TTVs and CAs through a Systematic Literature Review (SLR). The steps for performing the SLR are described in [1].

We organized the conceptual model as follows. First, we grouped the TTVs with their respective CAs. After that, we grouped only TTVs and CAs which present trade-off. The trade-off is characterized when a CA is applied in order to control a TTV and, by applying such CA, new TTVs may arise.

Figure 1 shows part of the conceptual model. The codes presented in each entity correspond to the ID of the TTV or CAs. These codes follow the structure: ValidityType-TypeOfEntity-Number. For instance, EXT-T01 means the entity is the first (01) TTV (T) to external validity (EXT) identified within the results from the SLR. This TTVs can be controlled by CA EXT-C21 (in which C means control). However, this CA may cause TTV COS-T03 (in which COS means conclusion validity). In order to address this TTV, COS-C04 can be applied. Nevertheless, CA COS-C04 may cause TTV EXT-T01.

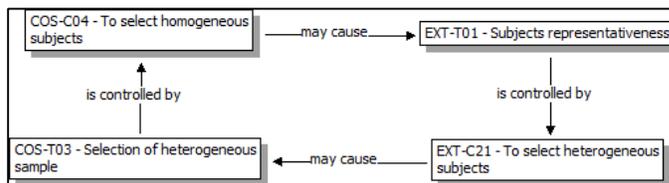


Figure 1. Trade-off between threats to external and conclusion validities.

As shown in Figure 1, there is a trade-off between validity types [3]. When increasing one type, another type may be decreased [14]. The researchers are responsible for prioritizing the validity types according to their needs. Depending on the experiment, some TTVs are more critical than others [2].

The conceptual model allows a visualization of the ways to mitigate TTVs and consequences that may occur when the researcher chooses a CA, since relationships (trade-offs) between TTVs and CAs are mapped. As way to facilitate the use of the conceptual model, we have developed a tool, which is described below.

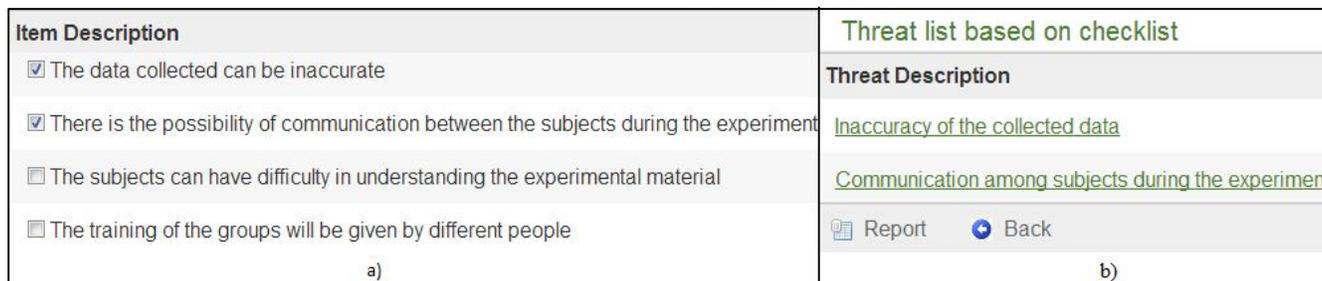


Figure 2. (a) Part of the checklist page in the TVA; (b) Example of the threat list in the TVA.

### B. The Threats to Validity Assistant

The Threats to Validity Assistant (TVA) was developed in order to facilitate the use of the conceptual model. We used the Grails<sup>1</sup> framework and the MySQL database to develop the tool. All relationships presented in the conceptual model were mapped to the database within the TVA.

From the list of TTVs obtained from SLR, we derived a checklist. The checklist contains 65 items which aim to capture the experiment context and identify TTVs that may occur. Such items are related to one or more TTVs. Likewise, the TTVs are related to one or more items. Thus, there are relationships between items and TTVs.

Besides the relationships between items and TTVs, there are two relationships between TTVs and CAs. The first one is a relationship in which a CA can be applied to address a TTV. Each TTV can have none, one or more CAs. Otherwise, a CA can control one or more TTVs. The second relationship characterizes the trade-offs. A new TTV may arise when a specific CA is applied to control a TTV. Thus, one CA can cause none, one or more TTVs.

Figure 2a shows part of the checklist page where we have selected the items “The data collected can be inaccurate” and “There is the possibility of communication between the subjects during the experiment”. Once the researchers fill in

the checklist, the next screen of the TVA shows the list of possible TTVs. For example, Figure 2b shows a list of TTVs which was produced according to the selected items in the previous page (Figure 2a).

Researchers may select one or more CAs to address a TTV, or even none, according to the purpose of the experiment. For instance, Figure 3a shows the CAs to address the TTV “Communication among subjects during the experiment”. When saving the selected CAs, if a CA may cause a new TTV, a warning message appears for the researcher. If the researcher confirms the use of the CA, the new TTV will be included in the list of TTVs on the previous screen (Figure 2b). In our example, we selected “Introduce observers in experimental environment” to address the TTV “Communication among subjects during the experiment”. However, this CA can cause the TTVs “Experimental environment representativeness” and “Subjects act differently when being observed”. Thus, these TTVs will be included in the TTVs list as shown in Figure 3b.

### III. EXPERIMENT DESCRIPTION

We have designed and executed a controlled experiment to answer the following research question: “Is the identification of TTVs and CAs more efficient and effective when the TVA is used”? To address this question, we have conducted a controlled experiment in order to measure the effectiveness and efficiency in identifying TTVs and CAs, with or without the use of the TVA.

<sup>1</sup> <http://www.grails.org/>

Actions to address Communication among subjects during the experiment	Threat list based on checklist
<b>Action Description</b>	<b>Threat Description</b>
<input type="checkbox"/> Distribute subjects with enough space between them	<a href="#">Inaccuracy of the collected data</a>
<input checked="" type="checkbox"/> Introduce observers in experimental environment	<a href="#">Communication among subjects during the experiment</a>
<input type="checkbox"/> Request that subjects do not talk during the experiment	<a href="#">Experimental environment representativeness</a>
<input type="checkbox"/> Do not allow sharing of material during the experiment	<a href="#">Subjects act differently when being observed</a>
<input type="button" value="Save"/> <input type="button" value="Remove Actions"/> <input type="button" value="Back"/>	<input type="button" value="Report"/> <input type="button" value="Back"/>
a)	b)

Figure 3. (a) Example of the control actions list in the TVA; (b) Example of the new threats list after being modified by the researcher's actions.

The definition, the planning (context, variables, hypotheses, selection of subjects, design, and instrumentation) and the operation of the experiment are explained in the following subsections.

#### A. Experiment Definition

The experiment was defined based on the template proposed in previous work [14] as:

**Analyze** Threat to Validity Assistant (TVA)  
**For the purpose of** characterize  
**With respect to** its effectiveness and efficiency  
**From the point of view** of the researchers  
**In the context of** the identification of TTVs and CAs by students from an Empirical Software Engineering course.

#### B. Context Selection

The experiment context was an Empirical Software Engineering (ESE) course offered at UFAM (Federal University of Amazonas) at the north of Brazil. Although participating in the experiment is a mandatory part of the course, the subjects could request that their data were removed from the analysis. The subjects who agreed in participating in the experiment were 10 senior-level (fourth-year) undergraduate Computer Science students, and 6 graduated Computer Science students. The senior-level graduated students enrolled in the course aiming to be prepared at a masters or doctorate degree (more details about the subjects are presented in Subsection E).

The controlled experiment used in this study was selected from the SLR and was based on the following criteria: (1) to describe more than 10 TTVs; (2) to contain TTVs of the four validity types; (3) to present different context used in the pretest activity; and, (4) to follow the experiment process described by Wohlin et al. [14].

Using these criteria, we selected the controlled experiment described by Madeyski [10] (hereafter referred to as Madeyski's experiment), which contains 23 TTVs and 12 CAs. The context of Madeyski's experiment was related to testing in software systems.

#### C. Variables Selection

The independent variable is the treatment applied by the groups in order to identify and address TTVs. Thus, the

experimental group used the TVA while the control group used the bibliographical material of the Empirical Software Engineering course, mainly the guidelines described by Wohlin et al. [14]. We chose this book since it was the most cited reference (with about 49 citations within the SLR presented in [1]) in empirical studies to identify and address TTVs.

The dependent variables are the effectiveness and efficiency in identifying TTVs and CAs. Effectiveness and efficiency are based on Conte et al. [5], and are defined as follows.

- Effectiveness in identifying **threats to validity**:

*Effectiveness = the ratio between the number of real threats found and the total number of threats known.*

- Effectiveness in identifying **control actions**:

*Effectiveness = the ratio between the number of real actions found and total number of actions known.*

- Efficiency in identifying **threats to validity**:

*Efficiency = the ratio between the number of real threats found and time spent.*

- Efficiency in identifying **control actions**:

*Efficiency = the ratio between the number of real actions found and time spent.*

#### D. Hypotheses

Using the variables described above, we defined the following null hypotheses:

- $H_{01}$  – there is no difference in the effectiveness indicator when identifying TTVs with or without using the TVA;
- $H_{02}$  – there is no difference in the effectiveness indicator when identifying CAs with or without using the TVA;
- $H_{03}$  – there is no difference in the efficiency indicator when identifying TTVs with or without using the TVA;
- $H_{04}$  – there is no difference in the efficiency indicator when identifying CAs with or without using the TVA.

### E. Selection of Subjects

The selected subjects were 16 students taking the ESE course. All subjects had lessons about the experiment process described by Wohlin et al. [14]. In addition, all subjects made presentations about validity evaluation in SE experiments and validity types. All subjects signed a consent form and filled out a characterization form that measured their expertise with controlled experiments.

We also applied a pretest activity to measure more directly the subjects' expertise. The pretest activity consisted in the identification of TTVs and CAs from a paper reporting a controlled experiment. In this activity, the subjects did not use any support. We selected the controlled experiment described by Thelin et al. [13]. This paper was retrieved by the SLR [1], which described a controlled experiment in the context of software inspection.

The pretest activity was conducted simultaneously with all subjects in a classroom. The pretest activity consisted in identifying and addressing TTVs without any support. The subjects had 90 minutes to perform the task. At the end, a researcher had analyzed the answers and assigned a grade to the results of the pretest activity.

Based on the results of the characterization form and the pretest activity, the subjects were divided into groups of: low (L), medium-low (ML), and medium (M) knowledge. Low knowledge means that a person identified one to four real TTVs and CAs, and does not have experience. Medium-low knowledge means that a person identified five to eight real TTVs and CAs, and has little experience (i. e., participated in an experiment as subject). Finally, medium knowledge means that a person identified more than eight real TTVs and CAs, and has little experience. In order to reduce the bias of having more experienced subjects in one or another treatment (with or without using the TVA), we equally and randomly distributed the subjects, balancing both groups.

### F. Design of the experiment

The design is one factor, with two treatments. The assignment of the treatments was randomized. Each group consisted of 8 subjects and each used only one treatment, either the TVA or the guidelines proposed by Wohlin et al. [14].

### G. Instrumentation

The instruments used in this experiment were: characterization and consent forms, guidelines described by Wohlin et al. [14], data collection form (control group, since the TVA automates such process), TVA, instructions to use TVA (presented on the initial screen of the tool), follow-up questionnaire (experimental group, since it was used to gather improvement opportunities), excerpts from Madeyski's experiment in the original language (English) and a translated version (Portuguese). These excerpts were: the experiment context, treatments details, metrics details, and experiment planning. The translation was carried out by a researcher and reviewed by another researcher with an advanced English level. All instruments were validated by the authors.

### H. Experiment Operation

The subjects were divided into two classrooms, one for the experimental group and another for the control group. All subjects had up to 90 minutes to carry out the experiment. During the experiment, the control group had to fill out the data collection form as they found TTVs or CAs. In addition, the control group used the checklist presented in [14]. On the other hand, the experimental group used the TVA. When the subjects from the experimental group finished the task, they had to fill out the follow-up questionnaire.

After finishing the experiment, we created a list containing all different TTVs without duplicated ones. Likewise, we created a unique CAs list. Both lists were based on TTVs and CAs described in the validity section of Madeyski's experiment. A total of 23 TTVs and 12 CAs were found in the TTVs section of the paper. However, there were TTVs that were not described in the validity section. Thus, an independent experienced researcher evaluated Madeyski's experiment and found 23 new TTVs and 29 new CAs. Therefore, a total of 46 TTVs and 41 CAs were identified. Other TTVs and CAs that were found by the subjects and that were not already included in the lists, were classified as false positives (in order words, they were not present in Madeyski's experiment). In the next section, we present the results of the controlled experiment.

## IV. RESULTS

We obtained the quantitative data from the data collection form resulting from the experimental task. After creating the list of TTVs and the list of CAs, we counted the number of TTVs and CAs. We also considered the time used by each subject to carry out the experimental task. Table I presents both the results per subject and per treatment.

We have performed a statistical analysis using the statistical tool SPSS v 20.0.0 and  $\alpha = 0.05$ . This choice of statistical significance was motivated by the small sample size used in this experiment [6]. In order to compare the effectiveness and efficiency of both samples, we used the Mann-Whitney non-parametrical statistic method. This test is equivalent of the t-Student parametrical test (for further information, please see [14]). It was used because we had two groups to compare, different subjects in each condition, and no assumption about the data distribution. We also used the boxplots graph to facilitate the visualization of results.

The boxplot graph which compares the effectiveness indicator for the identification of TTVs and CAs is shown in Figure 4. When analyzing the graph, we can see that the mean from the experimental group is higher than the mean from the control group for both indicators. Thus, the group that used the TVA was more effective than the group that did not use it. Also, the comparison using the Mann-Whitney statistic method showed that there was significant difference between the groups ( $p = 0.001$  – for TTVs; and,  $p = 0.005$  for CAs). These results suggest that the use of the TVA provided different effectiveness related to the identification of TTVs and CAs in controlled experiments; and that this difference is significant from a statistical point of view. These results reject the null hypothesis  $H_{01}$  and  $H_{02}$ .

TABLE I. QUANTITATIVE RESULTS PER SUBJECT AND TREATMENT.

	Control group (guidelines)								Experimental group (TVA)							
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Expertise Level	M	ML	M	L	ML	ML	ML	ML	L	ML	M	M	ML	ML	ML	ML
Time (min)	90	88	75	45	87	87	80	82	70	60	47	62	53	63	52	40
Total TTVs Found	6	4	7	12	14	8	4	6	18	42	33	50	28	15	35	14
Real TTVs Found	3	3	7	6	8	8	3	4	13	35	22	37	20	12	25	13
False Positive TTVs	3	1	0	6	6	0	1	2	5	7	11	13	8	3	10	1
Effectiveness in Identifying TTVs %	6.52	6.52	15.22	13.04	17.39	17.39	6.52	8.70	28.26	76.09	47.83	80.43	43.48	26.09	54.35	28.26
Efficiency in Identifying TTVs	2.00	2.05	5.60	8.00	5.52	5.52	2.25	2.93	11.14	35.00	28.09	35.81	22.64	11.43	28.85	19.50
Total CAS Found	6	4	7	12	10	8	4	5	10	32	17	49	18	6	20	8
Real CAS Found	1	3	5	3	6	8	1	2	5	26	12	29	12	5	18	6
False Positive CAS	5	1	2	9	4	0	3	3	5	6	5	20	6	1	2	2
Effectiveness in Identifying CAS %	2.44	7.32	12.20	7.32	14.63	19.51	2.44	4.88	12.20	63.41	29.27	70.73	29.27	12.20	43.90	14.63
Efficiency in Identifying CAS	0.67	2.05	4.00	4.00	4.14	5.52	0.75	1.46	4.29	26.00	15.32	28.06	13.58	4.76	20.77	9.00

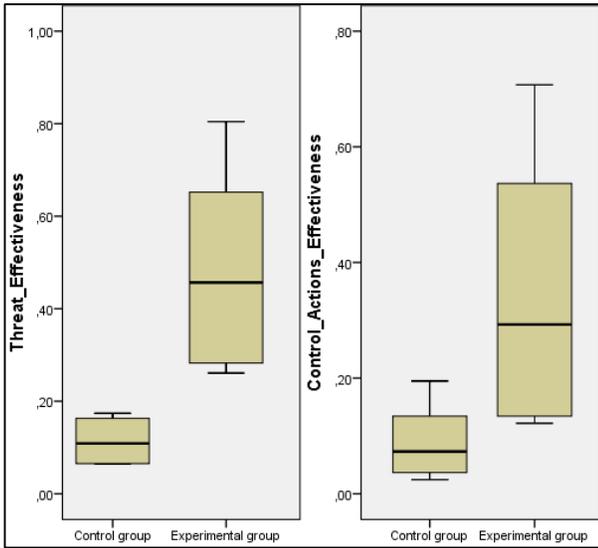


Figure 4. Effectiveness indicator for identification of threats and actions.

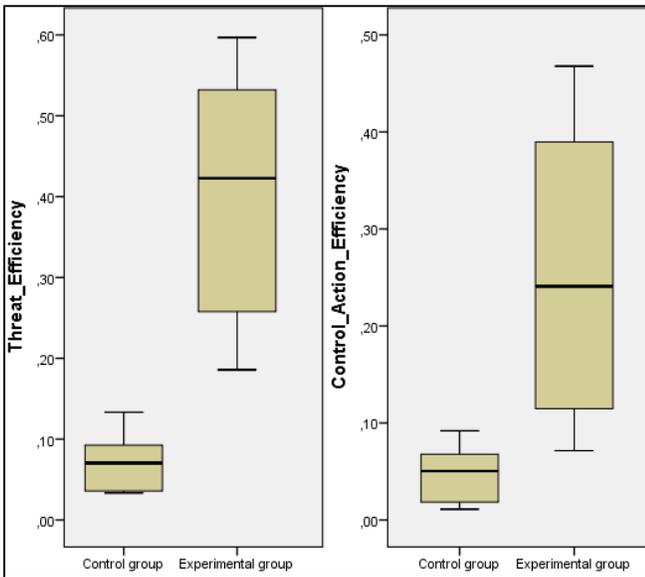


Figure 5. Efficiency indicator for identification of threats and actions.

The boxplot graph which compares the efficiency indicator for the identification of TTVs and CAs is shown in Figure 5. According to the graph, the experimental group was more efficient than the control group, finding more TTVs and CAs in lesser time. Also, the Mann-Whitney statistical test shows that the difference between the groups is statistically significant ( $p = 0.001$  – for both indicators). Thus, the use of the TVA provides better efficiency regarding the identification of TTVs and CAs. These results reject the null hypothesis  $H_{03}$  and  $H_{04}$ .

## V. THREATS TO VALIDITY

The following subsections present the TTVs considered in this controlled experiment. These TTVs have been grouped in four main categories: internal, external, conclusion and construct.

### A. Internal Validity

- *Individual differences among the subjects.* In order to address this TTV, we divided the subjects in balanced groups, according to the results from the characterization form and the pretest activity.
- *Language of the study object was different from the native language of the subjects.* In order to address this TTV, we translated the study object to the native language of the subjects. The translation was done by a researcher and reviewed by another researcher with high proficiency in English. Both documents (original and translated) were given to the subjects.

### B. External Validity

- *Representativeness of subjects.* The students are representative of the target population (novice researchers), but the results cannot be generalized. This experiment should be replicated for other samples and contexts.
- *Representativeness of artifact.* In order to avoid this TTV, we selected an experiment reported in literature. However, new experiments should be conducted with study objects from other contexts.

### C. Conclusion Validity

- *The small number of data points.* However, small sample sizes are a known problem in Software Engineering which is difficult to overcome [5]. Thus, the data extracted from this controlled experiment can only be considered indicators and not conclusive.
- *Heterogeneous characteristic of the subjects.* This TTV was not considered because all students do not have experience in the planning of controlled experiments. Thus, the sample was homogeneous.

### D. Construct Validity

- *Biased judgment of real TTVs and CAs.* In order to mitigate this TTV, an independent researcher evaluated Madeyski's experiment and judged which were real TTVs or CAs. However, only one independent researcher might not be enough to classify TTVs and CAs.
- *Only one study object was used.* Therefore, the results may depend on the study object used. In order to mitigate this TTV, we plan to conduct new controlled experiments with different study objects, to enhance the validity of our results.
- *Subjects may behave differently, because they were observed during the experiment execution.* In order to avoid this TTV, subjects were observed during the pretest activity. Thus, it is expected that the subjects adapted to the environment. However, the observation impacts cannot be measured.

In summary, we conclude that the existing TTVs were not considered large in this study. Also, we carried out CAs in order to reduce their impact.

## VI. CONCLUSION AND FUTURE WORK

This paper has presented a controlled experiment to answer the following research question: "*Is the identification of TTVs and CAs more efficient and effective when the TVA is used?*" In order to answer this question we have measured the results of the TVA in terms of effectiveness and efficiency.

Results showed that the use of the TVA provided better effectiveness and efficiency related to the identification of TTVs and CAs. The quantitative analysis using the Mann-Whitney statistic method showed that the effectiveness and efficiency indicators are better, when using the TVA, and that there is significant statistical difference. Due to the small sample used and the number of carried out experiments (only one), these results are not conclusive but indicate that the use of the TVA enhance the effectiveness and efficiency of the identification of TTVs and their CAs.

We intend to perform new empirical studies to validate the results, including subjects with high experience in planning controlled experiments. Thus, we will obtain evidence if the

TVA is useful for experienced researchers, while evaluating if experienced researchers agree with our categorization and relationships of TTVs and CAs.

We hope that our findings will be useful in the promotion and improvement of the current practice of experiments planning. We also hope that the proposed tool aids researchers in the validity evaluation of their empirical studies, improving the confidence in their results and reducing the effort during planning stages.

### ACKNOWLEDGMENT

We would like to acknowledge the financial support granted by FAPEAM process PAPE 032/2013.

### REFERENCES

- [1] A. Anderlin-Neto, and T. Conte, "Threats to validity and their control actions – results of a systematic literature review", TR-USES-2014-0002, UFAM, March 2014, Available at <http://uses.icomp.ufam.edu.br/attachments/article/42/TR-USES-2014-0002.pdf>.
- [2] C. Andersson, T. Thelin, P. Runeson, and N. Dzamshvili, "An experimental evaluation of inspection and testing for detection of design faults", ISESE, pp. 174-184, 2003.
- [3] E. Arisholm, H. Gallis, T. Dyba, and D. I. K. Sjoberg, "Evaluating pair programming with respect to system complexity and programmer expertise", TSE 33 (2), pp. 65-86, 2007.
- [4] S. Biffl, and M. Halling, "Investigating the defect detection effectiveness and cost benefit of nominal inspection teams", TSE 29 (5), pp. 385-397, 2003.
- [5] T. Conte, J. Massollar, E. Mendes, and G. H. Travassos, "Usability evaluation based on web design perspectives", ESEM, pp. 146-155, 2007.
- [6] T. Dyba, V. B. Kampenes, and D. I. K. Sjoberg, "A systematic review of statistical power in Software Engineering experiments", IST 48 (8), pp. 745-755, 2006.
- [7] R. Feldt, and A. Magazinius, "Validity threats in empirical Software Engineering research – an initial survey", SEKE, pp. 374-379, 2010.
- [8] V. B. Kampenes, "Quality of design, analysis and reporting of Software Engineering experiments, a systematic review", Doctoral Thesis, University of Oslo, 2007.
- [9] S. MacDonell, M. Shepperd, B. Kitchenham, and E. Mendes, "How reliable are systematic reviews in empirical software engineering?", TSE 36 (5), pp. 676-687, 2010.
- [10] L. Madeyski, "The impact of Test-First programming on branch coverage and mutation score indicator of unit tests – an experiment", IST 52 (2), pp. 169-184, 2010.
- [11] J. C. Maldonado, J. Carver, F. Shull, S. Fabbri, E. Dória, L. Martimiano, M. Mendonça, and V. Basili, "Perspective-Based Reading: a replicated experiment focused on individual reviewer effectiveness", EMSE 11 (1), pp. 119-142, 2006.
- [12] F. Shull, D. Cruzes, V. Basili, and M. Mendonça, "Simulating families of studies to build confidence in defect hypotheses", IST 47 (15), pp. 1019-1032, 2005.
- [13] T. Thelin, P. Runeson, and C. Wöhlin, "An experimental comparison of Usage-Based and Checklist-Based Reading", TSE 29 (8), pp. 687-704, 2003.
- [14] C. Wohlin, P. Runeson, M. Host, M. C. Ohlsson, B. Regnell, and A. Wessl, "Experimentation in Software Engineering", Kluwer Academic Publishers, 2012.

# Case-based Reasoning for Experience-based Collaborative Risk Management

Nielsen L. R. Machado, Luís A. L. Silva, Lisandra M. Fontoura  
Programa de Pós-Graduação em Informática  
Universidade Federal de Santa Maria – UFSM  
Santa Maria, Brazil  
{nielsenmachado, silva.luisalvaro, lisandramf}@gmail.com

John A. Campbell  
Department of Computer Science  
University College London – UCL  
London, UK  
j.campbell@cs.ucl.ac.uk

**Abstract** - In a collaborative risk management scenario, project stakeholders often need natural forms of recording and reusing past risk management experiences so that they could better assess whether there are threats to the goals of new projects. The contribution of this paper is to propose an enhanced case-based reasoning (CBR) approach to support project participants to exploit such experiences which are here expressed as collaborative risk management discussion cases. The paper shows how these debates are structured through the exploitation of a dialogue game protocol for risk management. Then, it discusses how users can utilize queries based on facts and arguments so that past risk discussion cases could be retrieved from a case base. Attention is also given to case-based explanation templates, which are relevant for the understanding of key moves of argumentation in debate trees recorded in such enhanced cases retrieved. To demonstrate the practical utility of this approach, a case study involving the collaborative experience-based risk management of a software project is discussed.

**Keywords-component:** *Case-Based Reasoning; Collaborative Risk Management Tool; Dialogue Game; Argumentation.*

## I. INTRODUCTION

Management of risks in new software projects is most effective if one can draw on concrete instances of dealing with risks in past projects. Once such experiences are collected and represented systematically in a reusable memory, project stakeholders involved in collaborative risk discussion tasks have the means of constructing a better assessment of the risks to their project' goals, considering consequences in case these risks occur and, in general, taking actions to guarantee that the associated threats could be controlled more effectively. The overall goal is to fully exploit these past experiences in order to not only learn relevant lessons from past risk management episodes but also to avoid the repetition of past risk management mistakes in current projects.

Some degree of experience-based risk management is achieved through the exploitation of different Artificial Intelligence (AI) techniques (e.g. [1, 2]). These kinds of approach allow one to learn from data of past software projects so as to better understand the relevance of risks in new projects. Moreover, security risk management methodologies, such as Magerit [3] (in step 2: threads), for instance, discuss that past experiences should be exploited in the identification of threads to the assets of a project. In this context, Case-Based Reason-

ing (CBR) [4, 5] is an AI problem-solving paradigm which emphasizes the role of past experiences in the solution of future problems. By its nature, CBR is well adapted to the capture and reuse of factual and/or prescriptive information along with a solution for a risk management problem. Moreover, if stakeholders debate or argue about aspects of their project, CBR can easily accommodate any knowledge that these arguments express explicitly.

The initial claim of our approach for risk management is that collaborative debates can be expressed as an argumentation process [6]. Argumentation is understood as a process of dialogue in which different project participants are able to present different kinds of arguments when stating or justifying the assessment of risk management issues in a collaborative risk management discussion. In this dialectical context, we capture and organize such risk discussions in the body of enhanced cases for CBR according to a dialogue protocol, or “dialogue game” [7, 8]. This dialogue game is expressed as a set of moves of argumentation (e.g. locution acts such as propose, ask, inform, etc) that occur when multiple agents (human and/or computational agents) are engaged in a debate. As presented in our initial paper [9], a dialogue game for risk management was designed to mediate the interaction of project stakeholders when these agents develop collaborative risk management tasks. Our approach captures and makes use of a dialogue form (the dialogue, or set of locution acts) which people working on risk management etc. use naturally in their own discussions and debates among themselves, i.e. it respects the kind of knowledge representation that is informally closest to what they actually use in practice.

In this paper, we discuss a CBR extension which approaches not only factual project characteristics but also structural and type details of the arguments observed in past stakeholders' discussions about how to construct solutions (e.g. risk management plans). We also discuss how project stakeholders can obtain personalized views of the results obtained when different kinds of queries are executed. In particular, the personalization resources proposed here rely on the exploitation of a set of case-based explanation templates, in a scenario in which CBR becomes a form of explanation-based reasoning [10]. In our project, these templates capture meaningful combinations of moves of dialogue, or locution acts, that are exploited in risk management discussions recorded in enhanced

---

*We gratefully acknowledge financial support from CAPES-Brazil.*

cases. Such explanation templates are relevant for project stakeholders when inspecting and, consequently, forming an understanding of key risk management steps of argumentation advanced in risk debates. We also describe how these case-based query and explanation resources can be exploited by different project stakeholders when a web-based Risk Discussion System (RD System) is used. To illustrate our approach, we describe a case study in which a risk management project is analyzed by different project participants.

This paper is organized as follows: Section II presents some background information about risk management, argumentation and CBR. Section III discusses a) the formation of enhanced cases through facts and arguments, b) the alternative query forms utilized to find past cases so as to support users when deliberating solutions for new risk management problems and c) the exploitation of explanation templates for improving the users' understanding of risk management discussion cases. Section IV presents a study case to illustrate the utility of our approach. Section V shortly reviews the paper proposals and presents some future steps in our project.

## II. CBR, ARGUMENTATION AND EXPLANATION IN THE CONTEXT OF COLLABORATIVE RISK MANAGEMENT

Risk management requires the development of activities for the identification of possible problems and their causes, the analysis of risk probability and impact resulting in the prioritization of more critical risks, the construction of plans for the mitigation or even elimination of prioritized risks and the execution and monitoring of risk management plans [11]. In this context, the CBR paradigm for problem-solving supports the solution of new problems through the systematic collection and representation of cases in a case base, the retrieval of similar cases to a given problem situation, the reuse of past case solutions retrieved in the solution of new problems, the adjustment of these solutions to the context of these new problems when necessary and the retention of experiences of problem-solving in the case base so that the associated software system can improve its problem-solving capabilities [4, 5].

According to Kolodner [5], a case captures a contextualized piece of knowledge representing an experience that teaches a lesson fundamental to achieving the goals of the reasoner. Intuitively, it is possible to observe that "lessons" are likely to be offered by domain users in terms of different kinds of arguments, although such arguments still do not have places in traditional frameworks for CBR. Traditionally, only the key factual characteristics of a problem situation are utilized to index the information content of cases for typical CBR applications. As in other machine learning contexts, these case properties are usually encoded as a vector of attributes and values. Then, when users want to retrieve such cases from a case base, they make use of similarity-based forms of computing the distance between current and past cases [4, 5]. Although similarity could be computationally evaluated in different forms, the weighted Euclidian distance computation is the simplest forms of distance assessment in different CBR applications (see [12] for other forms of encoding case features and computing distances between case-like entities). In a CBR application, the overall idea is to retrieve the most similar cases to a given problem situation by stating a query which

represents the current problem, or target case, to be solved. That is because CBR is built on the hypothesis that "similar problems have similar solutions", which is a hypothesis that can be exploited in different application problems.

In addition to standard factual attributes, cases can also capture arguments and argumentation processes. These kinds of augmented cases are usually found in studies of the nature of argumentation in legal applications of CBR (e.g. [13]), although frameworks involving the use of cases in assisting the generation and evaluation of arguments have been reported in other application scenarios (e.g. [14]). Alternative argumentation models [8] have also been exploited in the development of intelligent systems for supporting the solution of problems in these argumentation applications. In this context, as described in [9], an argumentation process in which a debate is developed among different agents can be organized in the body of a case through the exploitation of a dialogue game model [7, 8]. These dialogue protocols are mainly defined by a set of locution acts, which are typical moves of speech used by these agents. In addition to such set of locution acts, these protocols are described by rules expressing how these locutions can be combined (e.g. which locutions can be used as responses to certain locutions). The interaction of two or more project participants involved in a debate can be mediated by such communication protocols.

According to [6], argumentation and explanation are interlinked activities when one is deliberating a solution for a problem. Among other reasons, they are often combined because certain users lack relevant knowledge to understand arguments being posed in a problem-solving situation, as well as because users are likely to pose arguments which express different considerations and explanations with respect to a decision. In effect, when pro and con arguments of a solution for a case-based problem are presented to users, they can review these arguments in order to form an understanding of the rationale behind certain decisions, as exploited in design problems [15], for instance. In practice, just the exploitation of cases that are similar to a current problem situation is the most effective form of explaining a solution in different application domains. It is also relevant to observe that explanations that are constructed on the grounds of past experiences are likely to be more convincing than standard rule-based explanation, as experimental results described in [16] show. In summary, the overall idea is to collect and record users' arguments systematically in augmented cases for CBR so that one can exploit the content of these cases to provide additional layers of explanation on top of these argumentative structures.

## III. A ENHANCED CBR APPROACH FOR COLLABORATIVE RISK MANAGEMENT

The exploitation of a dialogue game for the development of risk management tasks permits project stakeholders to discuss collaboratively the risks of a software project. In such a debate, project participants can express their opinion about causes and effects of risks, in addition to designing plans so that risks identified can be minimized. Through the exploitation of this dialogue protocol, participants in risk debates can reach a consensus while deciding which actions shall be taken in each risk situation. In this context, the CBR resources of our

RD System permit that these users examine and reuse concrete problem-solving experiences of risk management in the determination of risks in current projects.

*A. How enhanced cases are formed when collaborative risk management tasks are developed*

Concrete cases of collaborative risk management are recorded in a reusable risk management memory, which takes the form of a case base in the RD System. As it is usual in CBR, we represent the cases of this case base through a list of factual properties. To identify which properties we needed to utilize, we started with the observation that recent software projects are been characterized in the agile and planned scenarios. Depending of the context that a project is developed, each one of these methodologies has their strong and weak points, which allows one to realize that contextual factors have a crucial role in the characterization of a project. Authors as Boehm and Turner [17] describe an approach which is based on risks for balancing the utilization of agile methods and planned methods in a project. Among others, attributes such as size, criticality, dynamism, skills of the team and culture are utilized in the characterization of a project. Additional attributes can be utilized so as to distinguish risk management cases obtained when different kinds of companies are considered.

**Risk assessment on the localization project**

**Characteristics of the Project:**

<b>Team Size:</b> Very Small	<b>Criticality:</b> Money Loss
<b>Team Distribution:</b> Collocated	...
<b>Rate of Change:</b> Low	

**Discussion:**

**Start discussion:** Risk assessment of a localization algorithm involving sensors - Programmer 1

**Propose risk:** The final product will not correspond to client expectations - Programmer 1

**Ask:** Are the project requirements defined properly? - Programmer 1

**Inform:** The requirements are okay but the time to develop this project is not enough - Technical Manager

**Propose consequence:** An unhappy client may try to find other people to develop this project; our competitors for instance - Technical Manager

**Argument pro:** If the client of this project is not happy with us, we may lose the project funds and the client - Programmer 1

**Propose probability:** High - Programmer 3

**Propose impact:** High - Programmer 1

...

**Propose plan:** We need to record the changes of requirements more formally - Programmer 2

**Summarize:** Requirements that are defined correctly, recording of requirement changes, new meetings in case the client is not happy and the frequent presentation of results to the client - Programmer 1

**Propose risk:** The timetable of the project is not real - Programmer 1

**Propose probability:** Medium - Technical Manager

...

**Propose plan:** We should discuss again the project timetable and the project requirements - Programmer 2

...

Figure 1. An example of a risk management discussion case

In the work of Krutchén [18], a model called Octopus is proposed to characterize projects, where the key factors are: Team Size, Team Distribution, Criticality and Rate of Change. As these contextual factors are important in the area of software processes, they are also relevant for the indexing of enhanced cases. In practice, the risks of a project are dependent of such contextual factors, and the efficacy of actions to pre-

vent them is also influenced by the project context. In addition to factual properties, we make it possible to enhance the body of each risk management case by exploiting argumentation-based characteristics. Integrating CBR and argumentation techniques, these novel case representation characteristics are grounded on the set of arguments that are collected and represented when our dialogue protocol for collaborative risk management [9] is exploited by project stakeholders. To present key locution acts of this dialogue protocol, we can examine a risk discussion case (see Fig. 1) in which different users deliberate risks of a software project. In this argumentation model for capturing arguments in cases, the general format of a single argument is: i) an identification of a locution act from our dialogue protocol, ii) a textual risk management statement (i.e. informal statements of the type that most users in this application domain are able to naturally offer) and iii) an identification of the discussion participant that is advancing such argument. These kinds of argument can be advanced by any discussion participant of a risk management debate which is started with “Propose risk” locutions. These argumentation moves are the root indexing concepts in the risk management discussion tree which is recorded in the body of an enhanced case. To gather information about such risks, a project participant can present question-like statements by using “Ask” locution acts, which can be answered via “Inform” locutions. These inform-like arguments can also be exploited in the presentation of any other contextual information for the development of the debate. Consequences related to the occurrence of risks in a project can also be examined explicitly when our dialogue protocol is used. In doing so, these consequence-like statements are advanced by discussion participants via “Propose consequence” locution acts. At any point of a debate, participants can also pose pro and con arguments in relation to different risk management issues. To promote the critical analysis of a risk management problem situation, our protocol permits participant to advance such arguments through “Argument pro” and “Argument con” locutions. Debate participants can also analyze the probability of a risk occurrence, as well as the impact that a risk is likely to have in a project. When this happens, “Propose probability” and “Propose impact” locution acts can be exploited by project participants. In practice, probability and impact estimates can be presented qualitatively or quantitatively, or both. Once initial steps of risks identification and analysis are completed, risk management plans should be constructed for the most relevant risks, or prioritized risks. In the protocol, risk mitigation, reduction, etc statements, for instance, can be advanced by discussion participants through “Propose plan” locutions. A participant can also summarize different aspects of a risk management discussion, through “Summarize” locutions. In summary, the set of locution acts available in our dialogue protocol for collaborative risk management expedites the debate of different risks in a project and the consequent recording of this discussion as a semi-formal argumentation model in the body of enhanced cases for use in CBR. As described here, this protocol is fully implemented in the RD System.

*B. How risk discussion cases are retrieved from the case base so as to support the solution of new problems*

The systematic recording of collaborative risk management experiences in a reusable memory is a fundamental issue for

an approach which promotes the exploitation of these experiences in the solution of new problems. However, it is also essential to make available to users alternative forms of consultation for this memory. We make it possible through similarity-based queries which can be formed and executed by discussion participants at any moment of a collaborative risk management debate. Such similarity estimates are determined when properties of the current risk management project, or query, are compared with properties recorded on past cases. Based on the enhanced case characteristics proposed, which are i) facts expressing the context of a risk management project and ii) argumentation moves from the discussion of typical risk management issues, our CBR framework allows users to retrieve the K most similar risk discussion cases to a given query. We utilize a standard “K nearest neighbor” algorithm [4, 5], and a weighted Euclidian distance function in the comparison of cases. So far in our project, an equal weighting scheme has been used despite the fact that the RD System permits the adjustment of such weights by users. For the comparison of argumentation information, our similarity algorithm matches textual statements expressed in a query with sentences recorded in past cases. In the PostgreSQL database [19], which is integrated with the RD System, this similarity assessment permits the identification of natural language arguments matching arguments recorded in past cases. In this context, two types of query methods are relevant for the retrieval of past risk management experiences.

The first query method is standard in CBR applications. In it, a user should input the factual properties of the current project (see Fig. 2 – A). An example of such query statement can involve a project which is very large (“Project Size = Very Large”), with many participants (“Team Size = Large”) that are scattered in different locations (“Team Distribution = Geographic Distribution”). In this project, a problem is that the project participants are uncertain about what risks should be considered (i.e. identified in a relevant list for future analysis) in the project and, consequently, what sort of things could fail in the project development. To solve this problem, discussion participants utilize the project properties mentioned as query parameters. When this query is executed, experiences of collaborative risk management involving projects with similar set of properties are likely to be retrieved from the case base. It means that the most similar collaborative risk management cases retrieved will be shaped by risk proposals (i.e. risk statements advanced by means of “Propose Risk”) that can be examine and, if useful, reused in different forms in the risk management discussion of a current project.

The second query method allows project stakeholders to construct queries that are formed by both factual and argumentation characteristics. In doing so, these users can utilize the list of factual project properties, as described in our first query method, and a set of keywords/sentences along with the corresponding locution acts that were advanced in a discussion. A typical example of such query involves the discussion of a current project that aims to develop a new feature in an existing system. So, the “Business Model” attribute of this project is “System Component”. This system also has a “high rate of changes in its requirements”. Because of this problem, the company responsible for the system maintenance is often

losing money (i.e. the “Criticality” attribute of this project is “Money Loss”). In this project, a participant needs to analyze risks regarding “software requirements that are not clearly described”. To reuse past collaborative risk management experiences in the solution of this problem, such participant constructs a query involving both the “Propose risk” and “Propose plan” locution acts. Along with the locution “Propose risk”, this user inputs the sentence: “unclear specification of software requirements”, expressing the kind of risk that this participant wants to review in similar risk management experiences retrieved from the case base (see Fig. 2 – C). This user also inputs the following keyword in the query statement: “software requirements”. This keyword is described along with the “Propose plan” locution act, expressing that the participant wants to review past risk management plans involving this subject. When this query is executed, previous cases of collaborative risk management are retrieved from the case base. In particular, these cases may contain “Propose risk” and “Propose plan” moves of dialogue in which those keywords/sentences mentioned were advanced by discussion participants. As a result, information regarding risk proposals and risk management plans can be reused in different forms as, for instance, in the construction of new arguments to be advanced in the current debate.

### *C. How explanation templates can be utilized by users in the understanding of a case retrieved for a given query*

In our project, project stakeholders are able to exploit a new form of case-based explanation with the purpose of locating and highlighting relevant argumentation information in past risk management cases retrieved for a given query. Among other reasons, these explanatory views of a query result were designed after the fact that argumentation trees representing collaborative risk management discussions might contain a large number of speech moves, which might hinder a user comparison between current and past cases when analyzing a query result. Therefore, we give users access to alternative template forms in order to supplement their retrieval results and, consequently, make easier the steps of reuse of past arguments in the discussion of new risk management problems. In doing so, such a template format provides a standard for the presentation of selected risk management locution moves available in an enhanced case. This explanation is possible since the argumentative structure of enhanced cases already provides an explicit history, or a sort of narrative explanation [10], for the risk management decision-making process. What a template does, in this case, is to offer users ways of highlighting relevant argumentation steps available in these narratives (see Fig. 2 – B).

An explanation template has a name, a textual description of what the template aims to emphasize in a discussion tree (e.g. what needs to be explained, what interests a user), and how it is done through the selection of a set of locution acts belonging to the dialogue protocol used. A template can be created, recorded and adjusted by a knowledge engineer user of the RD System. In a scenario of a template utilization, for instance, the most similar case for a given query is shown to a user. However, it is presented only through the template contents of certain locution acts, e.g. as selected for its relevance to the user in a suitably personalized consultation with the system.

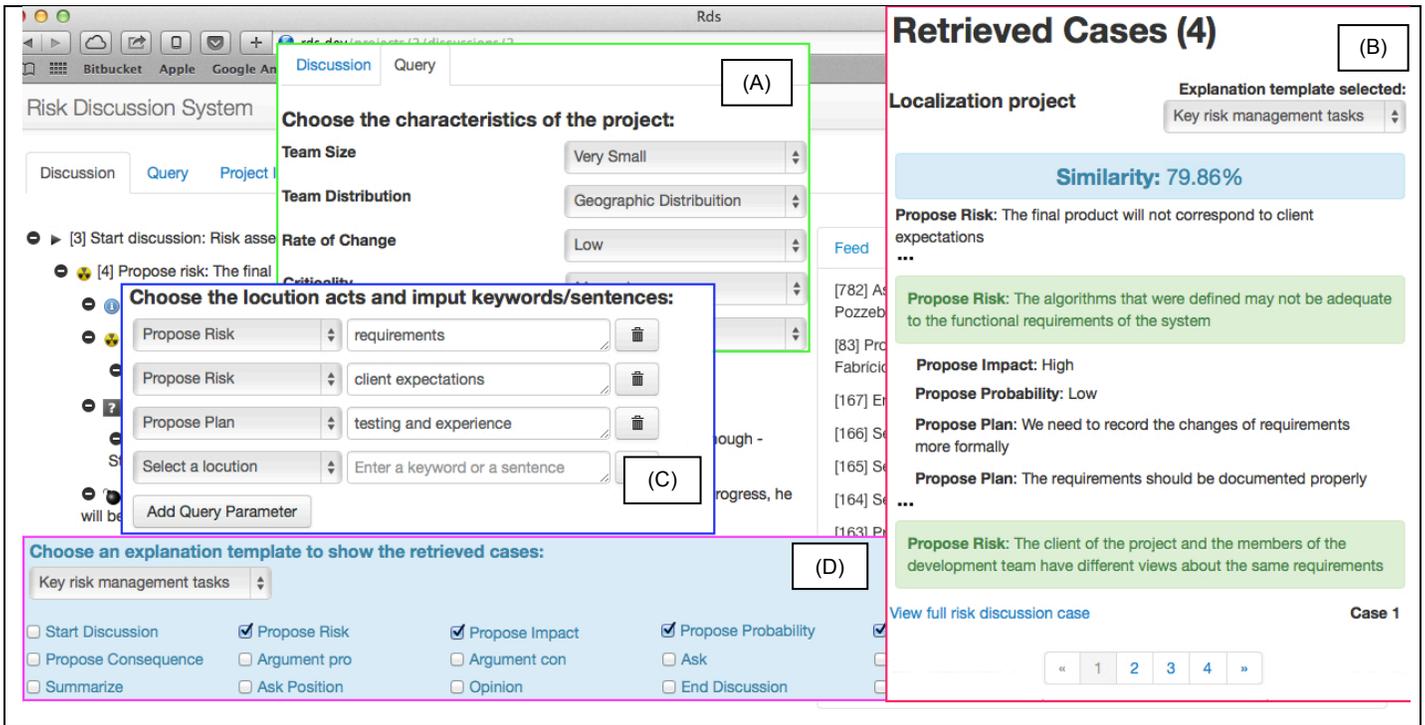


Figure 2. A print screen of the CBR resources of the RD System

The template “key risk management tasks”, for instance, locates and highlights the project participants’ arguments advanced for the solution of risk identification, risk analysis and risk response planning problems, while other arguments in a debate are temporarily hidden when a retrieved case is presented to a user. To achieve the template goal, this template structure should be described in terms of location acts: “Propose risk”, “Propose impact”, “Propose probability” and “Propose plan”, which are the key risk management locations available in the protocol. As this example shows, the template representation relies on a pre-defined mapping between the description of the template’s goal and the specific set of location acts that are able to present a solution for such an explanation goal (see Fig. 2 – D). However, if users desire more information about any argument in a retrieved case, independent of whatever personalization has been performed, they can scan through the entire content of the selected case, seeing the complete tree of the discussion that is recorded there. To help with the search for relevant risk management information in this tree, the RD System permits users to filter or expand the nodes or tree branches of these debates.

#### IV. A CASE STUDY OF COLLABORATIVE EXPERIENCE-BASED RISK MANAGEMENT

This illustrative case study involves a software project aiming to develop a web-based system for supporting a selected set of company managers to obtain the geographic location of certain employees of a company. The main problem of this project is that the software requirements are not described comprehensively since the project’s clients are not sure about the resources that this system would need to have for the privacy of the company employees not to be violated in particular situations. No experience with such a system was available there either among the project’s clients or the development

team involved in the construction of this system. It means that the reuse of past risk management experiences amounts to a form of identifying risks that these participants would not be aware in such a new kind of project for this team. This is one of the reasons that we choose to discuss this case study. The contextual properties of this project are: “c1: Team Size = Very Small”, “c2: Team Distribution = In House”, “c3: Criticality = Comfort Loss” and “c4: Rate of Change = High”.

In our work, we describe clear and intuitive resources for participants of a collaborative risk discussion (e.g. a project manager, a technical leader) to exploit past experiences in the enhancement of risk management tasks of a new project. In this case study, we show how our extended CBR approach can support the development of identification, analysis and planning tasks of risk management. We show how both queries based on facts and queries based on facts and arguments can be utilized by project stakeholders. With cases retrieved from the case base as a result of such queries, explanation templates are utilized to facilitate the comparison between the target problem and the most similar cases retrieved and, consequently, to improve the reuse of past experiences.

A first task in the risk management of the target project is the identification and analysis of threats that are likely to occur. By using our approach, a project manager can analyze what kind of risks can occur in past projects that contain factual characteristics (c1 – c4) that are similar to the ones described in the target project. To assess the most similar cases retrieved, the “key risk management tasks” explanation template can be utilized due to the fact that this template highlights essential risk management arguments in a debate. Two substantially relevant past cases, with similarity measures 90.1% (‘p1’) and 79.5% (‘p2’), are retrieved. Considering only p1

because of its high rating, the risks thus identified are: “p1.r1: The final product will not correspond to client expectations”, “p1.r2: The algorithms that were defined may not be adequate to the functional requirements of the system”, “p1.r3: There is a difficulty which is the construction of realistic scenarios and collection of testing data” and “p1.r4: The client of the project and the members of the development team have different views about the same requirements”. Of the four risks, the project manager determined that only r2 did not apply to the new project under consideration. The effectiveness of plans for the resolution of risks is a central topic in risk management. In this case, it is important to observe that management plans that were successful in past projects can be reused in new projects. To construct new plans, project participants can make a new kind of CBR query in the RD System. In it, both contextual project characteristics represented as factual properties and keywords or sentences occurring in the argumentative analysis of prioritized risks can be utilized. In this case study, the technical leader of the project stated such query by using both i) the characteristics c1-c4 and ii) the locution act “Propose risk” along with the keywords “s1: requirements” and “s2: client expectations”, as well as the locution act “Propose plan” with “s3: testing and experience”. The explanation template that was selected by this user was the “Risk and plan proposals” since it emphasizes risks and treatment plans, which is the information that the technical leader is looking for. The retrieval that followed brings up a most similar (86.3%) case p3. Its associated management plan proposals are: “p3.mp1: We need to record the changes of requirements more formally”, “p3.mp2: The requirements should be documented properly”; “p3.mp3: To use small examples in order to guide the test of the system”; “p3.mp4: We can have some training before the project starts” and “p3.mp5: We should get support from the technical leader of this project”. The technical leader accepted the immediate relevance of items 1, 2 and 4. In addition, mp3 suggested a consideration specially adapted to the conditions of the new project under analysis: “small examples could be developed and linked to the use cases of the project, as well as the test cases of the system.”

## V. CONCLUDING REMARKS

Collaboration and the reuse of experiences from past projects are crucial needs for users which aim to achieve effective risk management. Project stakeholders’ experience is naturally offered when these users advance different arguments in the collaborative debate of a project’s risks. Experience is also systematically captured when concrete risk management cases of problem-solving are shaped by not only factual information but also by such argumentation information. In this context, different extensions for CBR are proposed so that one could find and reuse these experiences in the solution of new risk management problems.

The contributions of this paper are the description of how to build a risk management memory, or case base, containing enhanced cases of collaborative risk management debates. Moreover, we discuss how debate participants can exploit different forms of querying so that they could have alternative ways of retrieving and reusing argument-based knowledge recorded inside similar cases for a given problem situation. We also propose case-based explanation template resources

for helping project stakeholders to inspect particular problems, proposals or solutions, which are expressed in the body of such past debate cases. In practice, this form of highlighting argumentation moves of collaborative risk management enables these users to achieve an understanding of reusable risk management information posed in a debate.

Future work involves giving the world access to (examples of) the RD System via the web, in addition to the development of evaluation experiments in collaborative risk management scenarios so that one could investigate forms of applying our approach in practice. We also plan to develop new explanation techniques for CBR and Argumentation in order to further improve the explanation capabilities of our CBR approach for experience-based collaborative risk management.

## REFERENCES

- [1] Y. Hu, X. Zhang, E. W. T. Ngai, R. Cai, and, M. Liu, “Software project risk analysis using bayesian networks with causality constraints,” *Elsevier Decision Support Systems*, vol. 56, pp. 439-449, 2013.
- [2] R. D. Choudhari, D. K. Banwet , and M. P. Gupta, “Assessment of risk in e-governance projects an pplication of product moment Correlation and cluster analysis techniques,” *An International Journal Electronic Government*, vol. 8, no. 1, pp. 85-102, 2011.
- [3] Crespo, Francisco L., Miguel AA Gómez, Javier Candau, and J. A. Manas, “Magerit-version 2, methodology for information systems risk analysis and management, book I – the method,” *Ministerio de Administraciones Publicas*, 2006.
- [4] R. L. d. Mántaras *et al.*, “Retrieval, reuse, revision and retention in CBR,” *Knowledge Eng. Review*, vol. 20, no. 3, pp. 215-240, 2005.
- [5] J. L. Kolodner, “*Case-Based Reasoning*,” San Mateo, CA: Morgan Kaufmann Publishers, Inc., 1993.
- [6] B. Moulin, H. Irandoust, M. Belanger, and G. Desbordes, “Explanation and argumentation capabilities: Towards the creation of more persuasive agents,” *Artificial Intelligence Review*, vol. 17, no. 3, pp. 169-222, 2002.
- [7] P. McBurney, and S. Parsons, “Dialogue games for agent argumentation,” *Argumentation in Artificial Intelligence*, I. Rahwan and G. R. Simari, eds., 2009.
- [8] T. J. M. Bench-Capon, and P. E. Dunne, “Argumentation in artificial intelligence,” *Artificial Intelligence*, vol. 171, pp. 619-641, 2007.
- [9] F. S. Severo, L. M. Fontoura, and L. A. L. Silva, “A dialogue game approach to collaborative risk management,” *Proc. of the 25th Int. Conf. on Software Engineering & Knowledge Engineering*, 2013.
- [10] F. Sormo, J. Cassens, and A. Aamodt, “Explanation in case-based reasoning – perspectives and goals,” *Artificial Intelligence Review*, vol. 24, no. 2, pp. 109-143, 2005.
- [11] *A Guide to the Project Management Body of Knowledge: PMBOK® Guide*: Project Management Institute, 2013.
- [12] P. H. Sneath, and R. R. Sokal, “Numerical taxonomy - The principles and practice of numerical classification,” San Francisco: W. H. Freeman and Company, 1973.
- [13] K. D. Ashley, and E. L. Rissland, “Law, learning and representation,” *Artificial Intelligence*, vol. 150, no. 1-2, pp. 17-58, 2003.
- [14] S. Heras, J. Jordán, V. Botti, and V. Julián, “Argue to agree: A case-based argumentation approach,” *International Journal of Approximate Reasoning*, vol. 54, no. 1, pp. 82-108, 2013.
- [15] W. C. Regli, X. Hu, M. Atwood, and W. Sun, “A survey of design Rationale systems: Approaches, representation, capture and retrieval,” *Engineering and Computers*, vol. 16, pp. 209-235, 2000.
- [16] P. Cunningham, D. Doyle, and J. Loughrey, “An evaluation of the usefulness of case-based explanation,” *Lecture Notes in Computer Science*. pp. 122-130.
- [17] B. Boehm, and R. Turner, “Using risk to balance agile and plan-driven methods,” *IEEE Computer Society*, vol. 36, pp. 57-66, 2003.
- [18] P. Kruchten, “Contextualizing agile software development,” *Journal of Software: Evolution and Process*, vol. 25, no. 4, pp. 351-361, 2013.
- [19] *PostgreSQL 9.3.3 Documentation*, “Full text search,” The PostgreSQL Global Development Group, 2014.

# Collaborative Merge in Distributed Software Development: Who Should Participate?

Catarina Costa<sup>1,3</sup>

<sup>1</sup>Technology and Exacts Science Center  
Federal University of Acre, UFAC  
Rio Branco - AC, Brazil  
catarina@ufac.br

José J. C. Figueirêdo<sup>2</sup>

<sup>2</sup>Information Technology Department  
Federal Institute of Acre, IFAC  
Rio Branco – AC, Brazil  
jair.figueiredo@ifac.edu.br

Leonardo Murta<sup>3</sup>

<sup>3</sup>Computing Institute  
Fluminense Federal University, UFF  
Niterói - RJ, Brazil  
leomurta@ic.uff.br

**Abstract**— Merge conflicts, which are rather common throughout the process of software development, are more frequent and complex to resolve when using the distributed software development approach, where developers are geographically dispersed. Normally, in the case of workspace merge, the last developer to merge code is responsible of conciliating the changes made in parallel and resolving conflicts. However, the last developer is not always the best team member to complete this task because he or she may not be familiar with the other parts of the code. With this in mind, the goal of this work is to analyze merge profiles of eight software projects and check if the development log is an appropriate source of information for identifying the key participants for collaborative merge. The obtained results are promising in this direction.

**Keywords**- Collaborative merge; distributed software development; version control

## I. INTRODUCTION

According to Bendix et al. [1], Configuration Management can assist either co-located or distributed software development. Besides the existence of similarities between both approaches, there are some important differences. One of them is how each approach handles conflicts. When a conflict arises in a co-located project, developers can gather together and work over it. Distributed projects, which aside from the geographic dispersion of the team may involve time and sociocultural differences, require a deeper effort to bypass this kind of impasse.

Merge conflicts are common, especially in massive and distributed software projects [2]. Identifying and resolving these conflicts are not trivial tasks. While two simple concurrently changes might be individually correct, their combination may raise conflicts. These conflicts demand conciliating choices and, consequently, this whole scenario usually introduces rework [3].

Most contemporary version control systems adopt optimistic version control and are based on the premise that the workspace merge is done by the last developer to commit. For this reason, the developer's knowledge regarding the changes performed in parallel is not taken into consideration.

Consequently, the last developer might not be the most suitable developer to fulfill the conciliation of decisions during merge.

Some state-of-the-practice tools, such as KDiff<sup>1</sup>, WinMerge<sup>2</sup>, Diffuse<sup>3</sup>, and SourceGear DiffMerge<sup>4</sup>, feature automated solutions for merge conflicts, but they not able to resolve conflicts that occur in the same region of the files. In contrast, state-of-the-practice tools, such as CASI [4], CloudStudio [5], CoDesign [6], Palantír [3], and Syde [7], try to avoid conflicts through awareness. Their main goal is to alert developers in a proactive way of situations that may lead to conflicts [5], [8]. However, these approaches require certain conditions, such as developers working in sync in terms of time and over the same branch, which is not always the case for distributed software development. All in all, even adopting the aforementioned tools, conflicts may still exist. Finding the appropriate developers to solve them is an important challenge.

The use of the existing shared knowledge of software projects is a missing element for conflict resolution in distributed software development [2]. The configuration management repositories have a trail of who changed the software and can help on identifying the most appropriate developers for solving conflicts. This information could be extracted, for instance, from the project development progress history until the point where the merge takes place.

This work aims at providing a deeper analysis of how parallel development occurs and who could participate in collaborative merge sessions. To do so, we analyzed the merge profiles of eight software projects versioned using the Git<sup>5</sup>. The analysis was guided by the following five research questions, whose objective is the identification of the developers that worked in parallel before a merge and that should take part in the merge process.

Q1: What is the average number of developers per branch?

---

<sup>1</sup> <http://kdiff3.sourceforge.net/>

<sup>2</sup> <http://winmerge.org/>

<sup>3</sup> <http://diffuse.sourceforge.net/>

<sup>4</sup> <http://www.sourcegear.com/diffmerge/>

<sup>5</sup> <http://git-scm.com>

Q2: What is the average number of commits per developer in each project?

Q3: What is the maximum number of developers in each branch?

Q4: How many merges have the same developers in both branches?

Q5: Who should participate in collaborative merge sessions?

These answers can provide important insights on how much the project development log can assist in the designation of key participants to resolve merge conflicts. This information can help on defining new methods for assign developers to participate in merge sessions, known as collaborative merge.

This work is organized in six sections, including this introductory section. Section II presents the concepts involved in the research. Section III features the related works. Section IV presents the research method. Section V contains the analysis results and some insights on how the development log could help on the identification of merge participants. Finally, the conclusions of this study are presented in Section VI.

## II. BACKGROUND

According to Mens [9], three types of conflicts may arise during merge: textual, syntactic, and semantic. Textual conflicts, also known as physical conflicts, occur when concurrent operations (e.g., addition, removal, or edition) take place over the same physical parts of a file (e.g., same line). Syntactic conflicts occur when concurrent operations break the syntactic structure of the file when combined. The syntactic structure of a file is usually defined in terms of a schema or grammar. Finally, semantic conflicts occur when concurrent operations break the semantics of the file when combined. The semantics of the file can be expressed both in terms of the programming language semantics or expected program behavior (usually verified by test cases).

The existing version control systems are limited to textual conflicts. This way, a conflict arises when two or more developers make inconsistent changes in the same part of the code. When this happens, the changes cannot be incorporated into the repository until the conflict is resolved by some developer [10]. Fig. 1 shows an example of merge, where four developers work in two branches, being d1, d2, and d3 in the first branch, and d2, d1, and d4 in the second branch. In a given moment, d4 decides to merge the second branch into the first branch. In this case, in most version control systems, d4 must conciliate the choices and possibly edit some parts of the code written by the other developers.

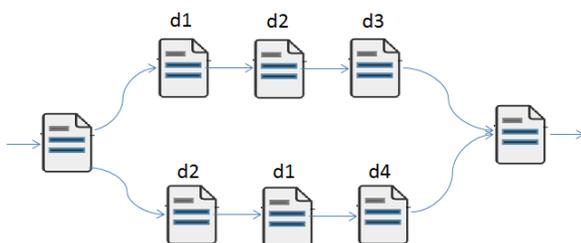


Figure 1. A software merge scenario.

This example evidences that, disregarding the content contribution of each developer, d1 and d2 probably know the part they are working with better than d3 and d4 for that merge. This occurs because they worked in both branches and, consequently, understand better the changes performed in both branches. The developer d4 could be at least assisted by d1 and/or d2 to more effectively conciliate the decisions.

## III. RELATED WORK

The majority of the works in the literature on merge conflicts focus on the prevention of conflicts through awareness. By monitoring workspaces, awareness tools, such as Palantir [3] and CloudStudio [5], notify the developers every time an change that may lead to a conflict is being made. Other approaches, such as Crystal [10] and WeCode [11], work by continuously trying to merge the local copies of the various users. Although the works might have distinct solutions, conflict prevention is the goal of these approaches.

All approaches presented here require developers to be constantly synced in time throughout the development of the software, but this is not always possible in distributed software development. As mentioned before, the related work also expects that developers are working over the same branch. However, for different reasons [12] sub-teams work in parallel over different branches. Thus, conflicts would not be preemptively detected, which would imply solving them during merge anyway.

Nieminen's work [2] proposes a web-based tool (CoRED) that allows multiple users to resolve merge conflicts in a collaborative way. When facing a conflict, developers can invite a workmate to take part in a conflict resolution session, aiming at reaching a consensus. In this work, the authors focus on conflict resolution with the assistance of a web tool, and the developer select the resolution session participant of their choice. Although relevant, this work can be seen as complementary to our work, as they do not mention how participants could be selected.

In other work, based in collaborative merge of models [13], the authors present a model merging approach that shows conflicts as open questions. To do so, the approach allows to represent conflicts explicitly as parts of the model and thereby to postpone them. This facilitates discussion on conflicts and collaborative conflict resolution. The study shows that developers like to discuss conflicts in a considerable percentage of conflicts. The authors defend that the resolution of a conflict must not be assigned to only one developer, becoming a collaborative task.

The last two works discussed in this section contributed with interesting insights for this paper. Our work differs from them because we aim at analyzing merge profiles and check if the development log is an appropriate source of information for identifying the key participants for collaborative merge. Besides, these works do not foresee the physical distance of the people involved in the project, which makes it harder to identify the developer that has to be contacted when a conflict arises.

#### IV. MATERIALS AND METHODS

Our research followed the process depicted in Fig. 2 for comprehending how merge happens in practice. This process comprises project selection, merge profile analysis, and consolidation of results.

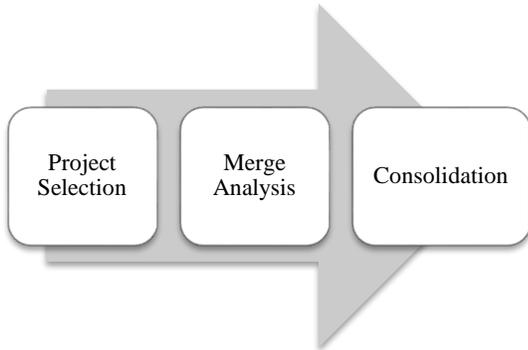


Figure 2. Research Steps.

In the **project selection step**, eight projects were chosen: Django, Git, JQuery, MaNGOS, Perl5, Rails, Voldemort, and Zotero. These projects were selected because they have different sizes, are popular, are coded in different programming languages, have different goals, and are very active. This contributes for a more embracing analysis. After select these projects, they were cloned from Github<sup>6</sup> making it possible to perform the next step of our research process.

In the **Merge Analysis Step**, the log of each selected project was analyzed to extract information about developers, commits, and merges. To do so, we implemented an automated infrastructure that parses the log provided by Git and extracts all merge commits. Moreover, it is able to identify the parent commits that were merged and navigate until the common ancestor just before forking the history.

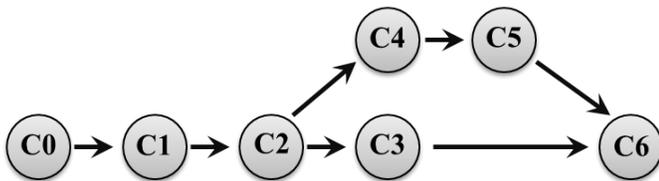


Figure 3. A simple merge example

Fig. 3 shows an example of merge commit (C6), two parent commits that were merged (C3 and C5), and the common ancestor (C2). From these commits, it is possible to identify the commits within each branch. These commits are comprised between the common ancestor and each one of the parent commits that were merged (including the parent commits). The first branch has two commits (C4 and C5) and the second branch has only one commit (C3).

After identifying each commit that was merged, we extract the author information (name and e-mail). Next, we extract multiple information, such as: who committed in each branch,

how many commits they did, who committed in both branches, and others.

The most complex merges were prioritized by inverse sorting the merges by the harmonic mean of the number of developers in each branch. An in-depth analysis was carried out for some of the most complex merge scenarios of these projects in order to investigate and indicate the developer to participate in the collaborative merge.

Finally, the **consolidation step** comprises the answers of the questions presented in Section I. The results were organized in a way that shows the merge profiles and information on the most suitable developers to take part in conflict resolution, answering the research questions.

#### V. RESULTS AND DISCUSSION

This section presents the answers to the research questions posed on Section I, according to the research process described in Section IV. Besides getting merge behavior patterns, information such as the average number of commits by developer and the number of developers working in branches, amongst other, are shown. Table I describes the analyzed projects with their total number of merges and developers. This data was extracted from projects between the end of 2013 and beginning of 2014.

TABLE I. PROJECT'S GENERAL DESCRIPTION

Project	General Information		
	Purpose	# Merges	# Developers
Django	Python web framework	404	396
Git	Distributed revision control system	7434	1103
JQuery	JavaScript Library	250	133
MaNGOS	A full-featured MMO server suite	255	266
Perl5	Programming language	1607	789
Rails	Ruby web framework	6147	2528
Voldemort	Distributed database	406	59
Zotero	Citation manager	223	16

By analyzing the information shown on Table I, it is noticeable that the projects range from just few merges (Zotero, JQuery, and MaNGOS) to a large number of merges (Git and Rails). It is also observable that the projects have varied numbers of participants, ranging from 16 in Zotero to 2528 in the Rails project. This diversity is important to allow us exploring different situations, enabling a wider observation over the development projects.

In the following, we answer each research question presented in Section I.

A. Q1: What is the average number of developers per branch?

The first question observes the quantity of developers involved in each branch of the merges, for each project.

<sup>6</sup> <https://github.com/>

The branches of all the merges, which comprise commits between each parent of the merge and the common ancestor, were processed and the results are shown in Table II. It is important to notice that there is a reasoning to number the branches: “the first parent is the branch you were on when you merged, and the second is the commit on the branch that you merged in” [14].

TABLE II. AVERAGE NUMBER OF DEVELOPERS PER BRANCH

Project	Developers per Branch			
	Average Branch 1	Standard Deviation	Average Branch 2	Standard Deviation
Django	7.05	20.60	2.64	8.88
Git	24.00	53.63	1.99	14.01
JQuery	4.56	7.26	1.57	2.07
MaNGOS	1.58	1.86	5.00	4.88
Perl5	2.02	7.01	3.46	6.51
Rails	10.95	43.77	2.41	8.58
Voldemort	1.47	1.67	1.73	1.45
Zotero	1.38	1.16	1.56	0.94

It is observable that, for example, Git has an average of 24 developers per merge on branch 1 and only 1.99 developers on branch 2. The same occurs in other three projects, having high values on branch 1: Django, JQuery, and Rails. In contrast, MaNGOS, Perl5, Voldemort, and Zotero have a higher average of developers on branch 2. However, with a clear exception of MaNGOS, the difference is not significant.

**B. Q2: What is the average number of commits per developer in each project?**

The second question observes the average number of commits per developer. To get the numbers in Table III, the totality of commits was divided by the totality of project participants.

TABLE III. AVERAGE COMMITS PER DEVELOPER

Projects Name	Commits per Developer	
	Average	Standard Deviation
Django	10.88	17.49
Git	176.26	418.56
JQuery	12.52	24.08
MaNGOS	7.31	21.37
Perl5	12.17	50.25
Rails	33.51	100.31
Voldemort	23.08	43.12
Zotero	42.06	74.08

The average number of commits per developer evidences the average participation of the developers in the project. Furthermore, it allows for the identification of the developers

that participate the most and, therefore, being the most suitable to resolve merge conflicts.

**C. Q3: What is the maximum number of developers in each branch?**

The third research question observes the maximum number of developers in each of the merging branches. Table IV shows the merges with the highest harmonic mean of branches 1 and 2. The harmonic mean is a plausible metric because it allows for finding scenarios with high values in both branches.

TABLE IV. MAXIMUM NUMBER OF DEVELOPERS PER BRANCH

Projects Name	Maximum Number of Developers			
	Merge Commit Hash	Branch 1	Branch 2	Harmonic Mean
Django	9cc6cfc4057e...	8	12	9.60
Git	a8816e7bab03...	158	10	18.80
JQuery	63aaff590ccc...	20	3	5.22
MaNGOS	404785091845...	11	17	13.35
Perl5	fed34a19f844...	10	6	7.50
Rails	bf2b9d2de3f8...	47	164	49.37
Voldemort	2a5c69145fb6...	4	8	5.33
Zotero	d456117ebe9c	3	5	3.75

Consider, for example, the merge with most developers in the Rails project, with harmonic mean of 49.37, having 47 developers that performed commits on branch 1 and 164 developers that performed commits on branch 2. For this case, the complexity of identifying the key participant(s) to solve a certain merge conflict can be rather high. The possibilities of communication exceed 7,000 channels between the involved developers. On the other hand, the Zotero project has harmonic mean of 3.75, leading to 15 different channels of communication amongst developers.

The amount of developers that performed commits in a given branch can help identify which participant is more suitable to participate in the collaborative merge. That is, who is a potential knower of the changes made in such merge.

**D. Q4: How many merges have the same developers in both branches?**

The fourth question observes the number of merges having the same developers in both branches. Table V displays information on the intersections of developers in merging branches. The first column presents the project names. The remaining columns show the quantity and proportion of:

- a) Merges that have all the developers in common in the branches;
- b) Merges that have some developers in common; and
- c) Merges that have different developers in both branches.

As seen in the Django project, only 1 merge (0.25%) has exactly the same developers in both branches and 8.66% have some intersection. In the Git project, for instance, the

intersection reaches more than 50% of the merges, and the same developers participate in both branches in about 5% of the merges. In the JQuery, MaNGOS, Voldemort, and Zotero projects, the number of people in both branches is also expressive, with more than 30% of the merges having some intersection. The Django, Perl, and Rails projects feature small intersections of developers in the branches.

TABLE V. INTERSECTION OF DEVELOPERS IN MERGING BRANCHES

Projects Name	Intersection of Developers					
	Exactly the same (a)		Some intersection (b)		Without intersections (c)	
Django	1	0.25%	35	8.66%	368	91.09%
Git	435	5.85%	3926	52.82%	3073	41.33%
JQuery	19	7.60%	72	28.80%	159	63.60%
MaNGOS	7	2.74%	71	27.85%	177	69.41%
Perl5	15	0.93%	95	5.91%	1497	93.16%
Rails	56	0.91%	886	14.41%	5205	84.68%
Voldemort	64	15.76%	83	20.44%	259	63.80%
Zotero	34	15.25%	94	42.15%	95	42.60%

Given the merge example presented on Fig. 1 on Section II, the information that there are developers that know each of the merging branches can aid in the designation of key participants to take part in the collaborative merge. It is believed that, as the developer participating on the merge knows parts of the work that is being completed in both branches, the chances of performing a more coherent choice during conciliation increases.

E. Q5: Who should participate in collaborative merge sessions?

In the fifth and last question we analyze some complex merge scenarios enlisted in Table IV and propose the most suitable participants to perform collaborative merge, should a conflict arise.

1) *Git*: the merge with the most complex setting of developers, by harmonic mean, has 158 developers in branch 1, and 10 developers in branch 2. Besides, 5 developers committed in both branches. Table VI shows the amount of commits made by each one of these 5 developers.

TABLE VI. DEVELOPERS IN BOTH BRANCHES FOR GIT

Merge	Developers in Both Branches – Git Project		
	Developers	Commits in Branch 1	Commits in Branch 2
a8816e7bab03 ...	Markus	23	5
	Alex	7	1
	Shawn	2	4
	Jens	1	4
	Ferry	1	1

For this merge, the average number of commits per developer is 8.9 in branch 1 and 2.3 in branch 2. In this case, although *Markus* was not the last developer to commit, should a conflict arise, he would be a strong candidate to participate in the collaborative merge.

In a collaborative merge based on logs, aside from the verification of the developer in both branches (Table VI), it is possible to verify the participants that performed most commits (Table VII). Table VII shows the six developers that committed the most, three in the first branch and three in the second branch. Besides possibly knowing a lot about the work being completed, if there is an intersection between the two analyses, the weight of the indication of the participant must be higher. For this merge, there are some intersections between developers in both analyses. Therefore, *Junio*, *Markus*, *Shawn*, and *Jens* would be candidates to be part of the collaborative merge.

TABLE VII. DEVELOPERS WITH MOST COMMITS

Merge	Developers with the Most Commits – Git Project		
	Developers	Commits in Branch 1	Commits in Branch 2
a8816e7bab03 ...	Junio	512	0
	Schindelin	69	0
	Jeff	58	0
	Markus	23	5
	Shawn	2	4
	Jens	1	4

2) *Perl5*: the merge with the most complex scenario of developers, by harmonic mean, has 10 developers in branch 1 and 6 developers in branch 2. Only 1 developer participated in both branches, as seen on Table VIII.

TABLE VIII. DEVELOPERS IN BOTH BRANCHES FOR PERL5

Merge	Developers in Both Branches – Perl5 Project		
	Developers	Commits in Branch 1	Commits in Branch 2
fed34a19f844...	Hans	2	1

For this merge, only *Hans* has committed in both branches and so he is a strong candidate to participate in the collaborative merge. In addition, he was also one of the six developers that committed the most (Table IX).

TABLE IX. DEVELOPERS WITH MOST COMMITS

Merge	Developers with the Most Commits – Perl5 Project		
	Developers	Commits in Branch 1	Commits in Branch 2
fed34a19f844...	Gurusamy	17	0
	Hugo	2	0
	Tom	2	0
	Jarkko	0	16
	Hans	2	1
	Kurt	0	1

Table IX shows the six developers that committed the most, three in the first branch and three in the second branch. Two other strong candidates to participate in the merge are *Gurusamy* and *Jarkko*, who were responsible for a large number of commits on branches 1 and 2, respectively, as shown on Table IX.

3) *Voldemort*: the merge with the most complex setting of developers, by harmonic mean, has 4 developers in branch 1 and 8 developers in branch 2. In this case, 4 developers participated in both branches, as shown in Table X.

TABLE X. DEVELOPERS IN BOTH BRANCHES FOR VOLDEMORT

Merge	Developers with the Most Commits – Voldemort Project		
	Developers	Commits in Branch 1	Commits in Branch 2
2a5c69145fb6 ...	Bhushesh	33	11
	Kirktrue	29	10
	Alex	20	13
	Bbansal	4	2

The developers who committed the most on branch 1 also committed on branch 2, which makes all of them (especially *Bhushesh*, *Kirktrue*, and *Alex*) strong candidates to perform the collaborative merge. In this case, the developers that participate in both branches were also the ones that performed the most commits, reinforcing the evidence that they should participate in the collaborative merge.

## VI. CONCLUSION

As shown in the previous sections, it is possible to use the history log to identify the potentially best developers to take part in the collaborative merge session. This can be done both through the identification of developers who committed in both branches or the ones that committed the most in one of the branches.

As a contribution, this study presented answers to some research questions regarding repository logs. Besides, some merge analyses were displayed, showing that it is possible to identify potential developers to perform collaborative merges.

As a threat to validity, the analyses were based on the committers' names and/or email addresses, which can generate inconsistency in the numbers presented. Furthermore, the analyses were carried out in merges that had already occurred. For this reason, it was impossible to effectively measure the effects of including the suggested developers in the merge session. Moreover, this study does not take under consideration developers' previous knowledge of libraries or programming languages being used, but solely the commit logs, that is, the participation of the developers in the specific project under analysis.

As a future work, we hope to apply this research in pending merges, in order to observe the advantages of indicating who is the most suitable developer. Moreover, we intend to conceive a method for indicating the most suitable developers and

automated such method. Finally, we want to add additional analysis related to the expertise over changed artifacts and the knowledge proximity amongst developers.

## ACKNOWLEDGMENT

The authors would like to thank CAPES, CNPq, and FAPERJ for the financial support.

## REFERENCES

- [1] L. Bendix, J. Magnusson, and C. Pendleton, "Configuration Management Stories from the Distributed Software Development Trenches," presented at the 2012 IEEE Seventh International Conference on Global Software Engineering (ICGSE), 2012, pp. 51–55.
- [2] A. Nieminen, "Real-time collaborative resolving of merge conflicts," in 2012 8th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2012, pp. 540–543.
- [3] A. Sarma, D. Redmiles, and A. van der Hoek, "Palant&#237;r: Early Detection of Development Conflicts Arising from Parallel Code Changes," *IEEE Trans. Softw. Eng.*, vol. 38, no. 4, pp. 889–908, Jul. 2012.
- [4] J. Portillo-Rodriguez, A. Vizcaino, C. Ebert, and M. Piattini, "Tools to Support Global Software Development Processes: A Survey," in 2010 5th IEEE International Conference on Global Software Engineering (ICGSE), 2010, pp. 13–22.
- [5] H. C. Estler, M. Nordio, C. A. Furia, and B. Meyer, "Unifying Configuration Management with Merge Conflict Detection and Awareness Systems," in Proceedings of the 2013 22nd Australian Conference on Software Engineering, Washington, DC, USA, 2013, pp. 201–210.
- [6] J. young Bang, D. Popescu, G. Edwards, N. Medvidovic, N. Kulkarni, G. M. Rama, and S. Padmanabhuni, "CoDesign: a highly extensible collaborative software modeling framework," presented at the 2010 ACM/IEEE 32nd International Conference on Software Engineering, 2010, vol. 2, pp. 243–246.
- [7] L. Hattori and M. Lanza, "Syde: a tool for collaborative software development," presented at the 2010 ACM/IEEE 32nd International Conference on Software Engineering, 2010, vol. 2, pp. 235–238.
- [8] C. Costa and L. Murta, "Version Control in Distributed Software Development: A Systematic Mapping Study," in 2013 IEEE 8th International Conference on Global Software Engineering (ICGSE), 2013, pp. 90–99.
- [9] T. Mens, "A state-of-the-art survey on software merging," *Software Engineering, IEEE Transactions on*, vol. 28, no. 5, pp. 449–462, 2002.
- [10] Y. Brun, R. Holmes, M. D. Ernst, and D. Notkin, "Proactive detection of collaboration conflicts," in Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering, 2011, pp. 168–178.
- [11] M. L. Guimarães and A. R. Silva, "Improving early detection of software merge conflicts," in Proceedings - International Conference on Software Engineering, 2012, pp. 342–352.
- [12] S. P. Berczuk and B. Appleton, *Software configuration management patterns: effective teamwork and practical integration*. Boston, Mass: Addison-Wesley, 2003.
- [13] M. Koegel, H. Naughton, J. Helming, and M. Herrmannsdoerfer, "Collaborative model merging," in Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion, New York, NY, USA, 2010, pp. 27–34.
- [14] S. Chacon, *Pro Git*. Berkeley, CA; New York: Apress ; Distributed to the Book trade worldwide by Springer-Verlag, 2009.

# *An Argument-based Collaborative Negotiation Approach to Support Software Design Collaboration*

Nan Jing

SHU-UTS SILC Business School  
Shanghai University  
Shanghai, P.R. China

Stephen C-Y. Lu

Computer Science  
University of Southern California  
Los Angeles, California, USA

Hung-Fu Chang

Computer Science  
University of Southern California  
Los Angeles, California, USA

**Abstract - Most software design involves multiple stakeholders and group decision making activities. To reach an agreement in such activities, negotiation is the key task. Our work synthesizes Toulmin's argument structure and a value-focused objective hierarchy to develop a new negotiation approach for the software design. This approach structures negotiation arguments in order to establish a common ground for a more effective negotiation. In this paper, we introduce this approach, discuss its foundations and main process, and present a case study in which we applied this approach for a real-life software system design project.**

**Keywords: Software Design, Group Decision, Collaborative Negotiation, Argument Structure**

## I. INTRODUCTION

Software design always involves multiple stakeholders and group decision making activities. While designing the software, stakeholders from various departments bring their information and expertise to determine major function or to lay out the architecture. However, making this group decision is never easy because their information is overwhelming, decision objectives and criteria are manifold, alternatives may not be clear, and preferences are subjective and often conflicting.

Collaboration, by its definition, means that every member in a team, through social interactions, can seek for useful information together, share objectives and criteria, and make mutually-agreed decisions based on various alternatives and preferences. Collaborative negotiation is one type of collaboration in which stakeholders communicate their opinions, negotiate the alternatives, influence each other's preferences, and try to resolve conflicts by finding an agreement. However, most past research focused on either individual decision making (e.g. [1, 2]) or a generic negotiation process (e.g. [3, 4]); and in authors' knowledge very few of them combine both aspects, i.e. identify and organize stakeholders' objectives and preferences, and then provide a systematic negotiation process to reconcile the conflicts in these organized objectives and preferences.

In this research, we propose a new research approach to identify stakeholders' objectives and preferences using the framework of Keeney's value-focus thinking [8] and describe a systematic negotiation process, which generates negotiation arguments based on identified objectives and preferences and then evaluates the arguments by calculating the objectives and preferences to find the most preferred one by the negotiation team. In this paper, we will introduce this approach in section

II. In section III we present a case study that is conducted to validate the approach. Section IV concludes this paper and outlines the future work.

## II. ARGUMENT-BASED COLLABORATIVE NEGOTIATION APPROACH

There are two foundations in our approach. One is the argument structure that is based on Toulmin's generic argument structure [9]; the other is an objective hierarchy that is built based on Keeney's value-focus thinking framework [8]. Argument is developed to persuade or convince others that one's reasoning is more valid or appropriate in negotiation. Objective hierarchy is used to organize stakeholders' objectives and preferences, which will be used to generate the arguments. The key step of our negotiation approach is to generate and evaluate arguments by synthesizing the two research foundations. In this section, we will first present the details of these two foundations, and then discuss our approach which is based on the synthesis of these two foundations.

### A. Objective hierarchy

Because "values, as the driving force to our decision making, are fundamental to all that we do", every stakeholder has his own value when he makes any decision. For example, when selecting a platform for developing web service in software system design, an individual contractor who favors open source may always choose Java based frameworks since most of those are open-source to the public and everything is free to use, however, a manager who has more concern with platform and development support may choose .Net as the main platform since using that it is more convenient for him to directly contact Microsoft for customer service instead of asking a question in the public community and desperately waiting for someone to kindly give an answer. Keeney emphasizes that each decision maker pursues his own values in a group decision-making process and the goal for which they participate in this process is to maximize his value.

In our framework, values are classified into fundamental objectives and means objectives. Fundamental objectives concern the ends in a specific decision context and the means objectives are the ways to achieve those ends. In this objective hierarchy, we also define attributes for evaluating how well these objectives are achieved. These attributes describe the objectives and have common interpretation to every stakeholder. They can be used as the common scales to evaluate the negotiation arguments for the degree to which they

achieve the objective (that the attributes belong to) and thus yield independent measurement values for each argument. Then the arguments can be ranked for how well they achieve the objectives. The evaluation results can be either a quantitative value within a pre-defined range, or a qualitative value, e.g. support, neutrality (indifferent or uninterested), opposition, controversy (not support but match the interest for the group).

### B. Argument structure

Practicing collaborative design and negotiation dialogue have been found to be positively linked with argument development and critical thinking skills. The work of Buckingham and his colleagues argue that standardizing an argument's structure facilitates its subsequent communication since important meta-information and relationships can be more easily perceived and analyzed by others [1]. Stephen E. Toulmin's 1958 work [10] has become commonplace in structuring arguments - Toulmin acknowledges as much in the preface to his 1984 text [11]. For example, Houp, Pearsall and Teheaux's textbook, Reporting Technical Information, introduces Toulmin logic as providing "a way of checking your own arguments for those overlooked flaws. It can also help you arrange your argument" [6]. The goal of developing arguments in negotiation is to persuade or convince others that one's reasoning is more valid or appropriate. Toulmin's structure of argument provides the language symbols and data structures that support the argumentation process. Toulmin's structure is procedural and the layout of this structure focuses on the movement of accepted data to the claim through a warrant. Toulmin also recognizes three secondary elements that may be present in an argument: backing, qualifier, and rebuttal. Backing is the authority for a warrant, provides credibility for the warrant, and may be introduced when the audience is unwilling to accept the warrant. A qualifier indicates the degree of force or certainty that a claim possesses. Finally, rebuttal represents certain condition or exception under which the claim will fail and hence anticipates objections that might be advanced against the argument to refute the claim [10]. As such, Toulmin's argument structure becomes a popular mechanism for structuring arguments between negotiating stakeholders. It aims to clarify the reasoning process by encouraging parties to make explicit important assumptions, distinctions, and relationships as they construct and rationalize ideas [1].

### C. Synthesis between the Objective Hierarchy and the Generic Argument Structure

Using the Toulmin's argument structure, which is generally more objective than implicit arguments, it is hard for stakeholders to hide bias because the grounds and backing of an argument are clearly listed and described to support the claims. Therefore, all stakeholders' perspectives are relatively easy to be fully observed by others through examination of the ground and warrants that the stakeholder expresses [7]. However, there are remaining unresolved issues in most of the above work, such as a clear guide to systematically generate the arguments and evaluating them for the best in an operational negotiation process. Our research try to resolve this challenge by synthesizing the argument structure with the value-focused objective hierarchy, developing practical

methods of generating and evaluating structured arguments (see Figure 1).

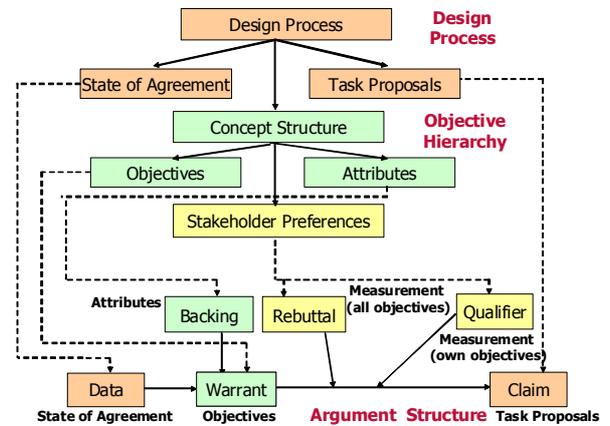


Figure 1 Synthesis between Objective Hierarchy and Argument Structure

In Fig. 1, the claim is proposed by the stakeholder and consists of a sequence of actions/objects to implement the task. The data specifies the state of team agreement about the design process (e.g., tasks, objectives) together with any applicable information to support the claim. The state of team agreement can be accomplished by previous design tasks and arguments, such as the results of the previous tasks, the common understanding of the team for these tasks, and the objectives jointly proposed by the team. The applicable information includes stakeholder's understanding and expectation for the current task and any facts that can be used to justify the proposal. This data describes the initial state of the current task, presents the applicable information, and therefore provides background support for the claim. Stakeholders' objectives are in the place of warrant, identifying the value that stakeholders want to achieve along with clarification of the relationship between the value and the current state of agreement. These objectives, as warrants, justify that the proposal can achieve the value based on the state of agreement (i.e. data). The attributes of each objective, corresponding to the 'backing' component, further explain the objectives by describing their measurement criteria and then validate the relationship amongst the objectives, the proposal and the current state of agreement. Based on these objectives and attributes, the measurement result regarding the achievement of the aforementioned objectives by stakeholder's own proposal work as a 'qualifier' to indicate the degree of desire of the stakeholders for the proposal. Similar results regarding the achievement of the objectives by others' proposals work as 'rebuttal' and describe possible conditions that could fail the claim or suspend the warrant.

Based on the concrete meanings (of proposals, objectives, attributes, measurement results) given to the argument components from the objective hierarchy, when the stakeholders cannot agree upon on one task proposal, argument evaluation is taken based on the level of the objectives achievement to get the best choice.

*D. Argument Generation and Evaluation*

The main strength of our approach is to use the structured arguments that are synthesized with the objective hierarchy to guide the stakeholders to generate and evaluate their arguments in collaborative negotiation. The stakeholders jointly propose an objective hierarchy and declare their perspectives (e.g., preferences) based on the objectives. Then based on the identified objectives and declared perspectives, the stakeholders are guided to systematically generate and exchange their negotiation arguments. If no argument is commonly accepted at the end of collaborative negotiation, all the arguments will be evaluated by aggregating stakeholders' preferences and ranked to recommend an optimal choice or a well-informed team leader will make the choice based on the evaluation results.

As such, there are four steps in this process. The first two steps guide the stakeholders to building the objective hierarchy and declare stakeholders' perspectives. The next two steps discuss how the arguments are utilized based on the objectives and perspectives collected in the first two steps.

*1) Propose an "objective hierarchy" for the identified conflicting design task*

Negotiating conflicting implementation proposals of a design task in its baseline design process indicates some differences in the stakeholders' objectives and perspectives (i.e., preferences for arguments and objectives). These objectives include the "fundamental objectives" (as the values) for which the task is undertaken, and the "means objectives" that helps achieve the fundamental ones. The objective's attributes for measuring the proposal about the degree to which the objective are achieved if the proposal is accepted. These attributes describe the objectives and should have a common interpretation to every stakeholder. If an objective does not have any means objectives or any attributes that are naturally used to interpret the objective (e.g., "network bandwidth of integrated system" is a so-called natural attribute of the objective "increase the throughput after the systems integration"), an attribute "support vs. opposition" will be added, based on which stakeholder in later steps can declare their perspective as either support or opposition. According to the goals defined at the beginning of the design process, the stakeholders should be able to identify objectives and attributes in this step.

Since our approach uses an objective hierarchy to organize the objectives and capture their differences, the hierarchy is jointly built by the stakeholders based on their understanding and expectations ("values") of the design tasks. And the objectives in this hierarchy will be dynamically changed by the social interactions among the stakeholders. In reference to the information in an objective hierarchy, the stakeholders can declare their preferences regarding how important the objectives (i.e. the weights of the objectives) are and how much each proposal is supported or opposed. The latter will be obtained in the next step from the values assigned by the stakeholders for the objectives' attributes. The weights of the objective are collected in this step after these objectives are declared in the structure. The relative importance of each objective is defined in one-to-ten scale as follows:

- 10: Very important;
- 8: Somewhat more important;
- 6: Important;
- 4: Somewhat less important;
- 2: Very less important;
- 1: Lowest important.

*2) Each stakeholder declares 'perspective' based on the above objective hierarchy*

Once an objective hierarchy is established by the team, each attribute of each objective should be assigned a value before the arguments are evaluated. In the case of the added attribute 'support vs. opposition', each stakeholder can express their own preferences based on their expertise and understanding and this opinion should describe their position of either supporting or opposing the argument according to how much the objective (to which the attribute belongs) can be accomplished if the argument is accepted.

Since there is no practical way that a complete analytical modeling of negotiation can be fully developed and incorporated for a group of decision makers, our approach takes a rather simplified view by focusing on modeling the dynamic impacts of negotiation, i.e. on the evolving "perspectives" of the stakeholders, as they express their opinions toward the objective hierarchy. These dynamically evolving perspectives are declared for the proposed objectives of which the stakeholders that have common interests or some expertise. In other words, the perspective dynamically depicts a stakeholder's perceptions of his/her or others' objectives. These perceptions could include the stakeholders' intention for their ideas to succeed and their support for or disagreement with how well their own or others' argument can achieve the objectives, either proposed by themselves or others. Therefore, the perspectives indicate the difference in the stakeholders' perceptions that cause the conflict in the technical proposals of the tasks and put the negotiation into necessity. Moreover, these perspectives will be further analyzed in order to systematically evaluate the arguments in our negotiation approach.

TABLE I. SCALE OF SUPPORT VS. OPPOSITION

10	<b>Strong Support:</b> the proposal will most likely help achieve the objective
8	<b>Support:</b> the proposal will likely help achieve the objective
6	<b>(1) Neutrality</b> (fair, unknown or uninterested): the proposal may not either contribute to or harm the achievement of the objective. <b>(2) Controversy:</b> the proposal may have some effect in achieving the objective, but the decision maker is not clear the effect.
4	<b>Opposition:</b> the proposal will likely bring negative effects in achieving the objective.
2	<b>Strong Opposition:</b> the proposal will most likely bring detrimental effects in achieving the objective.
1	<b>Strongest Opposition:</b> the proposal will definitely

bring detrimental effects in achieving the objective.

Although stakeholder's perspectives are often highly subjective in nature, a quantitative method is needed to define the measurement scales of the perspectives and further analyze these perspectives for argument evaluation. We define a one-to-ten scale (see Table I) to measure the stakeholder's perspectives of either supporting or opposing the arguments. When the task proposals were being evaluated, for the 'support vs. opposition' attribute, stakeholders declared their perspectives about the proposal's value based on their expertise and understanding.

### 3) Argument generation

While generating the negotiation arguments, claims and data are collected from the baseline design process representing technical decisions. Warrant, backing, qualifiers and rebuttals are obtained from the objective hierarchy and stakeholders' perspective models.

By using our approach, stakeholders will have a better understanding of each other because they share not only their claims, but also their underlining reasons and desires (e.g., perspectives). For example, Figure 2 shows the details of an argument of an engineer, whose claim for the task "define quality requirements" is to define the attributes of performance and security for the integrated system. The data describes the initial state (of this task) which includes design requirements, application constraints and architecture style. To justify the use of the data, the warrant has his fundamental and means objectives that state why the claim is proposed based on the data. The backing of this argument is the attributes of his objectives that further explain the warrant by providing its measurement scales. The measurement result given by the engineer for his own objectives is included in qualifier while the measurement result for the team's objectives is the rebuttal that describes his perspective regarding the performance regarding how well his argument may achieve the objectives proposed by the entire team.

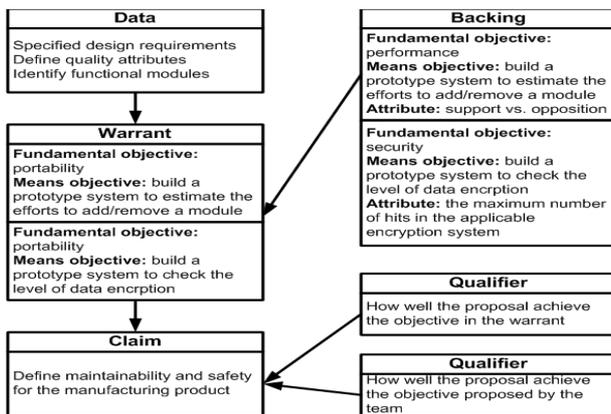


Figure 2 Argument Example

### 4) Argument Evaluation

As the stakeholders' arguments are generated and exchanged, their objectives and perspective models may evolve due to deepened understanding of each other. If all the stakeholders can jointly agree on a particular argument claim, they can take that claim as the final resolution. The evaluation method analyzes the stakeholder perspectives of the objectives within the arguments and compares the argument claims based on the result. In this work, a simple additive weighting function (a.k.a. weighted average) is used to build the evaluation method which ranks the arguments from most desired to least desired assuming stakeholders can characterize the consequences of each argument with certainty. Furthermore, "weighted average" is also applied when evaluating the arguments based on their value for the objective attributes with varying importance. Weighted average, by its definition, means an average that takes into account the proportional relevance and strength of each component, rather than treating each component equally.

The argument evaluation in our work includes three steps: (1) assign objective weights, (2) score the arguments and (3) aggregate the preferences. In these three steps, the objective weights and the score of attribute value (either natural attributes' or stakeholders' preferences) of arguments have been defined in previous steps. Therefore, in this step, the preferences and weights are aggregated to derive final argument evaluation results which are used to rank the arguments and select the one that is most preferred by the team. The calculation of final score for an argument defined as follows:

Definition 1: for a given argument  $A_j$

$$f_j = \sum_{i=1}^m c_i g_{ij} \quad (1)$$

Where  $f_j$  is the final score for alternative  $A_j$ ,  $m$  is the number of criteria,  $c_i$  is the normalize weight of attribute  $c_i$ ,  $g_{ij}$  is the performance grade (score) for argument  $A_j$  with respect to attribute  $c_i$ .

Based on the evaluation results, a most preferred argument (i.e. the one with highest evaluation score) will be recommended as the final agreement of the negotiation for the design task. They will move back to check for further decision conflicts with other tasks. These iterations continue until no more conflict is found (i.e. no more negotiation is necessary), and the team moves to the next phases of the software development lifecycle.

## III. CASE STUDY

To ensure that our framework can help software engineering team to reach an agreement more efficiently, a real-life software project was used for case studies. The team first used their existing way to collaborate and then applied our approach. Based on both results, we compared the time to reach the agreement. To assist the team to use our approach, a prototype of the argument-based collaborative negotiation (ACN) system, was implemented for stakeholder to follow. In this section, we will first introduce the ACN system and then demonstrate the comparison result.

### A. Argument-based collaborative negotiation system

The goal of our ACN system is to help users more effectively apply our negotiation approach. It was implemented

as a web-based application that uses Java Servlet Page (JSP). The system provides the link to each step of the negotiation process (i.e. the process indicator) and also reminds users what to do, such as the explanation of claim or warrant during the negotiation phase. Each stakeholder can view each other's proposed argument and the reason behind it. When stakeholders realize that there are different implementations in their software design tasks, they can activate the negotiation process. In addition, the system also provides tracking function for the evolving concepts and stakeholder perspective. Once the stakeholders start to choose a claim, the ACN system can automatically give the evaluation result from their proposed options. Therefore, the team can take the claim with the best score and continues their design work.

### B. The case study in real-life software design

We have applied our approach in a real-life team which was developing a consume software product on mobile devices. In this section, we will demonstrate one design task 'build the communication protocol' to show the application.

Their existing method for resolving the conflicts was to explore and discuss each other's arguments through the communication tools (e.g. email) or face to face meeting. In this way, the argument does not contain any specific structure; thus, one stakeholder just proposes his opinions and tries to get other's agreement. Therefore, the argument could be passed back and forth between team members and be revised based on the interaction iteratively until everyone in the team agrees. Below it is a dialog sample between an engineering manager and an engineer for determining an architecture design style.

Engineering manager: We need a database-centered architecture and here is why: because the database can help all the modules communicate with integral data.

Engineer: I see. But I still prefer a client-server application with Java Messaging Services. The messaging service can also enable all the modules to send or receive data.

Engineering manager: With database-centered design, we do not have to use Java Messaging Service.

Engineer: Sure, but why we choose the database-centered if the same advantage can be achieved by the other way?

Engineering manager: Let me think about it.

The team used the ACN system to conduct their negotiation in the development. In the beginning, the tool asked the team to assign a moderator to coordinate the negotiation. Then, through the aid of the tool, the baseline process was defined, the role of the stakeholder was identified, and the claim of the task was written. Figure 3 shows four different claims proposed by various stakeholders (e.g. product manager, engineering manager, engineer, and engineering director) regarding the task – build communication protocol.



Figure 3 Claim of Building Communication Protocol

Each stakeholder can fill in their objective hierarchy for others to refer. For example, Figure 4 shows the objective hierarchy of the product manager according his claim. With the support of the hierarchy, the manager added his fundamental objectives and the means objectives.

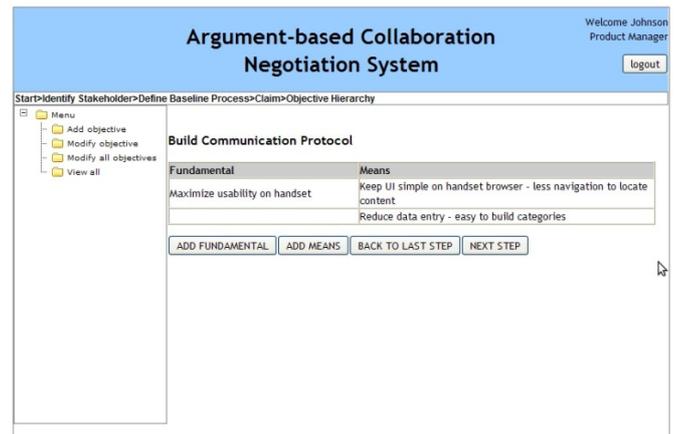


Figure 4 Objective Hierarchy of The Product Manager

### C. Comparison

Our goal of the case study was to compare the time that the team spent for negotiation and the number of rounds for generating/revising arguments, before the team reaches the agreement using the existing and proposed approaches respectively. We measured the time for the entire collaborative negotiation process, i.e., it included all the development tasks that need to be discussed and started with an agreement plan by the team. From the aforementioned case studies that were conducted in real-life software projects, it shows that this new approach has shortened the negotiation time and used fewer rounds to reach the design agreement. From the post-experiment survey and onsite observation, the negotiation procedure, arguments, and objective hierarchy helped stakeholder to use less time to understand each other's point and their concerns. The weight values and attributes also assisted the stakeholder to prioritize the importance of each claim. Although the value was subjective, it helped the team

members to review themselves' and others' opinions and was an aid to reach a more objective result.

#### IV. CONCLUSION AND FUTURE WORK

This paper describes a research approach to structure arguments with organized objectives and preferences of multiple stakeholders to support their group decision making in collaborative engineering design. It is established based on a synthesis between the generic argument structure and a value-focused objective hierarchy. This synthesis explains how stakeholders can generate the structured arguments according to their objectives and preferences. It helps us to better resolve the challenges in existing practices of generic argument structure by developing feasible ways of evaluating the arguments for the most preferred one by the team. As such, our work has developed a collaborative negotiation approach that utilizes these structured arguments including argument generation from stakeholders' objectives and preferences and argument evaluation to choose a most preferred argument based on how well the objectives have been achieved. In addition, this paper described a prototype system that we have developed to conduct case studies in real-life software development projects to validate the proposed approach.

Our future research work will develop more objective hierarchy templates in the software design domain and build more accurate and comprehensive models to quantify stakeholders' perspectives. Furthermore, we plan to thoroughly validate this research framework by conducting more case studies in software industry. When richer application results are gathered, the approach and system will be continuously improved to hopefully leading to the establishment of a scientific foundation for collaboration-based software engineering.

#### REFERENCES

- [1] S. Buckingham, A. MacLean, V. Bellotti, and N. Hammond (1997), "Graphical Argumentation and Design Cognition.", *Human-Computer Interaction*, 12(3), pp. 267-300.
- [2] M. R. Danesh and Y. Jin (1999), AND: An agent-based decision network for concurrent design and manufacturing. *Proc. of 1999 ASME Design Engineering Technical Conferences*. 1999, Las Vegas, Nevada.
- [3] S. Fatima, M. Wooldridge and N. R. Jennings (2002), Multi-issue negotiation under time constraints. *Proc. of the 1st International Joint Conference on Autonomous Agents and Multi-agent Systems (AAMAS-2002)*, ACM Press, 143-150.
- [4] P. Faratin (2000), Automated Service Negotiation Between Autonomous Computational Agents, Queen Mary and Westfield College, University of London.
- [5] X. Y. Gu (2002), Decision-Based Collaborative Optimization. *Journal of Mechanical Design*, 2002.
- [6] K. W. Houp, T. E. Pearsall, and E. Tebeaux (1998), "Reporting Technical Information", 9th Edition. Oxford UP New York. 1998.
- [7] T. Janssen and A. P. Sage (1996), Group decision support using Toulmin argument structures. *IEEE International Conference on Systems, Man, and Cybernetics*, Volume: 4, On page(s): 2704-2709 Vol.4, Oct. 1996.
- [8] R. L. Keeney (1992), Value-focused Thinking. Harvard University Press, Cambridge, 1992
- [9] R. Kowalczyk and V. Bui (2003), On constraint-based reasoning in e-negotiation agents, In Digham, F. & Cortes U. (eds.), *Agent-Mediated Electronic Commerce III (Lecture Notes in Computer Science, Vol. 2003)*, Berlin: Springer-Verlag, pp. 31-46
- [10] S. Toulmin (1958), *The uses of argument*. Cambridge University Press, London, 1958
- [11] S. Toulmin, R. Rieke and A. Janik (1984), *An Introduction to Reasoning*, Macmillan Publishing, New York, 1984.

# Comparing Two Approaches for Adding Feature Ranking to Sampled Ensemble Learning for Software Quality Estimation

Kehan Gao  
Eastern Connecticut State University  
gaok@easternct.edu

Taghi M. Khoshgoftaar & Amri Napolitano  
Florida Atlantic University  
khoshgof@fau.edu, amrifau@gmail.com

**Abstract**—High dimensionality and class imbalance are two main problems that affect the quality of training datasets in software defect prediction, resulting in inefficient classification models. Feature selection and data sampling are often used to overcome these problems. Feature selection is a process of choosing the most important attributes from the original data set. Data sampling alters the data set to change its balance level. Another technique, called boosting (building multiple models, with each model tuned to work better on instances misclassified by previous models), is found also effective for resolving the class imbalance problem. In particular, RUSBoost, which integrates random undersampling with AdaBoost, has been shown to improve classification performance for imbalanced training data sets. In this study, we investigated an approach for combining feature selection with this ensemble learning (boosting) process. We focused on two different scenarios: feature selection performed prior to the boosting process and feature selection performed inside the boosting process. Ten base feature ranking techniques and an ensemble ranker based on the ten were examined and compared over the two scenarios. The experimental results demonstrate that the ensemble feature ranking method generally had better or similar performance than the average of the base ranking techniques, and more importantly, the ensemble method exhibited better robustness than any other base ranking technique. As for the two scenarios, the results show that applying feature selection inside boosting performed better than using feature selection prior to boosting.

**Index Terms**—software defect prediction, feature selection, data sampling, boosting, base ranker, ensemble ranker

## I. INTRODUCTION

Software defect prediction is a process of utilizing classification models to classify program modules into quality-based classes, e.g., fault-prone (*fp*) or not-fault-prone (*nfp*) [1]. This kind of estimation can help practitioners effectively allocate limited project resources, making them focus on program modules that are of poor quality or likely to have a high number of faults. Classification models are usually built using software metrics such as code-level measurements and defect data, along with data mining techniques. A considerable amount of research has shown that prediction accuracy of a classification model is affected by the quality of the input (training) data. We are interested in investigating two common problems, high dimensionality and class imbalance, that exist in many software measurement data sets.

High dimensionality occurs when a large number of variables (features or software metrics) are available for building classification models. Several problems may arise due to high dimensionality, including longer learning time, a decline in prediction performance, and difficulty of understanding and interpreting the model. The purpose of feature selection is to choose the most important attributes from the original data set so that prediction performance will be improved, or at least maintained, while learning time is significantly reduced. Recent research [2] has shown that filter-based feature ranking techniques are simple, fast, and effective methods for dealing with this problem. In this study, we examined ten filter-based feature ranking techniques and an ensemble ranker based on the ten.

Class imbalance is a separate problem, wherein the class ratio is especially skewed. In a two-class classification problem, class imbalance manifests as one class (minority) having far fewer instances than the other class (majority). The main disadvantage of such imbalanced training data is that a traditional classification algorithm would tend to classify minority class instances (usually a class of interest, e.g., *fp* modules) as majority class (e.g., *nfp* modules) in order to increase overall prediction accuracy. In software quality engineering, this may result in buggy software in deployment and operation, thereby causing serious consequences and high repair cost.

Many solutions have been proposed to address the class imbalance problem. A simple but effective method is random undersampling (RUS), where instances from the majority class are randomly discarded until a certain balance (ratio) between the majority and the minority classes is achieved. Another technique, called boosting, is also effective to cope with the class imbalance problem. Boosting is a meta-learning technique designed to improve the classification performance of weak learners by building multiple models, with each model tuned to work better on instances misclassified by previous models. Although boosting is not specifically developed to handle the class imbalance problem, it has been shown to be very effective in this regard [3]. In this study, we use an ensemble learning approach, called RUSBoost, in which random undersampling (RUS) is integrated into AdaBoost [4].

To deal with both high dimensionality and class imbalance, we studied an approach which combines feature selection with an ensemble learning method (RUSBoost). We investigated two different scenarios: feature selection performed right before RUSBoost and feature selection performed inside RUSBoost. In this study, we are interested in learning the impact of feature selection on the final classification results. More specifically, we would like to investigate ten base feature ranking techniques and the ensemble ranker based on the ten, and compare their behaviors, including prediction performance and stability with respect to different learners and data sets. The experiment was conducted over the two different scenarios discussed. We used two groups of data sets from a real-world software system, all of which exhibit significant class imbalance between the two classes (*fp* and *nfp*). Five different learners were used to build classification models. The experimental results demonstrate that the ensemble ranker resulted in better or similar prediction performance than the average of the ten base rankers. In addition, the ensemble method displayed more stable behavior than most base rankers including the best base ranking technique. As to the two different scenarios of the approach, the result shows that feature selection performed inside RUSBoost provided better classification performance than when it was applied prior to RUSBoost.

The rest of the paper is organized as follows. Section II discusses related work. Section III provides methodology, including more detailed information about the feature selection, ensemble-

based boosting technique, learners, and performance metric applied in this study. The data sets used in the experiments are described in Section IV. Section V presents the experimental results. Finally, the conclusion and future work are summarized in Section VI.

## II. RELATED WORK

Significant research has been dedicated towards feature selection [5], and applied to a range of fields. Van Hulse et al. [2] proposed 11 new threshold-based feature selection techniques and applied them to 17 different bioinformatics datasets. Yu et al. applied feature selection to gene expression microarray data and studied the stability of feature selection via sample weighting [6].

In addition to excess number of features, many datasets are plagued with the class imbalance problem. Two techniques that have been discussed for addressing this problem are data sampling and boosting. The simplest form of sampling is random sampling. In addition, a few more intelligent algorithms for sampling data have been proposed [7]. The most commonly used boosting algorithm is AdaBoost [4]. Several variations have been proposed to improve AdaBoost's performance on imbalanced data. One promising technique is RUSBoost [3], which is a highly effective hybrid approach to learning from imbalanced data.

While a great deal of effort has been dedicated toward feature selection and data sampling separately for many years, research working on both together is starting to receive more attention [8]. Yang et al. [9] proposed an ensemble-based wrapper approach for feature selection from data with highly imbalanced class distribution. In this work, we create an approach that combines feature selection with RUSBoost and study two different combination scenarios. We also compare different base feature ranking techniques and the ensemble ranker over the two scenarios.

## III. METHODOLOGY

### A. Filter-Based Feature Ranking Techniques

The goal of feature ranking is to score each feature according to a particular method, allowing the selection of the best features. In this study, we investigated ten individual feature ranking techniques from three different categories: three standard methods, six threshold-based feature selection techniques, and the signal to noise ratio approach. In addition, we studied the use of an ensemble of feature ranking techniques.

1) *Standard Techniques*: The three standard filter-based feature ranking techniques used in this work include: chi-squared, information gain, and ReliefF. All three use the implementation found in the WEKA tool<sup>1</sup> [10] with default parameters. The chi-squared (CS) test evaluates the worth of a feature by computing the value of the chi-squared statistic with respect to the class. The null hypothesis is the assumption that the two features are unrelated, and it is tested by the chi-squared ( $\chi^2$ ) formula:  $\chi^2 = \sum_{i=1}^r \sum_{j=1}^c \frac{(O_{ij} - E_{ij})^2}{E_{ij}}$ , where  $O_{ij}$  is the observed frequency and  $E_{ij}$  is the expected (theoretical) frequency, asserted by the null hypothesis. The greater the value of  $\chi^2$ , the greater the evidence against the null hypothesis. Information gain (IG) is a measure based on the concept of entropy from information theory. IG is the information provided about the target class attribute  $Y$ , given the value of another attribute  $X$ . IG measures the decrease of the weighted average impurity of the partitions,

<sup>1</sup>Waikato Environment for Knowledge Analysis (WEKA) is a popular suite of machine learning software written in Java, developed at the University of Waikato. WEKA is free software available under the GNU General Public License.

---

### Algorithm 1: Threshold-Based Feature Selection

---

**input** :

1. Dataset  $S = \{(\mathbf{x}_i, y_i) | i = 1, \dots, n \text{ and } y_i \in \{P, N\}\}$  with features  $F^j, j = 1, \dots, m$ , where  $P = fp$  and  $N = nfp$ ;

2. The value of attribute  $F^j$  for instance  $\mathbf{x}_i$  is denoted  $F^j(\mathbf{x}_i)$ ;

3. Metric  $\omega \in \{\text{CS, IG, RF, MI, KS, Dev, GM, ROC, PRC, S2N}\}$ .

**output**: Ranking  $\mathbb{R} = \{r^1, \dots, r^m\}$  where  $r^j$  represents the rank for attribute  $F^j$ , i.e., the  $r^j$ -th most significant attribute as determined by metric  $\omega$ .

**for**  $F^j, j = 1, \dots, m$  **do**

Normalize  $F^j \mapsto \hat{F}^j = \frac{F^j - \min(F^j)}{\max(F^j) - \min(F^j)}$ ;

Calculate metric  $\omega$  using attribute  $\hat{F}^j$  and class attribute  $\{y_i | y_i \in \{P, N\}, i = 1, \dots, n\}$ ,  $\omega(\hat{F}^j)$ ; (The detailed formula of each metric  $\omega$  is provided in Section III-A2.)

Create attribute ranking  $\mathbb{R}$  using  $\omega(\hat{F}^j) \forall j$

---

compared with the impurity of the complete set of data. Relief is an instance-based feature ranking technique. ReliefF is an extension of the relief algorithm that can handle noise and multiclass data sets.

2) *Threshold-Based Feature Selection*: The threshold-based feature selection (TBFS) technique was proposed by our research team and implemented within WEKA [2]. The procedure is shown in Algorithm 1. Each independent attribute works individually with the class attribute, and this two-attribute data set is evaluated using different classification performance metrics. More specifically, the TBFS procedure includes two steps: (1) normalizing the attribute values so that they fall between 0 and 1; and (2) treating those values as the posterior probabilities from which to calculate performance metrics. The feature rankers we propose utilize four rates<sup>2</sup>. The value is computed in both directions: first treating instances above the threshold ( $t$ ) as positive and below as negative, then treating instances above the threshold as negative and below as positive. The better result is used. In this manner, the attributes can be ranked from most to least predictive based on each metric. Six metrics used in this study are presented as follows:

- a. Mutual Information (MI). Let  $c(\mathbf{x}) \in \{P, N\}$  denote the actual class of instance  $\mathbf{x}$ , and let  $\hat{c}^t(\mathbf{x})$  denote the predicted class based on the value of the attribute  $F^j$  and a given threshold  $t$ . MI computes the mutual information criterion with respect to the number of times a feature value and a class co-occur, the feature value occurs without the class, and the class occurs without the feature value. The MI metric is defined as:

$$\text{MI} = \max_{t \in [0, 1]} \sum_{\hat{c}^t \in \{P, N\}} \sum_{c \in \{P, N\}} p(\hat{c}^t, c) \log \frac{p(\hat{c}^t, c)}{p(\hat{c}^t)p(c)}$$

where

$$p(\hat{c}^t = \alpha, c = \beta) = \frac{|\{\mathbf{x} | (\hat{c}^t(\mathbf{x}) = \alpha) \cap (c(\mathbf{x}) = \beta)\}|}{|P| + |N|},$$

$$p(\hat{c}^t = \alpha) = \frac{|\{\mathbf{x} | \hat{c}^t(\mathbf{x}) = \alpha\}|}{|P| + |N|},$$

$$p(c = \alpha) = \frac{|\{\mathbf{x} | c(\mathbf{x}) = \alpha\}|}{|P| + |N|},$$

$$\alpha, \beta \in \{P, N\}.$$

<sup>2</sup>Analogous to the procedure for calculating rates in a classification setting with a posterior probability, the true positive rate, TPR( $t$ ), true negative rate, TNR( $t$ ), and false positive rate, FPR( $t$ ) can be calculated at each threshold  $t \in [0, 1]$  relative to the normalized attribute  $\hat{F}^j$ . Precision, PRE( $t$ ) is defined as the fraction of the predicted-positive examples which are actually positive.

- b. Kolmogorov-Smirnov statistic (KS). KS is a measurement of separability. The goal of KS is to measure the maximum difference between the distributions of the members of each class. The formula for the KS statistic is:

$$KS = \max_{t \in [0,1]} |TPR(t) - FPR(t)|$$

- c. Deviance (Dev). Dev, like GI, is a metric in which it is the minimum value over all the thresholds that is the chosen value for the attribute. Deviance measures the sum of the squared errors from the mean class based on a threshold  $t$ .
- d. Geometric Mean (GM). GM is a quick and useful metric for feature selection. The equation for the geometric mean is

$$GM = \max_{t \in [0,1]} \sqrt{TPR(t) \times TNR(t)}$$

A geometric mean of 1 would mean that the attribute is perfectly correlated. The maximum geometric mean across the thresholds is the score of the attribute.

- e. Area Under the ROC Curve (ROC). Receiver Operating Characteristic, or ROC, curves graph true positive rate on the  $y$ -axis versus the false positive rate on the  $x$ -axis. The resulting curve illustrates the trade-off between true positive rate and false positive rate. In this study, ROC curves are generated by varying the decision threshold  $t$  used to transform the normalized attribute values into a predicted class. The area under the ROC ranges from 0 to 1, and an attribute with more predictive power results in an area under the ROC closer to 1.
- f. Area Under the Precision-Recall Curve (PRC). PRC is a single-value measure that originated from the area of information retrieval. A precision-recall curve is generated by varying the decision threshold  $t$  from 0 to 1 and plotting the recall (equivalent to true positive rate) on the  $y$ -axis and precision on the  $x$ -axis at each point in a similar manner to the ROC curve. The area under the PRC ranges from 0 to 1, and an attribute with more predictive power results in an area under the PRC closer to 1.

3) *Signal to Noise Ratio*: Signal to noise ratio (S2N) [11] is a simple univariate ranking technique which defines how well a feature discriminates between two classes in a two class problem. S2N, for a given feature, separates the means of the two classes relative to the sum of their standard deviation. The formula to calculate S2N is  $S2N = \frac{(\mu_P - \mu_N)}{\sigma_P + \sigma_N}$ , where  $\mu_P$  and  $\mu_N$  are the mean values of a particular attribute for the samples from class  $P$  and class  $N$ , and  $\sigma_P$  and  $\sigma_N$  are the corresponding standard deviations. The larger the S2N value, the more relevant the feature is to the class attribute.

4) *Ensemble Filter Technique*: This approach combines different ranking techniques to yield more stable and robust results. The procedure of the ensemble technique includes two essential steps. First, a set of different ranking lists is created using corresponding filter-based rankers and input to the next combining step; second, these ranking lists are integrated using an aggregation technique. Diversity can be achieved by using various rankers [12]. In this study, we use the ten filters discussed above to form the ensemble filter. The aggregation method used is arithmetic mean, where each feature's score is determined by the average of the ranking scores of the feature in each ranking list. Finally, the highest ranked attributes are selected to be used.

#### B. The RUSBoost Technique

RUSBoost combines random undersampling (RUS) and boosting for improving classification performance. Boosting is a meta-learning

technique designed to improve the classification performance of weak learners by iteratively creating an ensemble of weak hypotheses which are combined to predict the class of unlabeled examples. This study uses AdaBoost [4], a well-known boosting algorithm shown to improve the classification performance of weak classifiers. Initially, all examples in the training data set are assigned equal weights. During each iteration of AdaBoost, a weak hypothesis is formed by the base learner. The error associated with the hypothesis is calculated and the weight of each example is adjusted such that misclassified examples have their weights increased while correctly classified examples have their weights decreased. Therefore, subsequent iterations of boosting will generate hypotheses that are more likely to correctly classify the previously mislabeled examples. After all iterations are completed a weighted vote of all hypotheses are used to assign a class to unlabeled examples. In this study, the boosting algorithm is performed using 10 iterations. RUSBoost applies the same steps as the regular boosting, but prior to constructing the weak hypothesis during each round of boosting, random undersampling is applied to the training data to achieve a more balanced class distribution. The base learners used in this work are NB, MLP, KNN, SVM, and LR, and these will be discussed in the next section. The procedure of RUSBoost is described in part of Figure 1. More details about the algorithm can be found in [3].

#### C. Learners

The software defect prediction models in this study are built using five different classification algorithms, including Naive Bayes (NB) [10], MultiLayer Perceptron (MLP) [13], K Nearest Neighbors (KNN) [10], Support Vector Machine (SVM) [14], and Logistic Regression (LR) [10]. Due to space limitations, we refer interested readers to these references to understand how these commonly-used learners function. The WEKA tool is used to instantiate the different classifiers. Generally, the default parameter settings for the different learners are used (for NB and LR), except for the below-mentioned changes. A preliminary investigation in the context of this study indicated that the modified parameter settings are appropriate.

In the case of MLP, the `hiddenLayers` parameter was changed to '3' to define a network with one hidden layer containing three nodes, and the `validationSetSize` parameter was changed to '10' to cause the classifier to leave 10% of the training data aside for use as a validation set to determine when to stop the iterative training process. For the KNN learner, the `distanceWeighting` parameter was set to 'Weight by 1/distance', the `kNN` parameter was set to '5', and the `crossValidate` parameter was turned on (set to 'true'). In the case of SVM, two changes were made: the `complexity constant c` was set to '5.0', and `build Logistic Models` was set to 'true'. A linear kernel was used by default.

#### D. Performance Metric

One of the most popular methods for evaluating the performance of learners built using imbalanced data is *receiver operating characteristic*, or ROC, curves. The ROC curve illustrates the performance of a classifier across the complete range of possible decision thresholds, and accordingly does not assume any particular misclassification costs or class prior probabilities. The area under the ROC curve (AUC) is used to provide a single numerical metric for comparing model performances. The AUC value ranges from 0 to 1.

## IV. DATASETS

In our experiments, we used publicly available data, namely the Eclipse defect counts and complexity metrics data set obtained from

TABLE I  
DATA CHARACTERISTICS

Dataset	Rel.	thd	#Attr.	#Inst.	#fp	%fp	#nfp	%nfp
Eclipse 1	2.0	10	209	377	23	6.1	354	93.9
	2.1	5	209	434	34	7.8	400	92.2
	3.0	10	209	661	41	6.2	620	93.8
Eclipse 2	2.0	5	209	377	52	13.8	325	86.2
	2.1	4	209	434	50	11.5	384	88.5
	3.0	5	209	661	98	14.8	563	85.2

the PROMISE data repository (<http://promisedata.org>). In particular, we used the metrics and defects data at the software packages level. The original data for the Eclipse packages consists of three releases denoted 2.0, 2.1, and 3.0, each release reported by Zimmerman et al. [15]. Membership in each class is determined by a post-release defects threshold *thd*, which separates *fp* from *nfp* packages by classifying packages with *thd* or more post-release defects as *fp* and the remaining as *nfp*. In our study, we used  $thd = \{10, 5\}$  for releases 2.0 and 3.0 and  $thd = \{5, 4\}$  for release 2.1. This resulted in two groups. Each group contains three data sets, one for each release. The reason why a different set of thresholds was chosen for release 2.1 is that we would like to keep similar class distributions for the data sets in the same group. All data sets contain 209 attributes (208 independent attributes and 1 dependent attribute). Table I shows the characteristics of the data sets after transformation for each group.

## V. EXPERIMENTS

### A. Design

The main objective of this study is to evaluate the individual filters and the ensemble filter. More specifically, we would like to investigate the ten individual filters (as discussed in Section III-A) with the ensemble filter based on those ten. This comparison was carried out in two different scenarios of feature selection combined with the ensemble learning (boosting) approach. The process of the ensemble learning approach is presented in Figure 1. The two different scenarios are highlighted at top.

- **Scenario 1:** External Feature Selection (EFS), i.e., feature selection performed **prior** to the ensemble learning process.
- **Scenario 2:** Internal Feature Selection (IFS), i.e., feature selection performed **inside** the ensemble learning process.

In the experiment, 11 feature selection methods, including ten base rankers (CS, IG, RF, MI, KS, Dev, GM, ROC, PRC and S2N) and an ensemble ranker based on the ten, were applied. The sampling technique used was the random undersampling (RUS) approach. The post-sampling class ratio (between *fp* and *nfp* modules) was set to 50:50 throughout the experiment. In addition, five base learners (NB, MLP, KNN, SVM, and LR) were used in the boosting process. The number of features selected in the feature subsets was set to  $\lceil \log_2 n \rceil = 8$ , where  $n$  is the number of independent attributes in the original data set ( $n = 208$  in this experiment). For all experiments, we employed ten runs of five-fold cross-validation. That is, for each run the data was randomly divided into five folds, one of which was used as the test data while the other four folds were used as training data. All the preprocessing steps (feature selection and data sampling) were done on the training data set. The processed training data was then used to build the classification model and the resulting model was applied to the test fold. This cross-validation was repeated five times (the folds), with each fold used exactly once as the test data. The five results from the five folds then was averaged to produce a single estimation.

TABLE II  
CLASSIFICATION PERFORMANCE FOR EXTERNAL FEATURE SELECTION

(a) Eclipse 1

Ranker	NB	MLP	KNN	SVM	LR
Ensemble	0.8399 <sup>4</sup>	0.8908 <sup>3</sup>	0.8846 <sup>5</sup>	0.9045 <sup>6</sup>	0.8729 <sup>6</sup>
CS	0.8000 <sup>11</sup>	0.8847 <sup>8</sup>	0.8768 <sup>8</sup>	0.8980 <sup>8</sup>	0.8598 <sup>11</sup>
IG	0.8409 <sup>3</sup>	0.8873 <sup>7</sup>	0.8891 <sup>3</sup>	0.9077 <sup>4</sup>	0.8850 <sup>3</sup>
RF	0.8204 <sup>10</sup>	0.8636 <sup>11</sup>	0.8100 <sup>11</sup>	0.8725 <sup>11</sup>	0.8723 <sup>7</sup>
MI	0.8246 <sup>7</sup>	0.8900 <sup>5</sup>	0.8821 <sup>7</sup>	0.9045 <sup>7</sup>	0.8853 <sup>2</sup>
KS	0.8294 <sup>5</sup>	0.8745 <sup>9</sup>	0.8674 <sup>9</sup>	0.8953 <sup>9</sup>	0.8613 <sup>10</sup>
Dev	0.8222 <sup>9</sup>	0.8881 <sup>6</sup>	0.8839 <sup>6</sup>	0.9054 <sup>5</sup>	0.8710 <sup>8</sup>
GM	0.8247 <sup>6</sup>	0.8721 <sup>10</sup>	0.8673 <sup>10</sup>	0.8927 <sup>10</sup>	0.8655 <sup>9</sup>
ROC	0.8532 <sup>1</sup>	0.8983 <sup>2</sup>	0.8985 <sup>1</sup>	0.9106 <sup>1</sup>	0.8759 <sup>4</sup>
PRC	0.8239 <sup>8</sup>	0.8908 <sup>4</sup>	0.8922 <sup>2</sup>	0.9085 <sup>3</sup>	0.8732 <sup>5</sup>
S2N	0.8449 <sup>2</sup>	0.9005 <sup>1</sup>	0.8887 <sup>4</sup>	0.9089 <sup>2</sup>	0.8955 <sup>1</sup>

(b) Eclipse 2

Ranker	NB	MLP	KNN	SVM	LR
Ensemble	0.8364 <sup>7</sup>	0.9084 <sup>5</sup>	0.8962 <sup>6</sup>	0.9183 <sup>7</sup>	0.9108 <sup>2</sup>
CS	0.8236 <sup>11</sup>	0.9076 <sup>6</sup>	0.8942 <sup>7</sup>	0.9183 <sup>8</sup>	0.9047 <sup>9</sup>
IG	0.8275 <sup>10</sup>	0.9090 <sup>4</sup>	0.8971 <sup>3</sup>	0.9210 <sup>2</sup>	0.9061 <sup>7</sup>
RF	0.8319 <sup>9</sup>	0.8587 <sup>11</sup>	0.8056 <sup>11</sup>	0.8752 <sup>11</sup>	0.8747 <sup>11</sup>
MI	0.8424 <sup>3</sup>	0.9058 <sup>8</sup>	0.8963 <sup>5</sup>	0.9210 <sup>1</sup>	0.9090 <sup>4</sup>
KS	0.8346 <sup>8</sup>	0.9040 <sup>9</sup>	0.8925 <sup>9</sup>	0.9185 <sup>6</sup>	0.9069 <sup>6</sup>
Dev	0.8395 <sup>5</sup>	0.9101 <sup>2</sup>	0.8986 <sup>2</sup>	0.9193 <sup>5</sup>	0.9089 <sup>5</sup>
GM	0.8414 <sup>4</sup>	0.9015 <sup>10</sup>	0.8905 <sup>10</sup>	0.9169 <sup>9</sup>	0.9044 <sup>10</sup>
ROC	0.8427 <sup>2</sup>	0.9125 <sup>1</sup>	0.8971 <sup>1</sup>	0.9209 <sup>3</sup>	0.9120 <sup>1</sup>
PRC	0.8390 <sup>6</sup>	0.9091 <sup>3</sup>	0.8970 <sup>4</sup>	0.9194 <sup>4</sup>	0.9060 <sup>8</sup>
S2N	0.8533 <sup>1</sup>	0.9066 <sup>7</sup>	0.8927 <sup>8</sup>	0.9136 <sup>10</sup>	0.9098 <sup>3</sup>

### B. Results and Analysis

The results of the two scenarios (EFS and IFS) averaged over the respective group of data sets (in terms of AUC) are reported in Tables II and III, each containing the results for all five learners. The tables show the classification performance of the ten individual filters as well as the ensemble filter based on the ten. Each value has a superscript that represents the rank of the filter among the 11 techniques. For instance, the ensemble filter ranks #4 among all the 11 ranking techniques when NB was employed on Eclipse 1 for the EFS scenario. From the results, we can see that the performance of the rankers were related to learners, training data set, and working scenarios. There was no method that always performed the best in all circumstances. For example, ROC performed relatively better than other rankers for the EFS scenario, while it performed averagely for the IFS scenario. IG performed best for Eclipse 1 in IFS when the MLP and SVM learners were employed, however showed the worst prediction among the 11 rankers when NB was used. The ensemble filter generally performed better than average of the individual base rankers and also displayed more stable performance than most base rankers (as the rank deviation of the ensemble filter is the second smallest among 11, GM had the smallest rank deviation, however, GM performed below the average). Figure 2 provides the comparison between the ensemble filter and the average of the ten base rankers for a given scenario and a data set across all five learners. The charts intuitively demonstrate that the ensemble method performed better than or similarly to the average.

We further carried out a three-way analysis of variance (ANOVA) F-test on the classification performance to examine if the performance difference (better/worse) is statistically significant or not. The three factors include: A representing two different scenarios of the approach, B representing the five learners, and C representing the ten base rankers and the ensemble method. The null hypothesis for the ANOVA test is that all the group population means are the same,

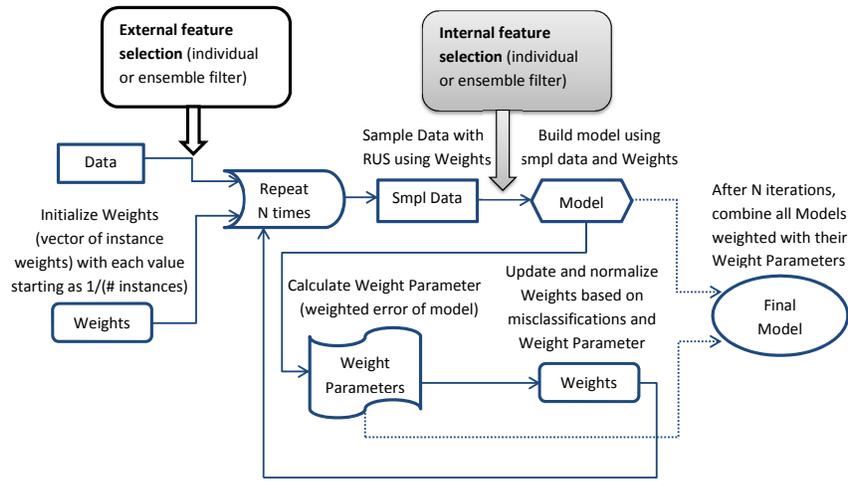


Fig. 1. Two scenarios of feature selection combined with the ensemble learning approach

TABLE III  
CLASSIFICATION PERFORMANCE FOR INTERNAL FEATURE SELECTION

(a) Eclipse 1

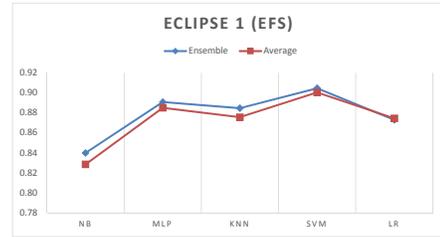
Ranker	NB	MLP	KNN	SVM	LR
Ensemble	0.8561 <sup>3</sup>	0.8985 <sup>5</sup>	0.9033 <sup>2</sup>	0.9105 <sup>2</sup>	0.8803 <sup>5</sup>
CS	0.8398 <sup>10</sup>	0.8931 <sup>9</sup>	0.8935 <sup>10</sup>	0.9096 <sup>3</sup>	0.8746 <sup>11</sup>
IG	0.8385 <sup>11</sup>	0.9042 <sup>1</sup>	0.9031 <sup>3</sup>	0.9159 <sup>1</sup>	0.8840 <sup>2</sup>
RF	0.8563 <sup>2</sup>	0.8874 <sup>11</sup>	0.8919 <sup>11</sup>	0.8975 <sup>11</sup>	0.8784 <sup>8</sup>
MI	0.8469 <sup>8</sup>	0.8935 <sup>8</sup>	0.8953 <sup>9</sup>	0.9078 <sup>6</sup>	0.8834 <sup>3</sup>
KS	0.8508 <sup>7</sup>	0.8966 <sup>7</sup>	0.8966 <sup>7</sup>	0.9087 <sup>5</sup>	0.8802 <sup>6</sup>
Dev	0.8551 <sup>4</sup>	0.8988 <sup>4</sup>	0.8988 <sup>6</sup>	0.9094 <sup>4</sup>	0.8798 <sup>7</sup>
GM	0.8511 <sup>6</sup>	0.8922 <sup>10</sup>	0.8956 <sup>8</sup>	0.9054 <sup>8</sup>	0.8755 <sup>10</sup>
ROC	0.8543 <sup>5</sup>	0.9006 <sup>3</sup>	0.8998 <sup>5</sup>	0.9076 <sup>7</sup>	0.8762 <sup>9</sup>
PRC	0.8669 <sup>1</sup>	0.8967 <sup>6</sup>	0.9035 <sup>1</sup>	0.9044 <sup>9</sup>	0.8856 <sup>1</sup>
S2N	0.8452 <sup>9</sup>	0.9018 <sup>2</sup>	0.9012 <sup>4</sup>	0.9044 <sup>10</sup>	0.8803 <sup>4</sup>

(b) Eclipse 2

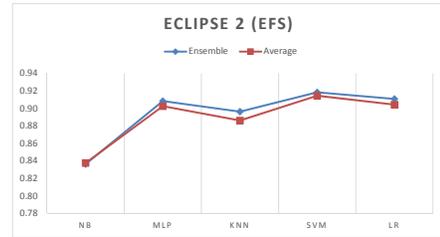
Ranker	NB	MLP	KNN	SVM	LR
Ensemble	0.8565 <sup>2</sup>	0.9055 <sup>8</sup>	0.9065 <sup>4</sup>	0.9147 <sup>8</sup>	0.9118 <sup>1</sup>
CS	0.8204 <sup>11</sup>	0.9085 <sup>4</sup>	0.9052 <sup>7</sup>	0.9175 <sup>1</sup>	0.9110 <sup>2</sup>
IG	0.8274 <sup>10</sup>	0.9103 <sup>1</sup>	0.9054 <sup>6</sup>	0.9169 <sup>2</sup>	0.9099 <sup>6</sup>
RF	0.8585 <sup>1</sup>	0.8923 <sup>11</sup>	0.8936 <sup>11</sup>	0.9032 <sup>11</sup>	0.9014 <sup>11</sup>
MI	0.8491 <sup>5</sup>	0.9059 <sup>7</sup>	0.9065 <sup>3</sup>	0.9164 <sup>3</sup>	0.9102 <sup>5</sup>
KS	0.8486 <sup>6</sup>	0.9095 <sup>2</sup>	0.9055 <sup>5</sup>	0.9162 <sup>4</sup>	0.9095 <sup>9</sup>
Dev	0.8420 <sup>8</sup>	0.9073 <sup>5</sup>	0.9043 <sup>9</sup>	0.9161 <sup>5</sup>	0.9105 <sup>4</sup>
GM	0.8519 <sup>4</sup>	0.9068 <sup>6</sup>	0.9047 <sup>8</sup>	0.9153 <sup>6</sup>	0.9097 <sup>7</sup>
ROC	0.8483 <sup>7</sup>	0.9090 <sup>3</sup>	0.9076 <sup>2</sup>	0.9139 <sup>9</sup>	0.9095 <sup>8</sup>
PRC	0.8543 <sup>3</sup>	0.9043 <sup>9</sup>	0.9079 <sup>1</sup>	0.9151 <sup>7</sup>	0.9108 <sup>3</sup>
S2N	0.8385 <sup>9</sup>	0.9030 <sup>10</sup>	0.9040 <sup>10</sup>	0.9103 <sup>10</sup>	0.9061 <sup>10</sup>

TABLE IV  
ANOVA FOR THE ECLIPSE DATA SETS

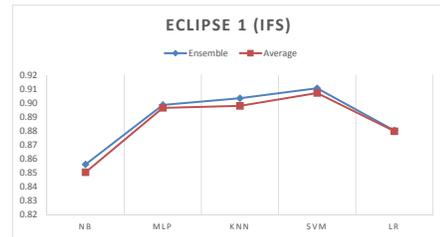
Source	Sum Sq.	d.f.	Mean Sq.	F	p-value
Scenario	0.1591	1	0.1591	133.27	0.000
Learner	3.6895	4	0.9224	772.49	0.000
Ranker	0.2963	10	0.0296	24.82	0.000
Error	7.8616	6584	0.0012		
Total	12.0065	6599			



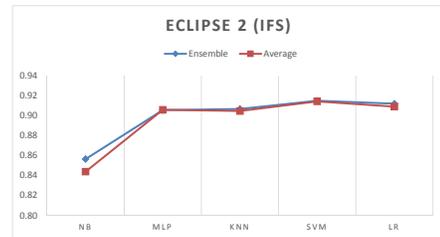
(a) Eclipse 1 (EFS)



(b) Eclipse 2 (EFS)



(c) Eclipse 1 (IFS)



(d) Eclipse 2 (IFS)

Fig. 2. Comparison between the ensemble and average of the ten individual filters in EFS and IFS scenarios

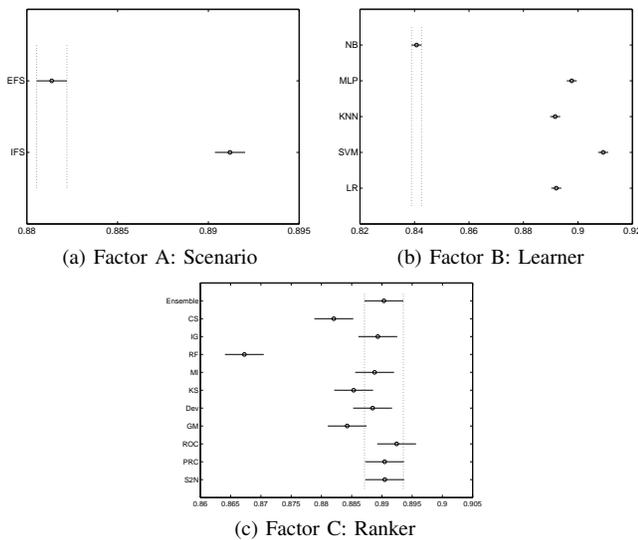


Fig. 3. Multiple comparison for Eclipse 1 and 2 data sets

while the alternate hypothesis is that at least one pair of means is different. Note that the two groups of data sets were analyzed together. Table IV shows the ANOVA results. The  $p$ -value is less than the cutoff 0.05 for all factors, meaning that for each main factor the alternate hypothesis is accepted, namely, at least two group means are significantly different from each other.

We further conducted a multiple comparison test on each main factor with Tukey's honestly significant difference (HSD) criterion. Note that for all the ANOVA and multiple comparison tests, the significance level was set to 0.05. Figure 3 shows the multiple comparisons for Factors A, B, and C. The figures display graphs with each group mean represented by a symbol ( $\circ$ ) and 95% confidence interval as a line around the symbol. Two means are significantly different if their intervals are disjoint, and are not significantly different if their intervals overlap. The assumptions for constructing ANOVA and Tukey's HSD models were validated. From these figures we can see the following points:

- Feature selection performed inside RUSBoost (IFS) resulted in better classification performance than when it is applied prior to RUSBoost (EFS).
- For the five learners, SVM presented the best performance, followed by MLP, then KNN and LR; NB performed the worst.
- Among the 11 feature ranking techniques, the ensemble method demonstrated competitive performance, ranking right behind the best performer, ROC, and had the same performance as PRC and S2N; IG, MI, and Dev were in the following group, then KS and GM, although the ensemble ranker did not show statistical difference between itself and those methods; CS and RF had the worst performance.

## VI. CONCLUSION

In this study, we proposed feature selection combined with an ensemble learning (boosting) technique to overcome the high dimensionality and class imbalance problems that often affect software quality classification. Two different scenarios (IFS and EFS) of the technique that combines feature selection with boosting were investigated. IFS (internal feature selection) refers to when feature selection is applied inside the boosting process, while EFS (external feature selection) refers to when feature selection takes place

prior to the boosting process. In this research, we were interested in investigating ten individual feature ranking techniques and the ensemble ranker based on the ten, and comparing them for the two scenarios. In the experiment, we applied these techniques to two groups of data sets from a real-world software system. We built classification models using five learners. The results demonstrate that the ensemble technique performed better than or similarly to the average of the ten base rankers, and more importantly, the ensemble ranker yielded more stable and robust performance than any other competing base rankers. In addition, the results show that performing feature selection inside of boosting generally performed better than using feature selection prior to boosting. Of the five learners, support vector machine performed the best, followed by multilayer perceptron, k nearest neighbors, and logistic regression; naïve Bayes had the worst prediction performance. Future work will involve conducting additional empirical studies with software measurement and defect data from other software projects.

## REFERENCES

- [1] Q. Song, Z. Jia, M. Shepperd, S. Ying, and J. Liu, "A general software defect-proneness prediction framework," *IEEE Transactions on Software Engineering*, vol. 37, no. 3, pp. 356–370, May/June 2011.
- [2] J. Van Hulse, T. M. Khoshgoftaar, A. Napolitano, and R. Wald, "Threshold-based feature selection techniques for high-dimensional bioinformatics data," *Network Modeling Analysis in Health Informatics and Bioinformatics*, vol. 1, no. 1-2, pp. 47–61, June 2012.
- [3] C. Seiffert, T. M. Khoshgoftaar, J. Van Hulse, and A. Napolitano, "Rusboost: A hybrid approach to alleviating class imbalance," *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 40, no. 1, pp. 185–197, Jan. 2010.
- [4] Y. Freund and R. E. Schapire, "Experiments with a new boosting algorithm," in *Proceedings of the 13th International Conference on Machine Learning*, 1996, pp. 148–156.
- [5] H. Liu, H. Motoda, R. Setiono, and Z. Zhao, "Feature selection: An ever evolving frontier in data mining," in *Proceedings of the Fourth International Workshop on Feature Selection in Data Mining*, Hyderabad, India, 2010, pp. 4–13.
- [6] L. Yu, Y. Han, and M. E. Berens, "Stable gene selection from microarray data via sample weighting," *IEEE/ACM Transactions On Computational Biology and Bioinformatics*, vol. 9, no. 1, pp. 262–272, Jan/Feb 2012.
- [7] N. V. Chawla, K. W. Bowyer, L. O. Hall, and P. W. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002.
- [8] T. M. Khoshgoftaar, K. Gao, A. Napolitano, and R. Wald, "A comparative study of iterative and non-iterative feature selection techniques for software defect prediction," *Information Systems Frontiers*, pp. 1–22, April 2013.
- [9] P. Yang, W. Liu, B. B. Zhou, S. Chawla, and A. Y. Zomaya, "Ensemble-based wrapper methods for feature selection and class imbalance learning," in *Proceedings of the 17th Pacific-Asia Conference, PAKDD 2013, Gold Coast, Australia, Part I, Lecture Notes in Computer Science 7818*. Springer-Verlag, April 14–17 2013, pp. 544–555.
- [10] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd ed. Morgan Kaufmann, 2005.
- [11] L. Goh, Q. Song, and N. Kasabov, "A novel feature selection method to improve classification of gene expression data," in *Proceedings of the Second Conference on Asia-Pacific Bioinformatics*, Dunedin, New Zealand, 2004, pp. 161–166.
- [12] J. S. Olsson and D. W. Oard, "Combining feature selectors for text classification," in *Proceedings of the 15th ACM international conference on Information and knowledge management*, 2006, pp. 798–799.
- [13] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed. Prentice-Hall, 1998.
- [14] J. Shawe-Taylor and N. Cristianini, *Support Vector Machines*, 2nd ed. Cambridge University Press, 2000.
- [15] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse," in *Proceedings of the 29th International Conference on Software Engineering Workshops*. Washington, DC, USA: IEEE Computer Society, 2007, p. 76.

# CoMoVi: a Framework for Data Transformation in Credit Behavioral Scoring Applications Using Model Driven Architecture

Rosalvo Neto \*<sup>†</sup>, Paulo Jorge Adeodato\*, Ana Carolina Salgado\*  
\*Department of Computer Science  
Federal University of Pernambuco (UFPE)  
Recife-PE, Brazil  
{rfon2, pjla, acs}@cin.ufpe.br

Dailton Filho<sup>†</sup>, Genival Machado<sup>†</sup>  
<sup>†</sup>Department of Engineering Computer  
Federal University of Sao Francisco Valley (Univasf)  
Juazeiro-BA, Brazil  
{dailton, genival}@univasf.edu.br

**Abstract**—The stage of transforming data in knowledge discovery projects is costly, in general, it takes between 50 and 80% of total project time. This step is a complex task that demands from database designers a strong interaction with experts that have a broad knowledge of the application domain, making the task prone to error. The activities of that border region require a conjugation of database, statistics and system analysis competences. These competences are not ordinarily found in the same project team, whether in academia or in professional environment. The frameworks that aim to systemize this stage have significant limitations when applied to Credit Behavioral Scoring solutions. This paper proposes CoMoVi, a framework inspired in the Model Driven Architecture to systemize this stage in Credit Behavioral Scoring solutions. CoMoVi is composed by a meta-model which maps the domain concepts and a set of transformation rules. In order to validate the proposed framework, a comparative study of performance between frameworks found in literature and the proposed framework applied to a database of a known benchmark was performed. Student's one-tailed paired t-test showed that CoMoVi gives better performance to a Multilayer Perceptron Neural Network with a confidence level of 95%.

**Keywords**—*Meta-Modeling; Model Driven Architecture; Credit Behavioural Scoring; Knowledge Discovery*

## I. INTRODUCTION

The stage of transforming data is costly, generally consuming between 50 and 80% of the total project time [1]. In this stage data stored in a relational database are prepared and transformed for the application of data mining techniques. Although this stage consumes more than half of a Knowledge Discovery in Databases (KDD) project time, researches in the area are focused mainly on the principal stage of the process, data mining algorithms, and proportionally few works are found in literature concerning the phase of data transformation. There are two general objectives in transformation: data must be transformed in the format that allows the data mining algorithm to be applied, and also enable the analysis necessary to evaluate the results after applying the mining technique [2]. The specific objectives of this stage are: construction of features, features selection and aggregation.

Conventional techniques of data mining like decision tree, artificial neural networks and logistic regression, applied in

KDD process require as input a table containing a row for each object of interest, and a set of columns that describe the characteristics of these objects. The existing frameworks to systemize this stage follow the propositional approach or the multidimensional data mining. The propositional approach transform the multidimensional data representation inside a simple organized relation into a denormalized table in the granularity in which the decision is to be taken, which serves as input to conventional data mining algorithms. The multidimensional data mining approach [3] proposes that knowledge is extracted from each list separately, and later combined, instead of joining the various relationships. Frameworks existing in literature show significant limitations when applied to solutions of Credit Behavioral Scoring.

This work proposes CoMoVi (the name is acronym of Conceptual Modeling Visions), a new framework inspired by Model Driven Architecture (MDA) to systematize the data transformation stage, which takes into account all peculiarities of Credit Behavioral Scoring solutions, embedding automatic knowledge in data view by automatically generating new variables increasing the discriminatory power of the data mining technique. The framework is composed by a meta-model and a set of transformation rules. To validate the proposed framework efficiency a comparative study with the main existing frameworks and CoMoVi was carried out. This comparative study verifies which one provides more discriminatory power to the Multilayer Perceptron Neural Network when applied in solutions of Credit Behavioral Scoring. The comparative study uses an experimental methodology with rigorous statistical base applied to the database from a known benchmark of an international competition organized by PKDD 1999, for a binary classification problem, in order to perform the comparison.

The remainder of this paper is organized as follows. Section 2 presents the problem definition of Credit Behavioral Scoring in relational databases. Section 3 provides a brief presentation of MDA. Section 4 presents the related work to systematize the stage of data transformation. Section 5 details the proposed framework. Section 6 shows the experimental methodology. Section 7 presents the experimental results and their interpretation. Finally, Section 8 concludes this paper and proposes future works.

This work was supported in part by the FAPESB under grant 1047/2013 and by the CAPES under grant 25001019004P6

## II. PROBLEM DEFINITION

Credit Scoring and Behavioral Scoring are data mining solutions that help financial institutions to decide whether to grant credit to consumers based on the credit risk of their requests [4]. The goal of these solutions is to assign a “score” which identifies how closely the consumer is to one of two groups: “good” which will eventually meet its financial obligations, or a group of “bad”, whose application should be denied because of their high probability of failing in its commitments with the financial institution. Researches in this area have grown in recent years as a result of the recent financial crisis on a global scale.

Credit scoring is used when a new consumer makes a credit application. Only demographic information such as age, gender, income and other variables are taken into account in assigning the score. Behavioral Credit Scoring is used when a consumer who already has a history of transactions in the database of the institution is requesting credit [5]. In this case, in addition to demographic information, behavioral information is also taken into account, such as timely payment history, arrears, amount of loans, among others. The aim of the solution is to find in the database a profile that separates the good from the bad clients. The output of a solution of Credit Behavioral Scoring is interpreted as the probability of the customer to honor its debt with the institution, in order words, being a good customer.

In a recent study [6], the authors highlight the opportunities for Credit and Behavioral Scoring solutions and describe the processes involved. The first step of the process corresponds to the selection of a sample of clients, ensuring that data regarding their products and consumptions are available at a given point of observation. The period before the observation point is called the performance window. Data contained in the performance window are structured attributes that will be used as input for the solution of Behavioral Credit Scoring. Figure 1 illustrates how data are partitioned according to temporality.

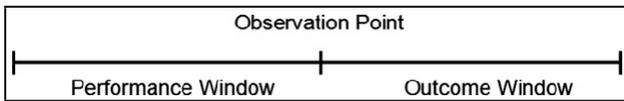


Fig. 1: Temporal Segmentation [6]

The period after the observation point is called the Outcomes Window. Data contained in Outcomes Window are structured attributes that will be used to assess the accuracy of the model, in this window the answering variable (“good” and “bad”) is constructed.

Behavioral Credit Scoring can be described as an instance of a relational classification problem in the domain of credit risk analysis. In a relational classification problem, the data available for the construction of a solution are in a database  $R$  containing a given target table,  $T_a$  and a set of background tables  $T_{b1} \dots T_{bn}$ . The background tables have relevant information to the decision problem, however they are not in target table. Each line belonging to  $T_a$  includes a single attribute called primary key (row identifier) and a categorical variable  $y$ , which represents the concept to be learned or “response variable”. The task of relational classification is finding a  $F(x)$  function which maps each  $x$  line of the target table for  $Y$

category. Figure 2 illustrates the problem of binary relational classification in the domain of credit risk analysis. The target table is represented by the Loan table on which the status column is the categorical variable that function  $F(x)$  has to learn. This variable has two values: good, if the loan was paid on time or bad, otherwise. The background tables are represented by tables that have a relationship with the target table, which is the case in the example of Figure 2 of tables instalment and client.

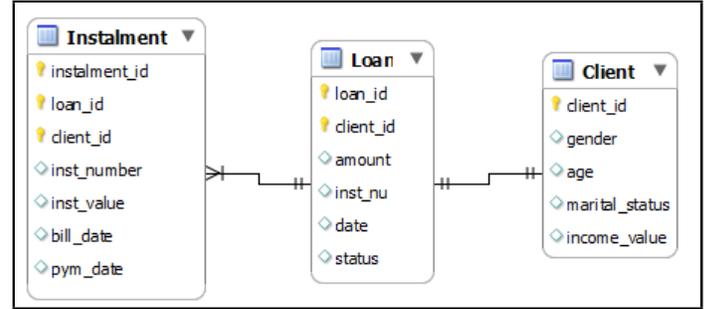


Fig. 2: Example of Relational Schema

## III. BASIC CONCEPTS IN MDA TECHNOLOGY

The major step in building an application is the conceptual modeling of the business domain. In this step the mapping of the real world to the model is done by specifying all details involved, including the relationships between entities and restrictions pertaining to the business. The Model Driven Engineering (MDE) is an approach especially focused on modeling techniques. The MDE proposes that conceptual models are used both for documentation as well as for software artifacts. One of the best known initiatives in this context is the Model-Driven Architecture (MDA) proposed by the Object Management Group (OMG) [7].

MDA is a way to develop software transforming an input model in an output artifact that may be another model or source code. These models can be Platform Independent Models (PIM) and Platform Specific Models (PSM). The transformation process is performed by a processing device following transformations rules. The rules of transformations specify how to generate a target model from a source model. To transform a given model to another model, the transformation rules map the source model using the target meta-model. The MDA provides the Meta Object Framework (MOF) for specifying meta-models and the Model Transformation Language (MTL) for specifying the rules that will be used to transform an input model into an output model. Output models are often source code.

## IV. RELATED WORKS

Research involving MDA and automatic code generation has been growing in recent years. Many tools and frameworks have been proposed and developed for different applications. However, according to our literature survey no proposal automates the data transformation stage in KDD projects. Among the closest researches it is possible to highlight: [8] and [9]. In [8], the authors proposed a framework based on MDA for mapping conceptual models of operational databases for Data Warehouses (DW). The framework consists in a meta-model

for specifying the conceptual model of the operational database and a set of rules for automatically generating the SQL script used in the construction of DW. In [9], the authors propose the construction of a software component based on MDA to systematize the analysis and visualization of academic information from management information systems. The authors' proposal is the semi-automatic construction of a DW in the field of higher education, thus facilitating the decision making of managers in the area. The component is based on three stages: multidimensional data modeling, data extraction and data visualization.

The two approaches found in literature applied in the stage of data transformation in a project of knowledge discovery are: propositionalization and multidimensional data mining. The approach of propositionalization transforms the multidimensional representation of data within a simple interface organized in a denormalized table in the granularity at which the decision is intended to be made. A distinct approach called multidimensional mining data [3] suggests that knowledge is extracted from each list separately, and later combined.

Relational Aggregations (RelAggs) [10] is the main framework for data transformation which follows the propositional line. In their approach the idea of aggregation, commonly used in the area of Data Warehouse is applied. Aggregation is an operation that replaces a set of values for a single value that summarizes the properties of these sets. For numerical values, simple descriptive statistics are used, such as maximum, medium and minimum value, for categorical values mode (most frequent value) can be used. RelAggs was adopted as a mechanism for transforming the Weka [1] platform, one of the main free tools for data mining. In [10] the performance of the main frameworks of transformation which follow the line of propositionalization was evaluated, and the authors concluded that RelAggs provides better performance when compared to the initial frameworks based in Inductive Logic Programming (ILP).

The Correlation-based Multiple View Validation (CbMVV) [11] is the main framework for data transformation which follows the multidimensional data mining approach. This approach is divided into three steps: firstly the relationships between tables are represented in the form of a graph, paths through this graph are combinations of visions between the target table and the background tables. In order to ensure the generation of non-cyclical paths, repeating paths are not allowed, and every path always starts from the target relation. The second step is to select the views which are relevant to the problem. For this, the authors propose an algorithm that calculates the relevance of visions by index calculation, taking into account the correlation between the attributes of vision, and also the correlation between attributes and the target concept "response variable". Views that have the lowest correlation with each other, and the highest correlation with the target, are selected. After selecting the relevant views, the algorithm enters the third and last stage in which a classifier is built for each view, and finally, a last classifier is constructed by using as input the responses from the individual classifiers of each vision. In [11], the authors demonstrated that this framework provides a higher predictive power for the data mining algorithm, when compared with ILP-based frameworks.

Based on frameworks found in literature, important charac-

teristics to the processing stage of data transformation in KDD projects applied in behavioral databases, like Credit Behavioral Scoring were identified:

- **Independence of mining technique (IMT):** This feature identifies whether the framework can be applied to any technique of data mining, since some frameworks mix the stages of data processing with the stage of data mining and, so, are specific to a certain technique.
- **Support the temporal segmentation (STS):** This feature identifies whether the framework addresses how to perform temporal segmentation during construction of behavioral variables, in other words, taking into consideration the partitioning in performance window and outcome window. This partitioning of data is essential to the success of the project using historical data as Credit Behavioral Scoring, since the use of available data on the performance window as input variables for the data mining technique makes all the project invalid.
- **Knowledge Acquisition (KA):** This feature identifies whether the framework addresses how to embed expert knowledge in the construction of variables during the stage of transforming the data to improve the discriminatory power of the data mining technique.

In Table 1, the "N" concept was attributed to frameworks that do not address the identified characteristic. And the "Y" concept for those addressing it in detail.

TABLE I: Comparison of Frameworks

Features	Frameworks		
	CbMVV	RelAggs	Based in ILP
IMT	Y	Y	N
STS	N	N	N
KA	N	N	N

## V. PROPOSED FRAMEWORK

Following the MDA approach this paper proposes CoMoVi, a framework for systematizing the stage of data transformation in KDD projects on the domain of Credit Behavioral Scoring. The framework takes into consideration the temporal peculiarities of the domain and embeds knowledge in the data vision building new variables to be used as input to a data mining technique. Figure 3 shows the architecture of CoMoVi.

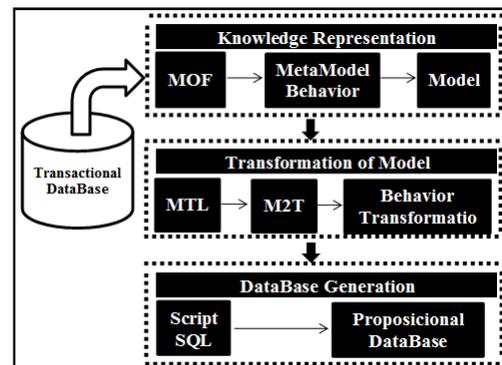


Fig. 3: Architecture of Proposed Framework

CoMoVi's architecture consists of three layers: in the first layer, called Knowledge Representation, the Behavior meta-model was defined using the MOF specification of MDA, and serves as the basis for the construction of specific models, which are adapted to the peculiarities of each database, however always following the concepts and rules defined by the meta-model. In the second layer, called the Model Transformation, the models generated by the Knowledge Representation layer are used as input to the module Transformation Behavior, which is responsible for automating the creation of SQL code for generating databases in propositional form. This module was written in MOF Model To Text (M2T) [12] which is an OMG approach to transform models into text artifacts. The third and last layer, called Database Generation, receives as input SQL codes generated by the Model Transformation layer and produces the database in propositional format that will be used as input to the data mining algorithm.

In order to embed expert knowledge and to support the temporal segmentation of data required by the Credit Behavioral Scoring the concept of behavioral Recency, Frequency and Monetary (RFM) analysis was introduced to CoMoVi meta-model [13]. The objective of the analysis is to distinguish clients based on three behavioral variables:

- **Recency (R):** Period of time since last purchase. It is the interval between the last transaction and present reference time. The lower this value is the more valuable the customer is;
- **Frequency (F):** Number of transactions in a given period until a present reference time. The higher this frequency is, the more valuable the customer is;
- **Monetary (M):** Total amount of money paid by the customer over a given period of time. The higher this value is, the more valuable the customer is.

In literature it is common to find studies using RFM variables as input for data mining techniques. In [14], the authors showed the importance of using RFM variables in building intelligent systems for e-commerce applications. The authors used the RFM variables as input to identify profiles of ecommerce users in a case study with one of the largest retail stores in Taiwan. In [15], the authors proposed a system of Customer Relationship Management (CRM) using RFM variables as input to clustering algorithms. The aim of the study was to identify niches with levels of customer loyalty to the institution.

The meta-model proposed in this paper was defined taking into account the peculiarities of the temporal segmentation and also creating new variables based on RFM analysis to embed expert knowledge. The meta-model proposed can be seen in Figure 4. The first element of the meta-model is Granularity, which represents the decision granularity of the project. This element has the "date" attribute, that represents the concept of Observation Point, used to divide the variables in a priori and a posteriori. The a priori variables represent the knowledge that happened before the Observation Point, so that they can be used as input for the data mining algorithm. The a posteriori variables represent the knowledge that happened after the Observation Point, so that they cannot be used as input, however they will be used as performance evaluation

variables such as, for instance, the response variable. The Entity element represents the background tables. This element is composed by a set of Fields elements which represent the characteristics of entities. The Relationship element is the relationship between the project granularity and another Entity. This type of relationship has one to one cardinality. The RelationshipTemp element represents the temporal relationship between the project granularity and another Entity. This type of relationship has cardinality one to many. The RelationshipTemp element has an attribute of type "date", which represents the date field of the Entity with greater cardinality in the relationship. This date field is mandatory, because it is from it that the temporal segmentation will be performed. The RelationshipTemp element has an attribute of type "fResume". This fResume attribute represents the field of the Entity from RelationshipTemp that will be used for building new RFM variables and descriptive statistics. The elements PerformanceWindow and OutcomeWindow represent the concepts of temporal segmentation defined in Section II. Each "Window" element has an attribute of type "Month". The Month attribute is an array that represents the interval of months for which RFM variables will be built, for example: if the instanced model from this meta-model uses a PerformanceWindows with two values for month, for example 6 and 12, variables will be built of type: frequency of transaction performed in the last 6 months and frequency of transaction performed in the last 12 months.

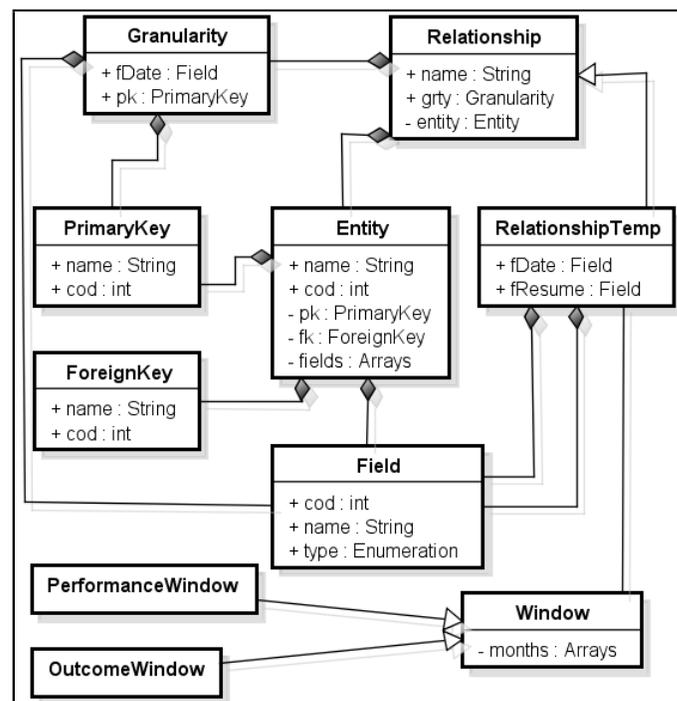


Fig. 4: Proposed Metamodel

After instantiating a model using the proposed meta-model, a set of model to text transformations is run in order to provide the SQL code that will generate the view to be used as input for a data mining algorithm. Three templates are executed. The first template RegisterData builds views with information of the backgrounds relations who have one to one relationship with the entity of granularity.

Listing 1: Code of template RegisterData

```
[template public RegisterData (rs: Relationship)]
CREATE VIEW [rs.name/] AS SELECT
[rs.grty.name/].[rs.grty.pk.name/]
[for(s: String | rs.entity.fields.name)] [rs.entity.
name/].[s/] [/for]
FROM [rs.entity.name/] , [rs.grty.name/]
WHERE [rs.grty.name/].[rs.grty.pk.name/] =
[rs.entity.name/].[rs.entity.pk.name/] [/template]
```

The second template, Behavior, constructs behavioral views from one to many relationships between background relationships and the granularity entity, the resulting View has two special attributes: APRIORI indicating whether the information may be used as input data for a data mining technique, and the DAYS field that tells how many days has that information from the Observation Point. This field will be used to calculate the Recency variable, as well as the partitioning of RFM variables in periods of months. The third and final template Windows constructs a set of new views containing the expert's knowledge by calculating RFM variables and descriptive statistics from the behavioral information generated by the template Behavior. The codes of templates are shown below.

Listing 2: Code of template Behavior

```
[template public Behavior (rst: RelationshipTemp)]
CREATE VIEW BEHAVIOR_[rst.name/] AS SELECT
[rst.grty.name/].[rst.grty.pk.name/],
([rst.fDate.name/] < [rst.grty.fDate.name/]) AS
APRIORI,
([rst.fDate.name/] - [rst.grty.fDate.name/]) AS
DAYS,
[for(s: String | rst.entity.fields.name)] [rst.
entity.name/].[s/] [/for]
FROM [rst.entity.name/],[rst.grty.name/]
WHERE [rst.grty.name/].[rst.grty.pk.name/] =
[rst.entity.name/].[rst.entity.fk.name/] [/template]
```

Listing 3: Code of template Windows

```
[template public Windows (pw: PerformanceWindows)]
[for(a: String | pw.months.first(2))]
CREATE VIEW RFM_[pw.rst.name/]_[a.trim()/] AS
SELECT [pw.rst.entity.fk.name/],
max(days) AS Recency_[a.trim()/],
count(*) AS Freq_[pw.rst.fResume.name/]_[a.trim()/],
sum([pw.rst.fResume.name/]) AS
Monetary_[a.trim()/],
max([pw.rst.fResume.name/]) AS
[pw.rst.fResume.name/]_max_[a.trim()/],
min([pw.rst.fResume.name/]) AS
[pw.rst.fResume.name/]_min_[a.trim()/],
avg([pw.rst.fResume.name/]) AS
[pw.rst.fResume.name/]_avg_[a.trim()/]
FROM BEHAVIOR_[pw.rst.name/]
WHERE APRIORI IS TRUE AND
(DAYS > 0 AND DAYS < [a.trim()/]*30 )
GROUP BY [pw.rst.entity.fk.name/] [/for] [/template]
```

## VI. EXPERIMENTAL METHODOLOGY

In order to validate the effectiveness of the proposed framework, an experimental study comparing the main proposed and existing frameworks in literature was performed. The RelAggs framework was chosen as representative of the propositional approach and the CbMVV framework as representative of the

multidimensional approach. The same data mining technique was applied to the databases generated by the frameworks to verify which data transformation framework provides greater discriminatory power for the data mining technique. The technique chosen was one of the most popular in the area of artificial intelligence and very used for Credit Behavioral Scoring solutions, the Artificial Neural Networks (ANN) Multi Layer Perceptron (MLP) [16]. The study was conducted over a public database of known benchmarks used in an international competition organized by PKDD [17]. The data describe the customers of a Czech bank with their bills, credit cards, loans, transactions on their accounts and aspects of the regions where customers and bank branches are located. Figure 5 shows the relational schema of the database.

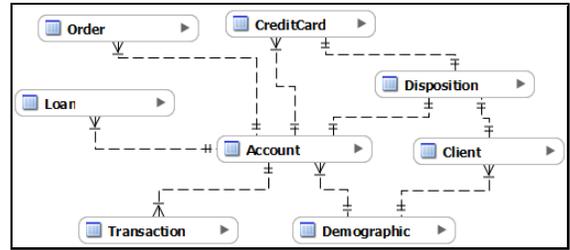


Fig. 5: Relational Schema of PKDD

The comparison was performed using the cross validation stratified k-fold framework ( $k = 10$ ), repeated 10 times to set the confidence intervals as recommended by the authors [1]. The performance evaluation metric used was the statistical maximum value of the Kolmogorov-Smirnov's curve (KS2) using MLP as a technique for data mining. The KS2 is a non-parametric statistical method used to measure the adhesion between functions of accumulated distributions [18]. In binary classification problems the KS2 curve is the difference between two cumulative distribution functions of each class having score as the independent variable. The one-tailed paired Student's t-test was applied to verify if there is statistically significant difference between the neural networks using the three frameworks. The test setup used in this study is detailed below.

- **Null Hypothesis:**  $\mu_d = \mu_1 - \mu_2$
- **Alternative Hypothesis:**  $\mu_1 > \mu_2$

where

- $\mu_1$  is the average maximum KS2 for a neural network using CoMoVi;
- $\mu_2$  is the average maximum KS2 for a neural network using an existing framework.

## VII. EXPERIMENTAL RESULTS

The simulations were performed according to the experimental setup described in Section VI for each one of the three frameworks, resulting in 10 testing sets, all statistically independent. CoMoVi provided greater predictive power for the neural network in 8 of the 10 sets of tests as shown in Figure 6, which shows the results obtained in the experiment. Table 2 shows the summary of results obtained in the one-tailed paired t-test. Since p-value is less than 0.05, we conclude

that all three frameworks provide different processing results. Specifically, data indicate that CoMoVi produces, on average, higher discriminatory power for the MLP network than RelAggs and CbMVV frameworks with a confidence level of 95%.

The results show that frameworks following the propositional approach (RelAggs and CoMoVi) outperform the CbMVV framework, which follows the approach of multidimensional data mining, in performance. The performance difference can be justified by appropriate choices of the metric for evaluating the performance and the artificial intelligence technique used in the stage of data mining, which in this work are more suitable for the domain of credit risk analysis. However, the most plausible explanation is the reduced functional capacity of the data mining algorithm caused by the input space sampling inherent to the approach of multidimensional data mining. This approach creates many local solutions with partial views of the problem, while the propositional approach builds a view with all variables using the whole functional capacity of the neural network, that is an universal approximator of functions.

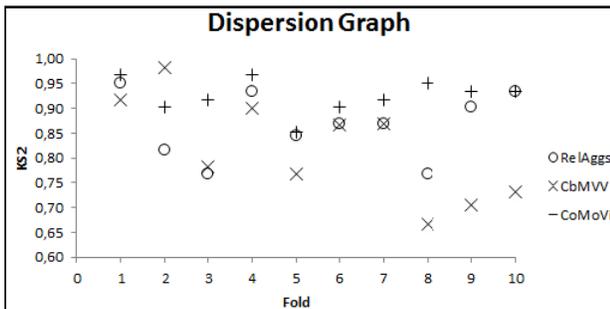


Fig. 6: Dispersion Graph

TABLE II: Summary of Results

$\mu_1$	$\mu_2$	Lower Limit	$\mu_d$	Upper Limit	p-value
CoMoVi	RelAggs	0,0169	0,0554	$\infty$	0,0133
CoMoVi	CbMVV	0,0427	0,1054	$\infty$	0,0065

## VIII. CONCLUSION

This paper presented a new framework inspired by MDA to systematize the stage of data transformation in KDD projects in the domain of Credit Behavioral Scoring. The framework is composed by a meta-model that maps key concepts of the domain and a set of transformation rules, which generate SQL code from models instantiated by the proposed meta-model. In comparison with the main existing frameworks, the experimental study showed that CoMoVi produces better performance to the technique of ANN when applied to a benchmark. The difference in performance can be explained by the construction of new variables generated by CoMoVi, based on RFM analysis within slide windows, which embeds new knowledge for the technique of data mining in the form of input variables.

Among the main contributions of the proposed framework the highlights are: 1) providing greater discriminatory power for the technique of data mining to build new variables based on RFM analysis, 2) minimizing errors in the calculation

of behavioral variables by automating temporal segmentation within the meta-model; 3) easing the use by using models for specifying data views, 4) Platform independence and technical data mining. As a future work, this study will be expanded to check the power of CoMoVi generalization in databases of different domains of credit risk analysis.

## REFERENCES

- [1] I. H. Witten, E. Frank, and M. A. Hall, *Data Mining: Practical Machine Learning Tools and Techniques*, 3rd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011.
- [2] S. Chakrabarti, E. Cox, E. Frank, R. H. Gting, J. Han, X. Jiang, M. Kamber, S. S. Lightstone, T. P. Nadeau, R. E. Neapolitan, D. Pyle, M. Refaat, M. Schneider, T. J. Teorey, and I. H. Witten, *Data Mining: Know It All*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2008.
- [3] L. Cao, H. Zhang, Y. Zhao, D. Luo, and C. Zhang, "Combined mining: Discovering informative knowledge in complex data," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 41, no. 3, pp. 699–712, June 2011.
- [4] N.-C. Hsieh, "Hybrid mining approach in the design of credit scoring models," *Expert Syst. Appl.*, vol. 28, no. 4, pp. 655–665, 2005.
- [5] N. Sarlija, M. Bencic, and M. Zekic-Susac, "Comparison procedure of predicting the time to default in behavioural scoring," *Expert Syst. Appl.*, vol. 36, no. 5, pp. 8778–8788, 2009.
- [6] K. Kennedy, B. M. Namee, S. Delany, M. O. Sullivan, and N. Watson, "A window of opportunity: Assessing behavioural scoring," *Expert Systems with Applications*, vol. 40, no. 4, pp. 1372–1380, 2013.
- [7] O. M. G. (OMG), "Catalog of omg modeling and metadata specifications," Tech. Rep., 2008. [Online]. Available: [http://www.omg.org/technology/documents/modeling\\_spec\\_catalog.htm](http://www.omg.org/technology/documents/modeling_spec_catalog.htm)
- [8] L. Zepeda, E. Cecena, R. Quintero, R. Zatarain, L. Vega, Z. Mora, and G. G. Clemente, "A mda tool for data warehouse," in *Proceedings of the 2010 International Conference on Computational Science and Its Applications*, ser. ICCSA '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 261–265.
- [9] J. Xie, X. Li, L. Wang, and Y. Niu, "A mda-based campus data analysis and visualization framework," in *Proceedings of the 2011 Third International Workshop on Education Technology and Computer Science - Volume 02*, ser. ETCS '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 118–121.
- [10] M.-A. Krogel and S. Wrobel, "Facets of aggregation approaches to propositionalization," in *Work-in-Progress Track at the Thirteenth International Conference on Inductive Logic Programming (ILP)*, T. Horvath and A. Yamamoto, Eds., 2003.
- [11] H. Guo and H. Viktor, "Multirelational classification: a multiple view approach," *Knowledge and Information Systems*, vol. 17, no. 3, pp. 287–312, 2008.
- [12] L. Rose, N. Matragkas, D. Kolovos, and R. Paige, "A feature model for model-to-text transformation languages," in *Proceedings of the 2012 Modeling in Software Engineering*, ser. MiSE '12. Zurich, Switzerland: IEEE Computer Society, 2012, pp. 57–63.
- [13] M. Y. Lee, A. S. Lee, and S. Y. Sohn, "Behavior scoring model for coalition loyalty programs by using summary variables of transaction data," *Expert Syst. Appl.*, vol. 40, no. 5, pp. 1564–1570, 2013.
- [14] Y.-L. Chen, M.-H. Kuo, S.-Y. Wu, and K. Tang, "Discovering recency, frequency, and monetary (rfm) sequential patterns from customers purchasing data," *Electronic Commerce Research and Applications*, vol. 8, no. 5, pp. 241–251, 2009.
- [15] C.-H. Cheng and Y.-S. Chen, "Classifying the segmentation of customer value via {RFM} model and {RS} theory," *Expert Systems with Applications*, vol. 36, no. 3, Part 1, pp. 4176–4184, 2009.
- [16] S. Haykin, *Neural Networks and Learning Machines*, 3rd ed. Upper Saddle River, NJ: Prentice-Hall, 2009.
- [17] P. Berka, "Guide to the financial data set," in *PKDD 2000 Discovery Challenge*, 2000, pp. 87–92.
- [18] W. Conover, *Practical nonparametric statistics*, 3rd ed. New York: Wiley, 1999.

# Tracing Domain Data Concepts in Layered Applications

Mohammed Daubal, Nathan Duncan, Delmar B. Davis, Hazeline U. Asuncion

Computing and Software Systems  
University of Washington, Bothell  
Bothell, WA, USA

{daubalm, njd91, davisdb1, hazeline}@u.washington.edu

**Abstract**—A goal of reusing source code is to lower development costs; however, reuse techniques usually require additional costs in creating generic source code and retrieving the appropriate source code from a code repository. In this paper, we present Domain Data Concept (DDC) Tracer, a novel traceability technique that facilitates reuse in layered applications in a cost-effective way. While current traceability techniques focus on establishing links between software artifacts, DDC Tracer is focused on tracing concepts between software and data, across different layers of an application, and across heterogeneous implementation files. We present a feature comparison which highlights the uniqueness of our approach. In addition, we conducted an industry case study where we analyzed software artifacts and solicited feedback from software engineers. Our case study indicates that our approach is a lightweight alternative to source code reuse techniques and is feasible to use in practice.

**Keywords**—Software traceability, reuse, structured query language, layered architecture style

## I. INTRODUCTION

Literature has shown that code reuse lowers costs by lowering the amount of software that needs to be developed [25]. The ability to reuse code remains challenging in practice because a key criteria for success is its economic benefit to the organization (i.e., whether there is a good return on investment on the upfront costs for reuse) [25]. Current techniques for reuse include: creating a reusable library of components, using domain engineering (e.g., model-driven development, product line engineering), or using software architecture [25]. Code search tools have also been used to facilitate code reuse by enabling developers to search by textual similarity, program structure, or code semantics [13, 43]. These techniques do not support reuse for applications that access and manipulate data itself. Moreover, while identifying related source code has traditionally fallen within the purview of software dependency analysis [27], contemporary development of layered applications often use third-party technologies that combine source code with configuration files and scripts, resulting in a heterogeneous mix of implementation files (e.g., [4, 6, 24]). Thus, traditional static and dependency analysis tools are unable to trace through these implementation files.

This paper aims to address this gap in supporting code reuse in data-centric applications. More specifically, we support reuse for software applications that connect to databases by tracing concepts from the database layer to the

user interface layer. Conceptually, these domain data concepts (DDCs) map to elements in the schema, such as table and column names. Our technique, Domain Data Concept Tracer (or DDC Tracer), connects data concepts with implementation files in a layered architecture to create a searchable traceability link chain. By centering traceability links on DDCs and leveraging the knowledge of how DDCs are related to each other (via a database schema) and how source code is structured (e.g., communications within a framework, caller graph, etc.), we can create a traceability link chain from DDCs to implementation files, regardless of the textual similarity of DDCs to source code, and regardless of the implementation file (e.g., configuration files, structured query language (SQL) statements). This approach also enables software developers to search for highly relevant implementation files for reuse.

The contributions of this paper include: (1) a data-centric approach to tracing DDCs in a layered architecture, (2) automatic mapper generators, and (3) evaluations based on an industrial case study and feature comparison.

The paper is organized as follows. The next section details the motivation behind our technique. Section 3 covers related work and background information. Section 4 presents our technique, DDC Tracer, and Section 5 presents our tool support. Our evaluation methods are discussed in Section 6. We conclude the paper with future work.

## II. MOTIVATION

In large corporations, software development is often distributed among different teams and explicit collaboration among these independent teams is usually difficult [22]. Such is the case with Saudi Aramco, a Saudi Arabian national oil and gas company [7]. The company is one of the largest oil companies in the world, managing proven conventional reserves of 260.2 billion barrels of oil and gas reserves of 284.8 trillion cubic feet and has the largest daily oil production.

Because development is distributed along different business units, explicit collaboration among these independent teams is usually difficult in a large company setting like Aramco [22]. Teams are not aware of source code they can reuse from other teams, resulting in redevelopment of software from scratch with similar functions. Developers try to reuse code by manually searching for existing code that was developed by that developer, a developer's colleague, or by the developer's team. This time consuming task does not always result in a

successful retrieval of reusable code, since the search is limited to the developer's memory or communication channels with other developers. In addition, a text search or a dependency analysis does not yield all the files that operate on a specified DDC because of the various technologies used. Thus, current techniques are inadequate to determine the connections between related files in an implemented system.

Additionally, software development projects in the context of large corporations operate on data that is often stored in back-end databases [31]. Usually, a common thread among these independent projects is its access to a centralized database, which contains information that is a core business asset to the company. (For example, these databases may contain data about a company's customers.)

### III. RELATED WORK AND BACKGROUND

#### A. Software Traceability

Software traceability has traditionally focused on requirements traceability, with the main goal of demonstrating that a delivered system meets the requirements [30]. Other approaches have also suggested centering links on the code [21] or on the architecture of the software [12]. Our approach centers traceability links on DDCs to connect together heterogeneous files that manipulate specified DDCs.

There are varying levels of tool support for capturing traceability links. One class of techniques, information retrieval (IR) techniques [17], uses textual similarities between source code and various documentations to automatically recover traceability links. These techniques fail to recover traceability links that cross layer or language boundaries.

The challenges of tracing across one type of software artifact (e.g., implementation files), referred to as vertical traceability, have also been examined in industry [35]. Our technique, meanwhile, focuses on the challenges encountered in tracing DDCs in a contemporary layered application.

#### B. Software Reuse

There are techniques for reuse, including using reusable libraries and reusable components [25, 41], but an existing challenge is the upfront costs for reuse before any cost savings can be realized [25]. Recent reuse techniques aim to address this challenge, such as using variant analysis between different source code to determine the commonality and variability [20], using a flexible code generator that allows developers to weave in their manual modifications [34], and connecting use cases to their implementations [45]. Our approach enables developers to find reusable implementation files based on specified DDCs.

#### C. Concept or Feature Location

Concept or feature location is a closely-related area of research that aims to find human concepts within a software implementation [26, 36, 40]. Concepts may take on different definitions, depending on the context of its usage [26]. For example, concepts may be related to software change [26], functions [9, 36, 40], or domain concepts [11]. These techniques generally use static and dynamic analyses [23, 42], IR techniques [36], or IR techniques with formal concept analysis [37, 40] or user feedback [26]. A challenge with these

techniques is determining the appropriate search term to obtain the desired implementation files. Our work, meanwhile, focuses on DDCs which are at a lower level of abstraction than features. This data focus allows us to connect concepts that are familiar to domain practitioners with the implementation files, enabling us to traverse the problem space to the solution space, through the use of concrete terms.

#### D. Recommender Systems

Recommender systems for software engineering (RSSEs) have been developed to assist developers locate relevant examples, guide software changes, or find reuse opportunities [32, 44]. RSSEs, based on recommender systems for online shopping applications, also have the challenge of determining the context of the query [44]. Our focus is on tracing DDCs across the various layers of a layered application, and not on determining the context of a query to provide source code recommendations. QueRIE is an SQL recommender system that aims to assist non-expert users of scientific databases to recommend queries that may be useful to a particular user [38]. Our technique is similar to QueRIE, but extends the connection beyond SQL queries to other implementation files.

#### E. Code Search and Code Clone Tools

Code search and code clone tools are often used to aid developers find code examples, find code that can be reused, or find similar code [13, 14, 43]. Code search tools use various techniques, such as text analysis [2], regular expression [3], code structure with vector space model [13], IR techniques with program analysis [29], or static and dynamic specifications with program analysis [43]. Some tools also include support for searching through SQL statements [5]. Meanwhile, code clone tools assist developers in locating similar source code [14]. Our approach supports searching for implementation files found in a traceability link chain and searching for a similar code based on similarly accessed DDCs.

#### F. Analysis of Data Intensive Applications

There are also techniques that leverage information found in SQL statements or data access code. These include database schema recovery using SQLs [15], identification of features using data access code [18], and static analysis of both SQL and source code to uncover SQL errors to avoid runtime errors [28]. Our tool, meanwhile, focuses on establishing traceability to support reuse.

#### G. Background on Databases and SQL

A database is a collection of interrelated data which represents concepts from the real world [19]. The definitions of these concepts are provided by a data model and the most popularly used data models are relational, object-oriented, and object-relational. The relationship between real-world concepts are represented in a schema [24].

In a relational database, a data concept is stored in a table (e.g., Employee table) and the attributes of this concept are stored as columns in the table (e.g., EmployeeName, StartDate, EmployeeClassification) [16]. Related DDCs are specified through table relationships. One way this is achieved is through the use of primary and foreign keys. A primary key is used to uniquely identify each row in a table. In our example,

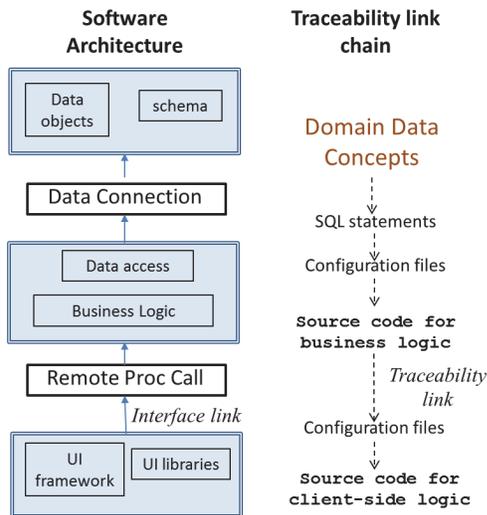


Figure 1. Client-Server application (left diagram) and DDC traceability link chain across the different layers (right diagram)

EmployeeID may be used to uniquely identify each tuple or row in the Employee table. A foreign key is used to refer to a primary key in another table. For example, employees may work in any department. This relationship between the Department concept and the Employee concept may be expressed through the use of DepartmentID as a foreign key within the Employee table.

#### IV. DOMAIN DATA CONCEPT TRACER

This section discusses key steps in our approach while the next section discusses how these steps can be automated. Before we discuss our technique, we first provide a general overview of software systems that can use our approach.

Software that follows the layered architectural style has source code organized in different layers [46]. Each layer uses the services provided by an adjacent layer. A client-server style is a special case of the layered style where the number of layers is either two or three. We use a three-tiered client-server application using current technologies (see Figure 1) as an exemplar. A client-server style is popularly used in business applications, where the logic and data model are centralized [46]. As shown in the figure, the system consists of a heterogeneous mix of technologies: data manipulation or extraction (e.g., query statements), business logic (e.g., Java, C++), and client-side logic (e.g., Javascript, HTML). On top of these, companies may use frameworks (e.g., [4]) to lower development and maintenance costs and to standardize the code base.

##### A. Determine the Starting Point for Tracing

A starting point for tracing may be a data model (e.g., schema) or data access code. A data model may be a starting point if developers are provided with a set of DDCs and they wish to reuse existing code that access this set of DDCs. With this starting point, some DDCs may not be used within a project and, thus, have no traceability links to the code.

An alternative is to start from a given project and extract the data access code from there. This option is more feasible in

cases where developers know they wish to reuse code from a given software project or in cases where a data model is not accessible. Note that DDC extraction must occur in the data access code (e.g., where database access occurs or where file access is performed) because data used in non-data access code may not semantically represent domain concepts, even if the same terms are used.

##### B. Create Mappers at the Layer Boundaries

Layer boundary mappers are crucial to bridge the syntactic gap that exists between layers of implementation files. We now discuss how mappers can be created between data and software (e.g., data and business logic layers) and between software layers (e.g., business logic and user interface layers).

###### 1) Data to business logic layer boundary:

Once we have a list of DDCs we wish to trace, we can create a mapper between data and software boundary by performing DDC name tracing at the data access code. Since the data access code is the first layer where data is either extracted or manipulated (often through the use of SQL statements), this guarantees that a data concept embedded within the data access code is semantically the same as the data concept in a data source (e.g., database schema). Thus, a mapping can be created between each DDC and all data access code that contains the DDC.

It is also possible that configuration files are used to store data access code instead of embedding them within a programming language (e.g., [4]). In this case, DDC name tracing can still be used between DDCs and configuration files. Then, mapping between the configuration file and a programming language can be created by resolving the data access identifiers (e.g., SQL IDs) in the configuration file and the programming language. Thus, a mapper can be created from the DDCs to configuration files via DDC name tracing and from configuration files to the programming language via data access identifiers.

###### 2) Business logic to user interface layer boundary:

The programming language used for business logic, which resides on a server, is often different from the programming language used for user interfaces, which resides on a client machine. It is also often the case that user interfaces are not necessarily implemented with a programming language, but with scripts (e.g., Javascript) or frameworks (e.g., [33]). To cross these language boundaries, one can create a mapper using the following steps. First, create a list of all the exposed services (i.e., provided interfaces) on the business logic layer that are related to at least one of the DDCs selected in the previous section. Then, find client-side implementation files that directly call these exposed services in the business layer. Among these client-side implementation files, one can perform a service name tracing for each of the exposed services. Thus, a mapping from the exposed services (on the business logic) to the user interface files that call those services can be created using service name mapping.

##### C. Create Mappers within Layer Boundaries

###### 1) Within business layer:

Generally, the business logic is implemented using one programming language. In such cases, creating a mapping

between implementation files is straightforward with a static analysis tool. One can start with data access code that is related to one of the DDCs. From this data access code, a static analysis can produce a call graph at the file-method level until it reaches the files with exposed services at the business layer. This call graph can act as a mapper among source files in the business layer.

There are cases where the business logic itself is implemented with multiple programming languages. In this case, one must identify which files reside at the language boundary, henceforth referred to as the language boundary file (LBF). The caller LBFs can be identified with reserved words that call on external files. For example, in Java, `ProcessBuilder` or `Runtime` [39] allows Java files to call executable files, which are created using other languages (e.g., C++, Visual Basic, C#). Java Native Interface (JNI) may also be used to call source code in another language [39]. After LBFs are identified for one programming language, one can locate the calls to the other programming language. A mapper can then be created between caller LBFs in one language and callee LBFs in another language using class-method name tracing.

#### 2) *Within user interface layer*

Within the user interface layer, if only one language is used, then it is possible to perform a static analysis and create mappers from call graphs as previously discussed. User interfaces are often implemented with a heterogeneous mix of files (e.g., stylesheets, scripts, HTML). In addition, user interface frameworks may be used to handle events (e.g., button clicks) or to dispatch events. If an event framework is used, traces between user interface files can be created based on the event name. In this case, a mapper among user interface files can be created by using event-name tracing.

#### D. *Connect Mappers to Create a Traceability Link Chain*

Once the mappers are created between boundaries (e.g., layer boundary, programming language boundary) and within boundaries, a complete traceability link chain can be composed from DDCs to the user interface layer. In addition, if a data model exists (e.g., schema), connections among DDCs can be obtained from the data model. This can be accomplished by searching for specified DDCs in the schema and examining how the DDCs may be related (e.g., usage of foreign and primary keys). These connections can provide additional indirect traceability links between source code and DDCs.

Because the mappers can be automatically generated, the traceability link chain can be regenerated any time a change occurs within a software project. As we will show later in the paper, our approach also yields reusable implementation files.

#### E. *Provide Search Capability*

Once we have a traceability link chain, we can search for any files along the chain. For example, we can search for DDCs and obtain related implementation files or search for source code and obtain related DDCs. We can display search results by complete traceability link chains, to aid reuse, or by specific type of implementation file (e.g., user interface code or business logic code).

## V. TOOL SUPPORT

Most of the features of the DDC Tracer have been implemented: DDC extractors, boundary mappers, call graph mappers, and search tool. The DDC extractor from data access code requires a parser. For this, we used the ZQL parser [8] to parse SQL statements. To support various types of name tracing, we used the Lucene search library [10]. To implement boundary mappers, we used Python scripts and Java to pre-process configuration and source code files. To implement our call graph mappers, we used `DependencyFinder` [1]. We used MySQL to store the traceability links between DDCs, SQL statements, and various implementation files. To increase the accessibility of the tool to software engineers within a company, we developed a web user interface.

Some of the usage scenarios we envision include the following. Junior and/or new developers to the project can find examples on how to access and manipulate the various database concepts. In this scenario, it would be useful for developers to have a list of reusable code ranked by complexity, where the easiest to reuse code is listed first. Developers who wish to refactor similar code can search for similar code based on their relationships with DDCs. When the database model changes (e.g., database concepts are added or deleted), maintenance engineers can also search for software that operate on related concepts or specific database concepts and modify those pieces of code. Finally, technical leads and project managers can use the tool to find DDCs that are most frequently accessed by the software to optimize the database.

## VI. EVALUATION

In this section, we provide a case study on industry projects, feedback from industry software engineers, and a feature comparison.

#### A. *Case Study on Aramco Software Projects*

In our case study, we selected software projects that are implemented in various technologies, such as `iBatis/myBatis` mapper framework [4], Java Remote Method Invocation [39], Spring Framework [6], and Adobe Flex [33]. While these technologies use Java as a programming language, these technologies also use configuration files and SQL statements as part of its implementation. The baselines are selected because they are an appropriate alternative for the given tasks. As previously mentioned, traceability techniques do not currently support connecting DDCs and implementation files. Mapping DDCs to features is also not straightforward, making it difficult to use a feature location technique as a baseline. We performed the following tasks:

##### 1) *Given a set of DDCs, find the relevant Java files*

In this task, we compared the results of our tool with a proprietary tool called “Find it EZ,” which performs string and regular expression matching on a directory of files specified by a user [3]. We ran single DDC queries five times and multiple DDC queries five times. We ran the query on two projects, which consist of 763 SQL statements, 687 configuration files, 984 Java files. With regards to accuracy, both DDC Tracer and Find it EZ retrieved correct SQL statements that contain the DDCs that were queried. Find it EZ was not able to find

TABLE I. FEATURE COMPARISON

	DDC Tracer (our tool)	Code Search [3, 5, 27]	Database Search [17]	Recommender Systems [30, 42]	Dependency Analysis [11,26]
Trace DDC to SQL statements	Yes	Yes	Only search within a database	Possible	Possible with [26]
Trace DDC to source code	Yes	Possible	No	Possible	Possible, if DDC is traced to a method or a file
Trace DDC to various configuration files	Yes	Possible	No	Possible	No
Create traceability link chain from DDC to UI	Yes	No	No	No	Traceability link chain within programming language only

related source code because they do not contain the exact DDC text. This limitation is similar to other text search tools.

### 2) Given a Java source code, return the relevant DDCs

In this task, our baseline is a similarity metric based on similar words between Java source code and a SQL statement. We created a word-document matrix where each Java source code file is associated with DDCs that are present within the file. We then compared the results of our baseline with our tool. We analyzed 32 business logic Java source code from three software projects, with each file averaging 209 lines of code. Among these files, the baseline technique was not able to detect related DDCs for 19 files, but our technique was able to return at least one correct DDC for each of the 32 files.

### 3) Find closely related Java code

In this task, our baseline is a similarity metric based on similar words between Java source code (with stop words removed). DDC Tracer, meanwhile, identified closely related Java code based on the number of similar DDCs that the files accessed. In this task, our technique is able to detect more semantically-related code than textual similarity techniques. For example, our baseline detected that there is a close similarity between File3 and File2, whereas DDC Tracer detected a close relationship between File3 and File9.

File3 has the following functionalities: getERigs, getERigsByDates, getDrillActivities, getEActivitiesByDates.

File2 has the following functionalities: getReportByDates, getWellList, getReportList, getTasksList, getTaskByPK, updateAllTasks, updateTasks, deleteTasks, getTaskTrackingList, updateAll, addWell, getAPLList, getUAPList, sendMail.

File9 has the following functionalities: getEWellSiteRigs, getWellSiteActivities.

Based on these method names, we see that File3 is indeed semantically closer to File9 than File2.

### B. Feedback from Aramco Software Engineers

We also solicited the feedback of eight system analysts from the company. The range of experience of these engineers is from 2 months to 15 years and six out of the eight analysts stated that they performed SQL and code search many times in the past. We presented the technique to the engineers and spent 10-15 minutes to demonstrate the tool to each of them. We then asked their feedback regarding our research questions:

Q1: Does DDC Tracer facilitate finding SQL statements and source code that could be reused for another project?

Q2: How does DDC tracer compare to the current process of searching for SQL statements and source code to reuse?

Q3: Are software engineers willing to use DDC Tracer?

### 1) Results

Q1: Seven out of the eight software engineers stated that the DDC tool can help them find SQL statements and source code that they can reuse for their project.

Q2: Five out of the eight software engineers state that using the DDC Tracer is much easier than their current process for searching for reusable code. Two of these engineers pointed out that compared to their current process of searching for reusable SQL statements and source code, the tool can provide them significant time savings. Another engineer prefers to use the tool over his current process because of its capability—the ability to search for multiple key words and the ranking of search results. In addition to the five engineers, one engineer stated that the tool will enable him to find all the possible solutions that he can use for any of his projects.

Q3: Seven out of the eight software engineers say that they are willing to use the DDC Tracer for their future software projects. On a scale of 1-5, where 1 is most useful and 5 is not useful at all, three of the five engineers rated the tool as 1, three engineers rated the tool as 2, and one engineer rated the tool as 3 (somewhat useful). The rating of 3 came from an engineer who has not searched SQL statements or source code before.

### 2) Discussion

A majority of the subjects who participated in the study provided positive feedback regarding our tool, especially those who regularly search for SQL statements or source code. Some of the subjects provided suggestions for improvement, such as integrating the tool with software repositories. One of the subjects who stated that ranking the search results was most useful would like to be able to customize the ranking feature.

A possible threat to external validity may be present in our study. While the feedback was based on responses from software engineers who chose to participate in the study and may be specific to their particular application domain, these preliminary results indicate that using DDC to trace related heterogeneous implementation files is a viable reuse technique in industry where layered applications are developed.

### C. Feature Comparison

Table 1 shows a feature comparison between our approach and existing techniques in the literature. The “possible” notation in the table indicates additional steps or information may be required by the technique to support the feature.

## VII. CONCLUSION

In this paper, we presented DDC Tracer, a data-centric traceability approach that facilitates code reuse in a fairly straightforward manner. By focusing on DDCs, we can find similar data concepts at the database layer, identify code that accesses and renders specified data concepts, and trace across the various layers of software regardless of the programming language or technology used. We evaluated our approach using an industrial case study that includes feedback from software engineers. We also compared DDC Tracer with existing techniques in the literature. The results of these evaluations indicate that DDC Tracer is effective in facilitating reuse and is feasible to use in practice.

Future avenues of work include analyzing the SQL statement structure, ranking the relevant traceability link chains, and creating visualization tools. We also plan to conduct further user evaluations to improve our technique.

## ACKNOWLEDGMENT

We thank the Aramco software engineers for feedback. We are also grateful to Hussein Al-Helal's suggestions and support.

## REFERENCES

- [1] Dependency finder. <http://depfind.sourceforge.net>.
- [2] Elasticsearch. <http://www.elasticsearch.org>.
- [3] Find it EZ Software. <http://www.finditez.com/>.
- [4] MyBatis: SQL mapping framework for java. <http://code.google.com/p/mybatis/>.
- [5] Ohloh code. <http://code.ohloh.net/>.
- [6] Spring framework. <http://www.springsource.org/spring-framework>.
- [7] Wikipedia: Saudi Aramco. [http://en.wikipedia.org/wiki/Saudi\\_Aramco](http://en.wikipedia.org/wiki/Saudi_Aramco).
- [8] ZQL. <http://sourceforge.net/projects/zql/?source=directory>.
- [9] G. Antoniol, E. Merlo, Y. Guéhéneuc, and H. Sahraoui. On feature traceability in object oriented programs. In *Proc of 3rd Int'l Workshop on Traceability in Emerging Forms of Software Engineering*, 2005.
- [10] Apache Software Foundation. Lucene. <http://lucene.apache.org>, 2013.
- [11] A. Aryani, F. Perin, M. Lungu, A.N. Mahmood, and O. Nierstrasz. Can we predict dependencies using domain information? In *Proc of Working Conference on Reverse Engineering (WCRE)*, 2011.
- [12] H. U. Asuncion and R. N. Taylor. *Software and Systems Traceability*, chapter Automated Techniques for Capturing Custom Traceability Links across Heterogeneous Artifacts. Springer-Verlag, 2012.
- [13] S. K. Bajracharya, J. O., and C. V. Lopes. Sourcerer - an infrastructure for large-scale collection and analysis of open-source code. *Science of Computer Programming*, 2012.
- [14] H.A. Basit and S. Jarzabek. A data mining approach for detecting higher-level clones in software. *Transactions on Software Engineering (TSE)*, 35(4):497–514, July 2009.
- [15] M.R. Blaha. Dimensions of database reverse engineering. In *Proceedings of Working Conf on Reverse Engineering*, 1997.
- [16] P. Chen. The entity-relationship model: toward a unified view of data. *SIGIR Forum*, 10(3):9–9, December 1975.
- [17] A. De Lucia, M. Di Penta, and R. Oliveto. Improving source code lexicon via traceability and information retrieval. *TSE*, 37(2):205–227, 2011.
- [18] C. Del Grosso, M. Di Penta, and I.G.R. de Guzman. An approach for mining services in database oriented applications. In *Proc of European Conf on Software Maintenance and Reengineering*, 2007.
- [19] K. R. Dittrich, D. Tombros, and A. Geppert. Databases in software engineering: a roadmap. In *Procs Conf on Future of Software Eng*, 2000.
- [20] S. Duszynski, J. Knodel, and M. Becker. Analyzing the source code of multiple software variants for reuse potential. In *WCRE*, 2011.
- [21] A. Egyed, S. Biffl, M. Heindl, and P. Grünbacher. Determining the cost-quality trade-off for automated software traceability. In *Proc of 20th Int'l Conf on Automated Software Engineering*, 2005.
- [22] K. Ehrlich and M. Cataldo. All-for-one and one-for-all?: a multi-level analysis of communication patterns and individual performance in geographically distributed software development. In *Proc of Conf on Computer Supported Cooperative Work*, 2012.
- [23] T. Eisenbarth, R. Koschke, and D. Simon. Locating features in source code. In *TSE*, 2003.
- [24] A. Eisenberg and J. Melton. SQL: 1999, formerly known as SQL3. *SIGMOD Record*, 28(1):131–138, March 1999.
- [25] W.B. Frakes and K. Kang. Software reuse research: status and future. *TSE*, 31(7):529–536, 2005.
- [26] G. Gay, S. Haiduc, A. Marcus, and T. Menzies. On the use of relevance feedback in IR-based concept location. In *Proc of Int'l Conf on Software Maintenance*, 2009.
- [27] M. Gethers, A. Aryani, and D. Poshyvanyk. Combining conceptual and domain-based couplings to detect database and code dependencies. In *Int'l Working Conf on Source Code Analysis and Manipulation*, 2012.
- [28] C. Gould, Z. Su, and P. Devanbu. Static checking of dynamically generated queries in database applications. In *Proc of Int'l Conf on Software Engineering (ICSE)*, 2004.
- [29] M. Grechanik, C. Fu, Q. Xie, C. McMillan, D. Poshyvanyk, and C. Cumby. A search engine for finding highly relevant applications. In *ICSE*, 2010.
- [30] J. H. Hayes, A. Dekhtyar, and S. K. Sundaram. Advancing candidate link generation for requirements tracing: The study of methods. *TSE*, 32(1):4–19, 2006.
- [31] G. Henriques, L. Lamanna, D. Kotowski, H. Hlomani, D. Stacey, and P. Baker. An ontology-driven approach to mobile data collection applications for the health-care industry. In *Proc of Int'l Information Reuse and Integration (IRI)*, 2012.
- [32] R. Holmes, R. J. Walker, and G. C. Murphy. Strathcona example recommendation tool. In *Proc of the European Software Eng Conf held jointly with Int'l Symp on Foundations of Software Eng*, 2005.
- [33] Adobe Systems Incorporated. Flex. <http://www.adobe.com/products/flex.html>.
- [34] S. Jarzabek and H. D. Trung. Flexible generators for software reuse and evolution. In *ICSE NIER track*, 2011.
- [35] M. Lindvall and K. Sandahl. Practical implications of traceability. *Software - Practice and Experience*, 26(10):1161–80, 1996.
- [36] A. Marcus, A. Sergeev, V. Rajlich, and J. I. Maletic. An information retrieval approach to concept location in source code. In *WCRE*, 2004.
- [37] S. Medini, G. Antoniol, Y. Gueheneuc, M. Di Penta, and P. Tonella. Scan: An approach to label and relate execution trace segments. In *WCRE*, 2012.
- [38] S. Mittal, J.S.V. Varman, G. Chatzopoulou, M. Eirinaki, and N. Polyzotis. QueRIE: a query recommender system supporting interactive database exploration. In *Proc of Int'l Conf on Data Mining Workshops (ICDMW)*, 2010.
- [39] Oracle. Java platform SE. <http://docs.oracle.com/javase/>.
- [40] D. Poshyvanyk, M. Gethers, and A. Marcus. Concept location using formal concept analysis and information retrieval. *Transactions on Software Engineering and Methodology*, 21(4):23:1–23:34, 2010.
- [41] R. Prieto-Diaz. Implementing faceted classification for software reuse. In *ICSE*, 1990.
- [42] V. Rajlich and N. Wilde. The role of concepts in program comprehension. In *Proc Int'l Workshop on Program Comprehension*, 2002.
- [43] Steven P. Reiss. Semantics-based code search. In *ICSE*, 2009.
- [44] M. P. Robillard, R. J. Walker, and T. Zimmermann. Recommendation systems for software engineering. *IEEE Software*, 27(4):80–86, 2010.
- [45] M. Smialek, A. Kalnins, E. Kalnina, A. Ambroziewicz, T. Straszak, and K. Wolter. Comprehensive system for systematic case-driven software reuse. In *SOFSEM 2010: Theory and Practice of Computer Science*, volume 5901, pages 697–708. Springer Berlin Heidelberg, 2010.
- [46] R. N. Taylor, N. Medvidovic, and E.M. Dashofy. *Software Architecture: Foundations, Theory, and Practice*. John Wiley & Sons, 2010.

# The Agile Quality Culture

## A survey on agile culture and software quality

Bruno Henrique Oliveira, Simone do Rocio Senger de Souza

Institute of Math and Computer Sciences  
São Paulo University  
São Carlos - SP, Brazil

E-mail: (brunohen, srocio)@icmc.usp.br

### Abstract

*Despite the importance of the agile methods and the acknowledgement that they do contribute to the software quality and client satisfaction, little is known about what factors, habits or tools usage have high impact on the product quality. It has been observed that agile practitioners have a different posture towards the development process they use. This paper goal is to report an analysis of agile practitioner's opinion and behavior on agile development methods and what's the impact of those issues on the product quality. To construct the analysis an online survey was planned and executed with agile methods practitioners. The results shows that agile practitioners have a very good knowledge of the method they use, they know the benefits, and pledge to be committed to the development process. The subjects showed a great interest in test activities and quality assurance activities. It's possible to grasp an insight on the commitment of agile practitioners with their development method. The relation between agile practice and software quality may lie on the agile culture. The agile methods make their participants more engaged to the development process.*

**Keywords:** *Agile Quality, Agile Culture, Quality Assurance, Agile Survey*

### I INTRODUCTION

Agile methods are arousing interest both in academy and industry [1]. They have gained acceptance in the commercial area because they embrace non-stable requirements, focusing on collaboration between developers and customers [2]. The great variety of tools and methods, and its capability to adapt, change and meet the customer need doesn't comply with the existing quality models. The agility imposed by the very same methods is a barrier between agile and enshrined quality models.

Agile practitioners have been claiming since the agile methods appearance that the use of its techniques greatly improves the quality of software products. However, a closer look into such claims reveals that there is a lack of a comprehensive technique to evaluate how agile processes meet software quality requirements [3].

This survey is part of a wider research about agile methods and software quality.

After planning and executing a case study to determine which activities agile teams run to do software quality assurance, the authors perceived a different posture from the developers towards quality assurance and control when compared with more developers who work with non-agile software development.

This factor motivated two news studies. The first one is a case study to compare how the developers on non-agile software companies and agile based software companies perceive quality. The second study is a survey where we try to identify why agile practitioners have the feeling of higher product quality as result of their work.

In this paper we report the results a survey on agile practices focused on quality assurance and test activities. We identify most often used practices and the perception of teams about product quality and customer relationship.

The body of this paper is organized into five sections. The second section describes the research method applied on the research, along with the questions used on the survey. Section 3 presents the related work. Section 4 presents the results of the survey itself and last section makes conclusion on the results and addresses suggestions for futures work.

### II. RELATED WORK

Reference [3] brought an innovative technique for evaluating agile methodologies in order to determine which factors of software quality they improve. The technique used a set of adapted software quality factors as defined by [4, 5]. The authors broke down two agile methods, XP and Lean

Development, into process activities. Then for each process activity of the agile method, an evaluation of what software quality factors are met was done [3].

They concluded that agility introduces a paradigm shift in project management in the sense that every part of the software development process is reviewed with the aim of reducing the activities and number of deliverables to the minimum needed in any given situation. The move is in fact from a command oriented management structure to a facilitator oriented management system. As seen from the way software quality factors are defined in agile processes, the central players in the development process are the customer and developer and not the manager [3].

Reference [6] presented a new Agile Quality Assurance Model (AQAM), with the purpose of being a flexible method to incorporate new changes in the software industry and provides detailed guidelines and templates for real world implementation and customization focused on medium and small companies. The researchers concludes that customizability and evolution of software processes and their models are acquiring a growing importance in the software process community [6].

Considering that traditional quality assurance techniques are reporting based and rely on heavy weight inspection methods whereas Agile Quality Assurance techniques are built-in daily activities by teams, on the paper [7] the author aims to study various challenges faced in terms of assuring quality in Agile , what are the key drivers or indicators of Quality in agile and proposing the framework to evaluate what aspects of Agile improve the quality of the product in terms of bug rates, development time and costs. The obtained conclusions are that agile methods require changes in way we do quality assurance. In waterfall development process, quality and stability are usually addressed in the later phases of the release, when changes are more costly to fix. However, in agile development, smaller builds that provide incremental functionality are presented to customers early and often in a fast-paced, iterative process. Therefore, quality assurance has to bring quality and stability to each of these iterations to be effective [7]. It is belief that the research will help in empirically proving that agile methods have build in quality management system and aims to provide software organizations deeper understanding about what factors are important to ensure quality in agile. When agile methods have good impact on quality they also has good impact on productivity and cost that in turn results in increased business value, could be a potential research finding.

Continuous integration is an agile practice for the continuous integration of new Source Code into the code base including the automated compile, build and running of tests. From traditional quality assurance we know software metrics as a very good approach to measure software quality.

Combining both there is a promising approach to control and ensure the internal software quality. The proposed approach adds continuous measurement and continuous improvement as subsequent activities to continuous integration and establishes metric-based quality-gates for an agile quality assurance [8]. The authors concluded that agile practices for continuous integration offer new opportunities for traditional software measurement. It provides a measurement infrastructure for the continuous measurement, which measures both traditional and agile metrics. It also shows the shift of metrics and quality attributes over time and leads to proper and also continuous improvement activities selected via the GQM-Approach. The improvement process does not only lead to selective improvements but to automated quality gates to preserve the improvements enduringly [8].

### III. RESEARCH METHOD

A survey is a strategy or design for an empirical study to provide a quantitative or numeric description of some fraction of a population through the data collection process of asking questions of people [9, 10].

The strategy for this survey was the one described at [11]. It is an instantiation of a general process for conducting empirical studies in software engineering.

The survey was conducted according to the following steps:

1. Study Definition - Determine the goal of the study.
2. Study Design - Operationalize the study goals into a set of questions and select the subjects.
3. Implementation - Operationalize the design.
4. Execution - Collect and process data obtained from answered questionnaires.
5. Analysis - Interpret the data.
6. Packaging - Create and publish a report about the survey results.

In this section, we describe the questions used in our survey and the steps conducted in our research. They are presented in Sections A, B and C. The Analysis is presented in Section IV.

#### A. Definition

We have found several surveys that mentioned the importance of people commitment in agile methods, but most research had no background to confirm this statement. Four articles were found that do not address people behavior as it main objective, but consider it as an important factor to the success of agile methods utilization [1, 12, 13, 14].

This survey tries to identify why agile practitioners have a different posture towards software development process

and what's the impact of this behavior on the product quality.

## B. Design

The time of performance of the survey was set to one month by convenience. The questionnaire was sent to four previous known companies that work with agile development. Also the survey was published on agile communities in the social network LinkedIn<sup>1</sup>.

The questions for the survey were elaborated based on the objectives of the study. In order to adjust and improve the questionnaire, the survey was applied on two selected individuals. As result some survey questions had to be transformed into essay questions. The complexity of some topics cannot be assessed in multiple-choice questions.

The questionnaire was formulated in five parts. The first part is related to basic information and it's only used for basic classification of the subjects. The second part is about the development process and tries to identify adaptations and how the person interacts with the process and people on agile development. The third and fourth part assesses quality assurance activities and tests respectively. The last part assess about agile methods and the perception of quality of the agile developer.

The questions created for the survey were, in it's majority opened with a exploratory character with some partially structured items as described in [15, 11].

To represent a possible set of statements as a product of this work with quality and credibility, the possible threats to its validity are listed in this subsection as are the possible workarounds or mitigation actions to reduce the risk of each threat to occur.

- High levels of dropouts - To avoid incomplete fill of the questions on the questionnaire, the first questions assessed basic information on quality and tests, to make the subject more comfortable, giving him the context of what's was next.
- Assure that subjects have skills agile methods - To assure that a significant part of the subjects have knowledge on the matter, we invited companies that are known to use agile practices, so answers would be significantly more trustable.
- Answers are shallow - To mitigate the risk of having shallow answers, the opened questions were arranged in some classifications and hints about the questions theme were given. When the answer was not clear, on not possible to arrange into the classification, it was not considered.
- The information sample given on a questionnaire is not trustworthy - If the information gathered does not

represent an agile practice or, did not corresponded to what was asked, the answered questionnaire was not included.

- Survey procedures or questions are not clear - A pilot with two subjects was run to identify possible misunderstandings or misinterpretation of the questions.

## C. Implementation and execution

The survey was taken online, in order to facilitate the process of gathering information, allowing that the subjects to answer the survey in their own time [11]. Based on the questions the environment for the web survey was created using the Google Forms<sup>2</sup> and sent to the subjects.

Teams of four companies were invited to fill the questionnaire in. Also it was released on social networks focused on agile practices. Within a month the survey was closed, and the data exported to a spreadsheet file for processing the data.

Twenty-five people answered the questionnaire. The result of the data analysis is described on Section IV.

## D. Packing

The survey report containing the questionnaire, spreadsheets and survey plan used to create and analyze the data on this survey is available at [http://brunohen.com/agile\\_survey](http://brunohen.com/agile_survey).

## IV. RESULTS

Figure 1 shows that the most used agile method among the survey subjects was Scrum, followed by Extreme Programming and Lean Software Development. The Scrum approach focuses on the management of development process and therefore it's one of the more flexible methods. Considering the process descriptions given on the questionnaires, the answers shows that most of the subjects knows the method theyre using. More than 90% of the process descriptions had a high compliance to the agile methods descriptions on the literature.

Figure 1 also shows that most of the subjects do not work with a dedicated team for tests or quality assurance. Fourteen subjects stated that the quality of the final product is everyone's responsibility and activities such as code review and testing, should be done by the whole team, with few exceptions. Five of the subjects also believe that testers are better developers, despite the confirmation bias that can be brought by this scenario. Confirmation bias is defined as the tendency of people to verify their hypotheses rather than refuting them and thus it has an effect on all software testing.

<sup>1</sup><http://www.linkedin.com/>

<sup>2</sup><https://docs.google.com/templates?type=forms>

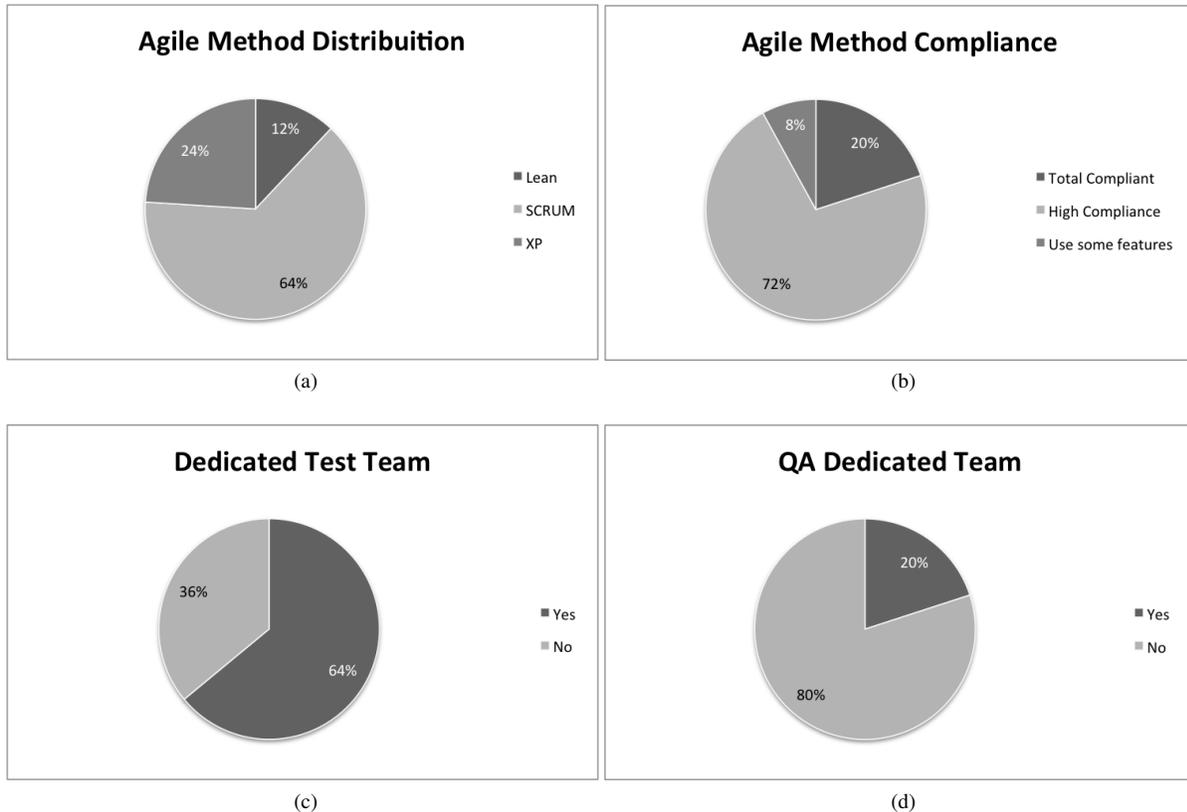


Figure 1: Quantitative data extracted from the questionnaire answers. Figure 1a show the distribution of agile methods among the subjects. The most used agile method was SCRUM followed by XP. Figure 1b shows that more than 90% of the subjects know the method they use as they are described on the literature. Figure 1c most of the subject's teams do not have a test dedicated team and even fewer have quality assurance dedicated team as show in Figure 1d.

It has been empirically proven that people have more tendencies to make positive tests rather than negative tests due to confirmation bias. However, it is highly probable that confirmation bias can be circumvented by factors like company culture [16].

An interesting information on this evaluation is regarding quality assurance activities, as show on Figure 2. More than 90% of the subjects claim that they do Code Review frequently, 64% claim that they do test activities and more than 50% participate on planning review. Further investigation is necessary to make any assumption, but this index seems elevated. Also some subjects claimed that they run these activities with an elevated frequency, this might be another issue to investigate.

Also for testing, subjects show a high level of usage on Unit Tools (such as PHPUnit and JUnit). Most of the subjects use proprietary solutions or Selenium<sup>3</sup> to execute front-end validation. To create the tests most subjects rely on the user history, which we can classify as functional

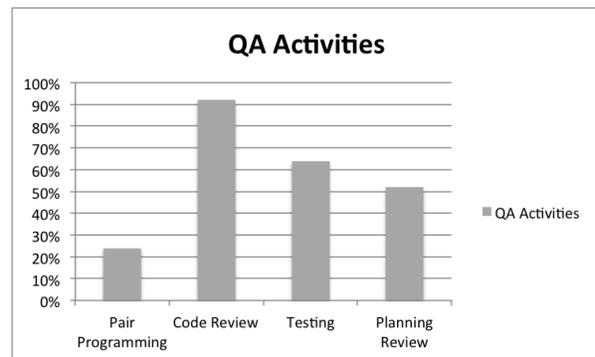


Figure 2: Quality assurance activities identified on the survey

tests. More than 20% of the subjects use Emma<sup>4</sup> for assess the code coverage. Emma can show coverage for stronger test criteria than Unit tools can provide.

<sup>3</sup><http://docs.seleniumhq.org/>

<sup>4</sup><http://emma.sourceforge.net/>

Analyzing the essay questions we could identify some convergence of opinions around the usage of agile methods.

About client satisfaction, the answers were divided into few or none contact with the client and excellent communication with the client. The low or none contact with the client came from agile projects that built SaaS, or shelf products. Instead of a client, they had a business analyst on the team. The analyst knows the business rules of the project, but he also knows technology and programming, which was pointed out as the main factor of success.

In the other hand, teams that had contact with the client have reported a good experience. In some cases the client was trained on agile methods in order to participate as much as possible on the project execution. Subjects also reported that once the client understood the agile method dynamic, they got more involved and participative, being this one of the most important keys to the project success. About 50% of the subjects that had contact with the client reported a communication improve or a desire to improve their communication with the client.

Few subjects had experience developing software with non-agile approaches. We asked about differences on non-agile and agile methods. The answers pointed out by those subjects addressed:

- Using agile methods makes easier to keep track of the project. They also reported they feel better knowing where the project is going and what are their part on it.
- Easier communication with client, being the fast feedback one of the most important contribution to the project.
- Focus on small features made the development process easier, helping to fix software bugs whenever they appeared.
- Client involvement helps to make the final scope closer to the one that was expected. Also, in the subject's perception, the client understood the importance of some definitions better, and by the end of the project, they were more worried about defining rules earlier to facilitate the developer's work.

Nine of the subjects reported that they do not imagine themselves in the future working with a development approach other than agile methods (even tough this was not a question of the survey).

On the project challenges, subjects reported that the agile method have a great impact on their posture towards the projects. More than 50% stated that they feel thrilled when facing project complications. The most pointed out reasons were:

- Perspective of the whole project make it more exciting and challenging.
- Dynamic work make developers more motivated.

- It facilitates the team work in a lot of aspects such as communication, performance and quality assurance.
- Motivates better communication.

## V. CONCLUSIONS AND FUTURE RESEARCH

There's a need on software development for developers to know more about quality and to be engaged on making better software. It's important that development teams know how to assess the quality of a software product. From the results of the survey we can conclude that agile methods do have quality assurance activities. The frequency of these activities is high, considering that they occur during the short iterations.

Considering qualitative and quantitative data is possible to identify that agile developers knows the importance of quality assurance activities and testing. Despite the low usage of documentation, teams have structured a process for testing, evaluation, user test acceptance, and communication tools for propagating the project scope and it's evolution.

What can be extracted from the essay questions is that agile practitioners are always trying to show the results of agile methods. They know the process and they believe it's the best way to develop software. Results pointed on Section IV lead us to believe that the agile practices is a culture among the agile practitioners and this might be one of the most important reasons agile development shows good quality results, as already mentioned in the literature in [12, 17, 2].

This survey was used as part of an action research, as it's limited by its sample size and distribution. The sample may not be significant for conclusion about the agile developers community, but it indicates that there is more to investigate on this matter. Some of the subjects worked on same company, affecting the results according to a company agile culture and method.

As future work we intend to re-run this survey to increase the number of subjects and separating questions by agile method so it's possible to make a deeper analysis on the relation between development process and software quality. Some developers reported that the participation of the client a crucial factor for project's success. We also intend to deepen the study on software team engagement and how can developers engage customers to the agile culture.

## ACKNOWLEDGMENT

We thank all the participants who voluntarily performed the experiments, ICMC-USP for providing the infrastructure that allowed this project to be developed and CNPq for the financial support.

## References

- [1] B. Boehm, "Get ready for agile methods, with care," *Computer*, vol. 1, pp. 64–69, 2002.
- [2] M. Huo, V. June, L. Zhu, and M. A. Babar, "Software quality and agile methods," in *Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC04)*, 28th Annual International Software and Application Conference (COMPSA'04), 2004.
- [3] E. Mnkandla and B. Dwolatzky, "Defining agile software quality assurance," in *International Conference on Software Engineering Advances (ICSEA'06)*, 2006.
- [4] J. A. McCall, *An Introduction to Software Quality Metrics*. Petrocelli, 1979.
- [5] B. Meyer, *Object-Oriented Software Construction*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1st ed., 1988.
- [6] G. Hongying and Y. Cheng, "A customizable agile software quality assurance model," in *Information Science and Service Science (NISS), 2011 5th International Conference on New Trends in*, vol. 2, pp. 382–387, 2011.
- [7] S. Bhasin, "Quality assurance in agile: A study towards achieving excellence," in *AGILE India (AGILE INDIA), 2012*, pp. 64–67, 2012.
- [8] A. Janus, R. Dumke, A. Schmietendorf, and J. Jger, "The 3c approach for agile quality assurance," in *2012 3rd International Workshop on Emerging Trends in Software Metrics WETSoM 2012 Proceedings*, pp. 9–13, 2012.
- [9] J. Floyd J Fowler, *Survey Research Methods*. SAGE Publications, Inc, 2009.
- [10] J. W. Creswell, *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*. SAGE Publications, Inc, 2008.
- [11] T. Punter, M. Ciolkowski, B. Freimut, and I. John, "Conducting on-line surveys in software engineering," in *International Symposium on Empirical Software Engineering (ISESE03)*, IEEE Computer Society, 2003.
- [12] A. Cockburn, J. Highsmith, K. Beck, R. Jeffries, M. Beedle, A. v. Bennekum, W. Cunningham, M. Fowler, J. Greening, A. Hunt, J. Kern, B. Merick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas, "Agile manifesto, <http://agilemanifesto.org/>," 2001.
- [13] B. Ramesh, "Agile software development: Ad hoc practices or sound principles?," *IT Professional*, vol. 9, pp. 41–47, 3 2007.
- [14] M. Pikkarainen, J. Haikara, O. Salo, P. Abrahamsson, and J. Still, "The impact of agile practices on communication in software development," *Kluwer Academic Publishers*, no. 13, pp. 303–337, 2008.
- [15] D. C. Miller, *Handbook of research design and social measurement*. SAGE Publications Inc., 1991.
- [16] G. Calikli, A. Bener, and B. Arslan, "An analysis of the effects of company culture, education and experience on confirmation bias levels of software developers and testers," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - ICSE '10*, vol. 2, (New York, New York, USA), p. 187, ACM Press, 2010.
- [17] P. Rodriguez, J. Markkula, M. Oivo, and K. Turula, "Survey on agile and lean usage in finnish software industry," in *Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement - ESEM '12*, (New York, New York, USA), p. 139, ACM Press, 2012.

# Story Point Approach based Agile Software Effort Estimation using Various SVR Kernel Methods

Shashank Mouli Satapathy<sup>1</sup>, Aditi Panda<sup>2</sup>, Santanu Kumar Rath<sup>3</sup>

Department of Computer Science and Engineering

National Institute of Technology

Rourkela - 769008, Odisha, India.

Email: shashankamouli@gmail.com<sup>1</sup>, aditipanda@yahoo.com<sup>2</sup>, skrath@nitrkl.ac.in<sup>3</sup>

**Abstract**—Agile software development process represents a major departure from traditional, plan-based approaches to software engineering. Estimating effort of agile software accurately in early stage of software development life cycle is a major challenge in the software industry. For improving the estimation accuracy, various optimization techniques are used. The Support Vector Regression (SVR) is one of these techniques that helps in getting optimal estimated values. The main objective of the research work carried out in this paper is to estimate the effort of agile softwares using story point approach. An attempt has been made to optimize the results obtained from story point approach using various SVR kernel methods to achieve better prediction accuracy. A performance comparison of the models obtained using various SVR kernel methods is also presented in order to highlight performance achieved by each method.

**Keywords**—Agile Software Development; Software Effort Estimation; Story Point Approach; Support Vector Regression.

## I. INTRODUCTION

Agile methods are a reaction to the traditional ways of developing software and acknowledge the “need for an alternative to documentation driven, heavyweight software development processes” [1]. Now-a-days effort estimation of softwares developed using agile methods are creating a buzz in the software development community, drawing their fair share of advocates and opponents. Agile software development emphasizes on good communication between the developers, rapid delivery of software and change on demand [2]. Due to the increasing use of agile methods in industry in the last few years, the effort estimation of software developed by agile methods has become an important issue [3].

Accurate effort estimation of an agile software project is always important for performing cost-benefit analysis and determining the feasibility of the project [4]. Agile methodologies use user stories as requirement artifacts. In the case of agile projects, story point is used to measure the effort required to implement a user story. And by adding up the estimates of user stories that were finished during an iteration (story point iteration), the project velocity is obtained. Hence in this paper, total number of story points are used along with project velocity to calculate the effort required for agile software development. In order to achieve better prediction accuracy, various kernel methods-based support vector regression techniques are introduced. In case of SVR, a linear regression function is computed in a high dimensional feature space. The input data are mapped via a nonlinear function. Further, the SVR kernel methods can be applied in transforming the

input data and then based on these transformations, an optimal boundary between the possible outputs can be obtained. The results of all these SVR kernel-based techniques are compared and their performance is accessed.

## II. RELATED WORK

Keaveney et al. [5] investigated the applicability of conventional estimation techniques towards agile development approaches by focusing on four case studies of agile method used across different organizations. Coelho et al. [6] described the steps followed in story point-based method for effort estimation of agile software and highlighted the areas which need to be looked into further research. Andreas Schmietendorf et al. [2] provided an investigation about estimation possibilities, especially for the extreme programming paradigm. Ziauddin et al. [7] developed an effort estimation model for agile software projects. The model was calibrated using the empirical data collected from 21 software projects. The experimental results show that model has good estimation accuracy in terms of MMRE and PRED (n). Hearty et al. [8] proposed a bayesian network model of an Extreme Programming environment and showed how it can learn from project data in order to make quantitative effort predictions and risk assessments without requiring any additional metrics.

Adriano L.I. Oliveira [9] provided a comparative study on SVR, radial basis function neural networks (RBFNs) and linear regression for estimation of software project effort. The experiment was carried out using NASA project data sets and the result showed that SVR performs better than RBFN and linear regression. Kocaguneli et al. [10] investigated non-uniform weighting through kernel density estimation and found that nonuniform weighting through kernel methods cannot outperform uniform weighting Analogy Based Estimation (ABE). Braga et al. [11] proposed and investigated the use of a genetic algorithm approach for selecting an optimal feature subset and optimizing SVR parameters simultaneously aiming to improve the precision of the software effort estimates.

## III. EVALUATION CRITERIA

The performance of the various models generated using SVR kernel methods can be evaluated by using the following evaluation criteria.

- The **Mean Magnitude of Relative Error (MMRE)** can be achieved through the summation of MRE over

N observations

$$MMRE = \sum_{i=1}^N \frac{|ActualEffort_i - PredictedEffort_i|}{ActualEffort_i} \quad (1)$$

- The **Prediction Accuracy (PRED)** can be calculated as:

$$PRED = (1 - (\frac{\sum_{i=1}^N |actual_i - predicted_i|}{N})) * 100 \quad (2)$$

where

N = Total number of data in the test set.

#### IV. PROPOSED APPROACH

The proposed approach is based on twenty one project data set [7]. The data set is used to evaluate software development effort and to validate the improvement. The results obtained in the validation process have provided initial experimental evidence of the effectiveness of story point approach. The block diagram, shown in Figure 1, displays the proposed steps used to determine the predicted effort using various kernel-based SVR techniques. To calculate the effort of a given

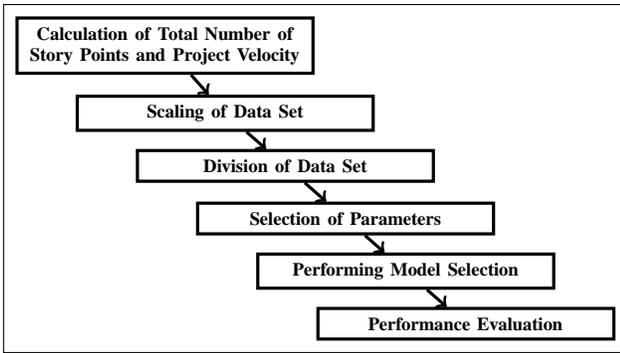


Fig. 1: Proposed Steps Used for Effort Estimation using Various SVR Kernel Methods

software project, basically the following steps have been used.

#### Steps in Effort Estimation

- 1) **Calculation of Total Number of Story Points and Project Velocity:** After collecting the data from other developed projects, the number of story point and their corresponding project velocity is calculated from the user stories using the steps provided in section 3.
- 2) **Scaling of Data Set:** In this step, the generated number of story points and project velocity values are used as input arguments and are scaled within the range [0,1]. Let X be the data set and x is an element of the data set, then the scaled value of x can be calculated as :

$$x' = \frac{x - \min(X)}{\max(X) - \min(X)} \quad (3)$$

where

$x'$  = Scaled value of x within the range [0,1].

$\min(X)$  = the minimum value for the data set X.

$\max(X)$  = the maximum value for the data set X. If  $\max(X)$  is equal to  $\min(X)$ , then  $\text{Normalized}(x)$  is set to 0.5.

- 3) **Division of Data Set:** In this step, the data set is divided into three parts using five-fold cross validation approach. These are learning set, validation set and test set.
- 4) **Selection of Parameters:** The tunable parameters have been selected to find the best parameter C and  $\gamma$  using a five-fold cross validation procedure .
- 5) **Performing Model Selection:** In this step, a five-fold cross validation approach is implemented for model selection. The model that provides lowest MeanMagnitude of Relative Error (MMRE) value and highest Prediction Accuracy (PRED) value is selected as the best model for each fold.
- 6) **Performance Evaluation:** The performance of the model is evaluated using final MMRE and PRED values obtained from test samples. The performance of the model is evaluated using final MMRE and PRED values obtained from test samples. The average of MMRE and PRED accuracy(from each fold) are found out for this.

The SVR kernel-based methods are implemented using the above steps. Finally, a comparison of results obtained using various kernels-based SVR effort estimation model is presented to assess their performances.

#### V. EXPERIMENTAL DETAILS

In this paper to implement the proposed approaches, data set given in [7] is used. The detailed description about the data set has already been provided in the proposed approach section. The inputs of the model are total number of story points and project final velocity and the output is the effort i.e., time required to complete the project. Initially, three sets of data are extracted from the available data set i.e., training data, testing data and validation data. First of all, every fifth data out of those two data sets, is extracted for testing purpose and rest data will be used for training purpose. Then the training data is partitioned into the learning and validation sets. After partitioning data into learning set and validation set, the model selection for  $\epsilon$  and  $\gamma$  is performed using five fold cross validation process. In this paper, to perform model selection, the  $\epsilon$  and  $\gamma$  values are varied over a particular range. The  $\gamma$  value ranges from  $2^{-7}$  to  $2^7$  and  $\epsilon$  value ranges from 0 to 5. Hence, ninety number of models are generated to perform the model selection operation.

Table I and II show the validation error of the ninety models generated using SVR linear kernel and SVR polynomial kernel respectively based on the value of  $\epsilon$  and  $\gamma$ . For SVR Linear kernel, 0.0132 value has been chosen as the minimum validation error. Hence based on the minimum validation error, the best model is  $C = 0.78121$ ,  $\gamma = 0.0078125$  and  $\epsilon = 0$ . Similarly for SVR Polynomial kernel, 0.0536 value has been chosen as the minimum validation error. Hence based on the minimum validation error, the best model is  $C = 0.78121$ ,  $\gamma = 8$  and  $\epsilon = 0$ .

Similarly, Table III and IV show the validation error of ninety models generated using SVR RBF kernel and SVR

TABLE I: Validation Errors Obtained Using SVR Linear Kernel

$\gamma = 2^{-\gamma}$	$\epsilon = 0$	1	2	3	4	5
$2^{-6}$	0.0132	0.0712	0.0712	0.0712	0.0712	0.0712
$2^{-5}$	0.0132	0.0712	0.0712	0.0712	0.0712	0.0712
$2^{-4}$	0.0132	0.0712	0.0712	0.0712	0.0712	0.0712
$2^{-3}$	0.0132	0.0712	0.0712	0.0712	0.0712	0.0712
$2^{-2}$	0.0132	0.0712	0.0712	0.0712	0.0712	0.0712
$2^{-1}$	0.0132	0.0712	0.0712	0.0712	0.0712	0.0712
$2^0$	0.0132	0.0712	0.0712	0.0712	0.0712	0.0712
$2^1$	0.0132	0.0712	0.0712	0.0712	0.0712	0.0712
$2^2$	0.0132	0.0712	0.0712	0.0712	0.0712	0.0712
$2^3$	0.0132	0.0712	0.0712	0.0712	0.0712	0.0712
$2^4$	0.0132	0.0712	0.0712	0.0712	0.0712	0.0712
$2^5$	0.0132	0.0712	0.0712	0.0712	0.0712	0.0712
$2^6$	0.0132	0.0712	0.0712	0.0712	0.0712	0.0712
$2^7$	0.0132	0.0712	0.0712	0.0712	0.0712	0.0712

TABLE II: Validation Errors Obtained Using SVR Polynomial Kernel

$\gamma = 2^{-\gamma}$	$\epsilon = 0$	1	2	3	4	5
$2^{-6}$	0.0916	0.0712	0.0712	0.0712	0.0712	0.0712
$2^{-5}$	0.0916	0.0712	0.0712	0.0712	0.0712	0.0712
$2^{-4}$	0.0916	0.0712	0.0712	0.0712	0.0712	0.0712
$2^{-3}$	0.0916	0.0712	0.0712	0.0712	0.0712	0.0712
$2^{-2}$	0.0896	0.0712	0.0712	0.0712	0.0712	0.0712
$2^{-1}$	0.0746	0.0712	0.0712	0.0712	0.0712	0.0712
$2^0$	0.0764	0.0712	0.0712	0.0712	0.0712	0.0712
$2^1$	0.2870	0.0712	0.0712	0.0712	0.0712	0.0712
$2^2$	0.1987	0.0712	0.0712	0.0712	0.0712	0.0712
$2^3$	0.0536	0.0712	0.0712	0.0712	0.0712	0.0712
$2^4$	0.0646	0.0712	0.0712	0.0712	0.0712	0.0712
$2^5$	0.0646	0.0712	0.0712	0.0712	0.0712	0.0712
$2^6$	0.0643	0.0712	0.0712	0.0712	0.0712	0.0712
$2^7$	0.0619	0.0712	0.0712	0.0712	0.0712	0.0712

TABLE III: Validation Errors Obtained Using SVR RBF Kernel

$\gamma = 2^{-\gamma}$	$\epsilon = 0$	1	2	3	4	5
$2^{-6}$	0.0874	0.0712	0.0712	0.0712	0.0712	0.0712
$2^{-5}$	0.0833	0.0712	0.0712	0.0712	0.0712	0.0712
$2^{-4}$	0.0756	0.0897	0.0897	0.0897	0.0897	0.0712
$2^{-3}$	0.0601	0.0712	0.0712	0.0712	0.0712	0.0712
$2^{-2}$	0.0364	0.0712	0.0712	0.0712	0.0712	0.0712
$2^{-1}$	0.0139	0.0712	0.0712	0.0712	0.0712	0.0712
$2^0$	0.0113	0.0712	0.0712	0.0712	0.0712	0.0712
$2^1$	0.0088	0.0712	0.0712	0.0712	0.0712	0.0712
$2^2$	0.0115	0.0712	0.0712	0.0712	0.0712	0.0712
$2^3$	0.0158	0.0712	0.0712	0.0712	0.0712	0.0712
$2^4$	0.0222	0.0712	0.0712	0.0712	0.0712	0.0712
$2^5$	0.0315	0.0712	0.0712	0.0712	0.0712	0.0712
$2^6$	0.0370	0.0712	0.0712	0.0712	0.0712	0.0712
$2^7$	0.0453	0.0712	0.0712	0.0712	0.0712	0.0712
$2^8$	0.0525	0.0712	0.0712	0.0712	0.0712	0.0712

Sigmoid kernel respectively based on the value of  $\epsilon$  and  $\gamma$ . For SVR RBF kernel, 0.0088 value has been chosen as the minimum validation error. Hence based on the minimum validation error, the best model is  $C = 0.78121$ ,  $\gamma = 1$  and  $\epsilon = 0$ . Similarly for SVR Sigmoid kernel, 0.0169 value has been chosen as the minimum validation error. Hence based on the minimum validation error, the best model is  $C = 0.78121$ ,  $\gamma = 1$  and  $\epsilon = 0$ .

Finally, based on the model's parameter values, the model is again trained and tested using training and testing data

TABLE IV: Validation Errors Obtained Using SVR Sigmoid Kernel

$\gamma = 2^{-\gamma}$	$\epsilon = 0$	1	2	3	4	5
$2^{-6}$	0.0895	0.0712	0.0712	0.0712	0.0712	0.0712
$2^{-5}$	0.0874	0.0712	0.0712	0.0712	0.0712	0.0712
$2^{-4}$	0.0833	0.0712	0.0712	0.0712	0.0712	0.0712
$2^{-3}$	0.0754	0.0712	0.0712	0.0712	0.0712	0.0712
$2^{-2}$	0.0593	0.0712	0.0712	0.0712	0.0712	0.0712
$2^{-1}$	0.0348	0.0712	0.0712	0.0712	0.0712	0.0712
$2^0$	0.0115	0.0712	0.0712	0.0712	0.0712	0.0712
$2^1$	0.0080	0.0712	0.0712	0.0712	0.0712	0.0712
$2^2$	0.0169	0.0712	0.0712	0.0712	0.0712	0.0712
$2^3$	0.1693	0.0712	0.0712	0.0712	0.0712	0.0712
$2^4$	0.4131	0.0712	0.0712	0.0712	0.0712	0.0712
$2^5$	0.4285	0.0712	0.0712	0.0712	0.0712	0.0712
$2^6$	0.4503	0.0712	0.0712	0.0712	0.0712	0.0712
$2^7$	0.2520	0.0712	0.0712	0.0712	0.0712	0.0712
$2^8$	0.1211	0.0712	0.0712	0.0712	0.0712	0.0712

set respectively to estimate the effort. After implementing the SVR-based model using four different kernel methods for software effort estimation, the following results are generated.

**SVR Linear Kernel Result:**

**Param: -s 3 -t 0 -c 0.78121 -g 0.0078125 -p 0**

\* Mean Squared Error (MSE\_TEST) = 0.2068

\* Squared correlation coefficient = 0.8936

**SVR Polynomial Kernel Result:**

**Param: -s 3 -t 1 -c 0.78121 -g 8 -p 0**

\* Mean Squared Error (MSE\_TEST) = 0.2057

\* Squared correlation coefficient = 0.9006

**SVR RBF Kernel Result:**

**Param: -s 3 -t 2 -c 0.78121 -g 1 -p 0**

\* Mean Squared Error (MSE\_TEST) = 0.0030

\* Squared correlation coefficient = 0.9843

**SVR Sigmoid Kernel Result:**

**Param: -s 3 -t 3 -c 0.78121 -g 1 -p 0**

\* Mean Squared Error (MSE\_TEST) = 0.0194

\* Squared correlation coefficient = 0.8734

The *squared correlation coefficient* ( $r^2$ ) is also known as the *coefficient of determination*. It is the proportion of variance in actual effort that can be accounted for by knowing story point value for test set. In the output generated, it is quite clearly observed that the *squared correlation coefficient* value for RBF kernel is very high (greater than 0.95). Thus it can be observed that a strong positive correlation exists between the story point, velocity and the predicted effort required to develop the software, and minor changes in one lead to significant changes in another. The proposed model generated using the SVR

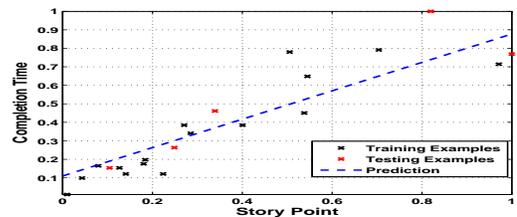


Fig. 2: SVR Linear Kernel based Effort Estimation Model

linear, polynomial, RBF and sigmoid kernel have been plotted based on the training and testing sample data set as shown in Figure 2, 3, 4 and 5. The graphs show the variation of

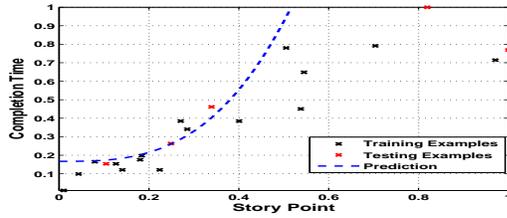


Fig. 3: SVR Polynomial Kernel based Effort Estimation Model

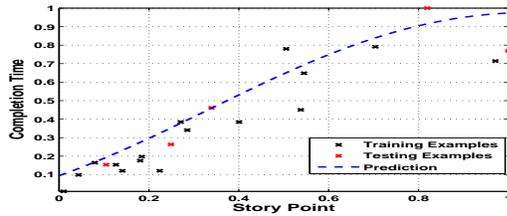


Fig. 4: SVR RBF Kernel based Effort Estimation Model

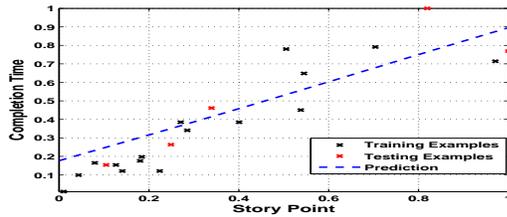


Fig. 5: SVR Sigmoid Kernel based Effort Estimation Model

predicted effort value with respect to its corresponding class point value. In these graphs, it is clearly shown that the data points are very little dispersed than the regression line. Hence the correlation is higher. While comparing the dispersion of data points from the predicted model in the above graphs, it is clearly visible that the data points are less dispersed for SVR RBF kernel-based model than other models. Hence, SVR RBF kernel-based effort estimation model exhibits lower error value and higher prediction accuracy value.

## VI. COMPARISON

On the basis of results obtained, the estimated effort using various SVR kernel methods are compared. While using the MMRE and PRED in evaluation, good results are implied by lower value of the MMRE and higher value of the PRED.

TABLE V: Comparison of Efforts obtained using various SVR kernel methods

Actual Effort	SVR Linear Effort	SVR Polynomial Effort	SVR RBF Effort	SVR Sigmoid Effort
1	0.4615	0.3436	0.2332	0.4194
2	0.7692	0.7698	1.1027	0.7865
3	0.2637	0.2295	0.2059	0.2325
4	1.0000	0.7030	1.9282	0.8908
5	0.1538	0.1635	0.1691	0.1489

Table. V shows the comparison of actual effort with an estimated effort by various SVR kernel methods for agile software on the five data taken for testing out of a data set

of twenty one data.

TABLE VI: Comparison of errors and prediction accuracy values obtained using various SVR kernel methods

	Various SVR Kernel Methods	MMRE	PRED
1	SVR Linear Kernel	0.1492	90.8112%
2	SVR Polynomial Kernel	0.4350	68.7382%
3	SVR RBF Kernel	0.0747	95.9052%
4	SVR Sigmoid Kernel	0.1929	89.7646%

The Table VI displays the final comparison of MMRE and PRED values for different SVR kernel methods. The result shows that effort estimation using SVR RBF kernel-based model gives lower MMRE value and higher prediction accuracy value than those obtained using other SVR kernel methods.

## VII. CONCLUSION

The story point approach is one of the effort estimation models that can be used for agile software's effort estimation. In this paper, first the total number of story points and project velocity are used to estimate the effort involved in developing an agile software product. The results obtained are optimized using four different support vector regression kernel methods. At the end of the study, the results generated are compared in order to access their accuracy. While comparing the results obtained using various SVR kernel methods, it can be concluded that RBF kernel-based support vector regression technique outperformed other three kernel methods. The computations for above procedure have been implemented, and outputs were generated using MATLAB. This approach can also be extended by applying other machine learning techniques such as SGB, Random Forest etc. on the story point approach.

## REFERENCES

- [1] M. Fowler and J. Highsmith, "The agile manifesto," *Software Development*, vol. 9, no. 8, pp. 28–35, 2001.
- [2] A. Schmietendorf, M. Kunz, and R. Dumke, "Effort estimation for agile software development projects," in *5th Software Measurement European Forum*, 2008.
- [3] D. Cohen, M. Lindvall, and P. Costa, "An introduction to agile methods," *Advances in Computers*, vol. 62, pp. 1–66, 2004.
- [4] S. M. Satapathy, M. Kumar, and S. K. Rath, "Fuzzy-class point approach for software effort estimation using various adaptive regression methods," *CSI Transactions on ICT*, vol. 1, no. 4, pp. 367–380, 2013.
- [5] S. Keaveney and K. Conboy, "Cost estimation in agile development projects." in *ECIS*, 2006, pp. 183–197.
- [6] E. Coelho and A. Basu, "Effort estimation in agile software development using story points," *development*, vol. 3, no. 7, 2012.
- [7] Z. K. Zia, S. K. Tipu, and S. K. Zia, "An effort estimation model for agile software development," *Advances in Computer Science and its Applications*, vol. 2, no. 1, pp. 314–324, 2012.
- [8] P. Hearty, N. Fenton, D. Marquez, and M. Neil, "Predicting project velocity in xp using a learning dynamic bayesian network model," *Software Engineering, IEEE Transactions on*, vol. 35, no. 1, pp. 124–137, 2009.
- [9] A. L. Oliveira, "Estimation of software project effort with support vector regression," *Neurocomputing*, vol. 69, no. 13, pp. 1749–1753, 2006.
- [10] E. Kocaguneli, T. Menzies, and J. W. Keung, "Kernel methods for software effort estimation," *Empirical Software Engineering*, vol. 18, no. 1, pp. 1–24, 2013.
- [11] P. L. Braga, A. L. Oliveira, and S. R. Meira, "A ga-based feature selection and parameters optimization for support vector regression applied to software effort estimation," in *Proceedings of the 2008 ACM symposium on Applied computing*. ACM, 2008, pp. 1788–1792.

# Identifying and Recording Software Architectural Assumptions in Agile Development

Chen Yang

State Key Lab of Software Engineering  
School of Computer, Wuhan University  
Wuhan, China  
cyang@whu.edu.cn

Peng Liang\*

State Key Lab of Software Engineering  
School of Computer, Wuhan University  
Wuhan, China  
liangp@whu.edu.cn

**Abstract**—Architects and involved stakeholders constantly make Architectural Assumptions (AAs) in architecture design. These assumptions, as an important part of architectural knowledge, need to be well managed in the whole architecting lifecycle. However, they are always retained in the heads of various stakeholders and left undocumented, which results in architectural knowledge vaporization, especially in agile development when documentation is not a must and priority. In this paper, we propose a novel method that is composed of two parts – Architectural Assumption Library (AAL) to help architects identify AAs and Architectural Assumption Card (AAC) to record them based on a simplified conceptual model of AA. An architect from an industry project of Embedded Systems that using agile development was invited to employ this method in the project to identify and record AAs. It shows that the proposed method can help architects to easily identify and record AAs, which is promising to facilitate architecture design, maintenance, and evolution in agile development.

**Keywords**—software architecture; architectural assumption; agile development

## I. INTRODUCTION

Software architecture (SA) presents a high-level design of a system, which has an important role to manage complicated interactions between stakeholders of software-intensive systems in order to balance all kinds of constraints [1]. There are many definitions of SA. Bass *et al.* define SA as the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both [1]. In a design decision perspective, a software architecture is composed of a set of architectural design decisions (ADDs), which lead to the resulting design artifacts [2]. Traditional architecting process tends to employ big design up-front (BDUF) leading to massive documentation and implementation of YAGNI (you ain't gonna need it) features, which introduces redundancy in architecting as well as development effort [3].

Agility means to strip away as much of the effort-intensive activities in software development, commonly associated with the traditional software development methodologies, as possible to promote quick response to changing environments, changes in user requirements, accelerated project deadlines and the like [4]. Augustine notes that the common

characteristics of agile development methods are iterative and incremental lifecycle, which focus on small releases, collocated teams, and a planning strategy based on two levels: release plan driven by a feature or product backlog, and an iteration plan handling a task backlog [5]. These common characteristics adhere to the values of the agile manifesto [6].

It is challenging to apply SA in agile development. Proponents of agile approaches usually see little value for system's customers in the up-front design and evaluation of architecture [3]. Agile development on the contrary focuses on enough design till the last responsible moment which can lead to simplicity in development and a rapid and iterative process of producing artifacts [8]. To combine the benefits from both approaches (agility and architecture), a transformation from traditional SA to agile SA is emerging [8]. In an architecting process, numerous implicit and undocumented ADDs are more or less arbitrarily made on the fly based on personal experience, domain knowledge, budget constraints, available expertise, and the like [9]. Some researchers called these decisions as architectural assumptions (AAs) [9][10], which are an important part of architectural knowledge to be managed during the entire software lifecycle [11]. Therefore, how to identify and record AAs in agile development with a lightweight approach is a challenging issue to align architecture and agile development [8].

It worth noting that ISO 42010 standard [12] provides a conceptual model for architecture description, but this standard is too heavyweight to document SA in an agile environment and considers AA as an implicit component of certain standard elements, such as *Concern*, rather than an explicit element that should be documented. We tried to address this limitation by proposing a simplified model for recording AAs in this work.

In agile development, AAs are usually made implicitly. For example, a system needs high security, and we asked the architect of the system why not consider the external security (such as broken access control, cross-site scripting, etc.) of the system (we didn't find any information about the external security of the system in the SA documents) when he designed the architecture. He explained that: “*the system will be deployed in an internal environment which is secure enough, so we do not need any other mechanisms to achieve security.*”

---

\* Corresponding author

This work is partially sponsored by the NSFC under Grant No. 61170025 and the Ubbo Emmius scholarship by University of Groningen.

*This decision can avoid some redundant development work.”* This is a typical example of AA in software development without any documentation but stored in stakeholders’ (e.g., architect) mind. Does this AA fit the requirement of customers? What are the pros and cons of making this AA? It is difficult to answer such questions, since the AA was implicit and only a few stakeholders knew it, especially when the knowledge about the AA was vaporized gradually.

Implicit and undocumented AAs can lead to some problems such as misunderstanding of architecture, design violation, etc. It may be difficult to decide whether an assumption is an appropriate one for the current development context, but it is beneficial to make AAs explicit. Therefore, we need to manage AAs in an explicit and lightweight way, which can make AAs play an active and important role in agile development lifecycle.

The rest of this paper is organized as follows. Section II summarizes related work. In Section III, we introduce AA and propose a simplified conceptual model of AA. In Section IV, we propose a novel method – Architectural Assumption Library and Architectural Assumption Card which is based on the simplified conceptual model in Section III – to identify and record AAs. Section V presents a case study of using the proposed method in an industry agile project. Section VI concludes this work with future work directions.

## II. RELATED WORK

### A. Assumption in Software Development

The concept of assumption is not new. There are many type of assumptions in software development, such as requirement assumption, architectural assumption, etc., which focus on different aspects of software development [10]. Related work on assumptions in software development covers the areas of their classification, impact, and the methods of processing them (such as recovering, identifying, and documenting assumptions).

In [10], a prototype Assumptions Management System in software development was developed by Lewis *et al.*, which can extract assumptions from Java source code and record them into a repository for management. They provided a taxonomy of general assumptions in software development, including control assumptions (expected control flow), environment assumptions (expected environmental factors), data assumptions (expected input or output data), usage assumptions (expected use of applications), convention assumptions (followed standards or conventions in development).

### B. Architectural Assumption

In [13], Garlan *et al.* identified four general categories for architectural assumptions that are implicit and undocumented and consequently lead to architectural mismatch: nature of components, nature of connectors, global architectural structure, and software construction process. This categorization of architectural assumptions is based on a structural view of architecture, which considers SA as a set of structures, including components and connectors.

In [11], Ireo and Michel proposed a lightweight method to manage architectural assumptions in agile development. They summarized four main tasks of architectural assumptions management from existing literature: recording new assumptions, monitoring assumptions on a regular basis, searching for assumptions, and recovering past assumptions. They use the taxonomy of assumptions proposed in [14]. Our method focuses on identifying and recording AAs based on a simplified conceptual model in agile development, which covers the task “recording new assumptions” in [11].

In [9], Roeller *et al.* classified architectural assumptions into four groups as: implicit and undocumented (the architect is unaware of the assumption, or it concerns of course knowledge), explicit but undocumented (the architect takes a decision for a very specific reason), explicit and explicitly undocumented (the reasoning is hidden), explicit and documented (this is the preferred, but often exceptional, situation). They also developed a method (Recovering Architectural Assumptions Method, RAAM) to recover assumptions in architecture design.

In [14], Patricia and Hans distinguish architectural assumptions from requirements and constraints. An assumption meta-model was provided to document these assumptions in an explicit way. They classified architectural assumptions into three groups: managerial, organizational, and technical assumptions. However, it is complicated and might not be appropriate to use this meta-model and classification of assumptions in agile development. Our method of identifying and recording assumptions is lightweight to accommodate agile development and focuses on architectural assumptions.

### C. Knowledge Management in Software Development

In [15], Rus and Lindvall described the knowledge evolution cycle in software development, which is composed of five phases: (1) Originate/create knowledge (members of an organization develop knowledge through learning, problem solving, innovation, creativity, and importation from outside sources); (2) Capture/acquire knowledge (members acquire and capture information about knowledge in explicit forms); (3) Transform/organize knowledge (organizations organize, transform, or include knowledge in written material and knowledge bases); (4) Deploy/access knowledge (organizations distribute knowledge through education, training programs, automated knowledge-based systems, or expert networks); (5) Apply knowledge (organizations apply the knowledge - which is the ultimate goal of knowledge management and aims to make knowledge available whenever it is needed). Our proposed method covers all these five phases of architectural knowledge management through identifying and recording architectural assumptions (a type of architectural knowledge) [7].

## III. ARCHITECTURAL ASSUMPTION

We follow the definition of architectural assumption by Roeller *et al.* which regards AA as a kind of architectural design decision [9]. These decisions, as well as the reasons for making the decisions, are often not explicit and usually remain undocumented.

AAs are a kind of assumptions which focus on architecture design and they are an important type of architectural knowledge to be managed during the entire software development lifecycle [7]. AAs are always ignored and violated if they are not appropriately identified and recorded since these assumptions can fail at any time with serious consequences (e.g., misunderstanding of architecture, design violation, project delay, and low quality) if not being managed (identified and explicitly recorded) appropriately [11].

According to the four groups of AAs identified in [9]: implicit and undocumented, explicit but undocumented, explicit and explicitly undocumented, explicit and documented. We argue that most AAs in agile development are implicit and undocumented since documentation is not a must and priority [6]. Managing all the AAs in software lifecycle would cost a huge amount of resources which is not advocated and suitable in agile development. Therefore, AAs need to be managed in a lightweight manner. In this paper, we propose to identify and record AAs in agile development with a simplified conceptual model of AA as shown in Fig. 1.

This conceptual model is originated from the template for recording ADDs in [16]. The template has 14 elements: Issue, Decision, Status, Group, Assumptions, Constraints, Positions, Argument, Implications, Related decisions, Related requirements, Related artifacts, Related principles, and Notes. Compared to the ADD template, our simplified AA conceptual model only contains 5 elements: “Stakeholder” element can improve the traceability of AAs to related stakeholders, who are the source of this AA. “Description” element provides basic information about what the AA is. “Pros” and “Cons” support stakeholders to understand the positive and negative impact when considering the AA. “Relationship” between AAs provides traceability information between AAs which is needed in architecture impact analysis [17]. We argue that these five AA elements are the core and fundamental elements for identifying and recording AAs in agile development, which is also largely identified as the major elements (assumptions, arguments) in ADD conceptual models [18].

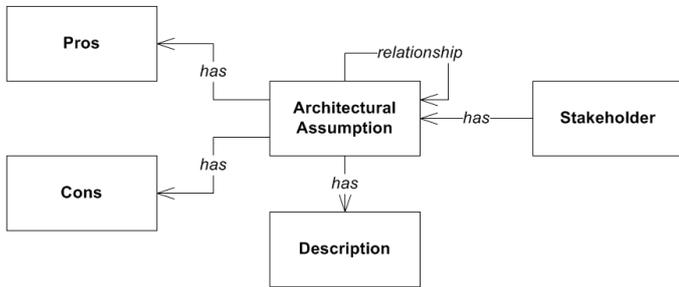


Fig. 1. A simplified conceptual model of Architectural Assumption

#### IV. METHOD

We propose a novel method, which is composed of two parts – Architectural Assumption Library (AAL) and Architectural Assumption Card (AAC) – for identifying and recording AAs in agile development. These two parts are both lightweight and based on the simplified conceptual model of AA proposed in Section III. A process of identifying and

recording AAs is specified to connect these two parts, which is described in detail in this section.

##### A. Architectural Assumption Library

Identifying AAs usually cause extra effort in development. Roeller *et al.* proposed to use interviews to identify AAs [9]. However, it is overcomplicated for agile development. For example, architects and developers may spend several days to discuss with stakeholders and identify AAs in several iterations in one release [9].

An Architectural Assumption Library acts as a repository of AAs, and helps architects to reduce the effort of identifying AAs in agile development by providing existing and reusable AAs as a reference and starting point, which can also reduce the time and identification effort on the discussion of AAs among stakeholders. There are two activities in the AAL management: creation and maintenance of AAL. The creation activity includes initializing an AAL and adding AAs into the AAL, for example, the new AAs can be identified, recorded, and added during project development. The maintenance activity of AAL includes checking and removing duplicated AAs in an AAL, clarifying AAs description and their relationships, etc. In an agile development, time and human resources are precious and it may not be appropriate to maintain the AAL during the development, because for example, checking and removing duplicated AAs should be performed by architects, which would cost additional effort but has little value to the current development. We propose that architects can work with stakeholders to maintain AAL at the end of a project which is time wise and when they still have a clear memory about the AAs of the project.

We use the simplified conceptual model of AA in Fig. 1 as a basic template to record and add AAs into AAL. The core items of the template are described in TABLE I.

TABLE I. TEMPLATE FOR RECORDING AND ADDING AAs INTO AAL

Item	Description
Assumption Name	This element describes the name of the AA. It should be meaningful to stakeholders, which means that it should not be named as “1”, “A”, etc. A meaningful name provides hint for users to quickly identify the AA.
Stakeholders	This element describes the stakeholders who are concerned about the AA. It provides the possibility to discuss the AA with the involved stakeholders to get a better understanding of the AA or modify this AA.
Description	This element provides more detailed information about the AA which helps stakeholders to understand the assumption.
Pros	This element helps stakeholders to get a clear understanding of the positive impact when considering the AA.
Cons	This element helps stakeholders to get a clear understanding of the negative impact when considering the AA.
Relationship(s)	This element specifies the relationship between AAs. Kruchten provided a taxonomy for ADD in [19]. We adapt this taxonomy for AA with

	minor change. TABLE II presents the five relationships between AAs.
--	---

TABLE II. RELATIONSHIPS BETWEEN ARCHITECTURAL ASSUMPTIONS

Relationship	Description
A Constrains B	Architectural assumption B is tied to assumption A. If A is removed, then B should be removed as well.
A Conflicts with B	A symmetrical relationship indicating that the two architectural assumptions A and B are mutually exclusive.
A Comprises B <sub>1</sub> , B <sub>2</sub> , ... B <sub>n</sub>	A high level and somewhat complicated, wide-ranging architectural assumption A is made of or decomposed into a series of narrower, more specific architectural assumptions B <sub>1</sub> , B <sub>2</sub> , ... B <sub>n</sub> .
A Is an alternative to B	A and B are similar architectural assumptions, and they can replace each other.
A Is related to B	There is a relation of some sort between the two architectural assumptions, but it is not of any kind listed above.

### B. Architectural Assumption Card

We provide Architectural Assumption Card based on the simplified conceptual model of AA as well (as shown in Fig. 1), to record AAs in agile development. These recorded AAs will be shared among all stakeholders of a project using blackboard or collaborative tools. An example AA recorded using AAC is provided in TABLE III. By using AAC, AAs can be recorded in an explicit and lightweight manner, which helps involved stakeholders to communicate the architecture design related to certain AAs.

TABLE III. AN EXAMPLE OF AA RECORDED USING ARCHITECTURAL ASSUMPTION CARD

AAC Element	Content of Element
Assumption Name	Response time of the system
Stakeholders	Jun Lan Yang (architect)
Assumption Description	The response time of the system should be within 2 seconds
Pros	High performance and usability
Cons	Extra effort (such as testing, hardware, etc.)
Relationship(s)	<i>Is constrained by</i> Response strategy

### C. Process of Identifying and Recording Architectural Assumptions

This process comprises three parts – using AAL to identify AAs, using AAC to record AAs, and adding new AAs into AAL (which is part of the creation activity in AAL management as discussed in Section IV.A). Fig. 2 shows the process of identifying and recording AAs using AAL and AAC, which is composed of five steps:

- (1) Use search strategies, such as keywords, to search related AAs in AAL;
- (2) If result AAs are retrieved, then identify the AA which is usable for the current design context. Otherwise, go to Step (4);

- (3) If the identified AA does not directly suit the current design context, then this AA should be adapted. Otherwise, go to Step (4);
- (4) Record the AA using AAC. The AA recorded can be both an AA identified and adapted in AAL or a new AA from the current project;
- (5) If the AA is new, then add it into AAL.

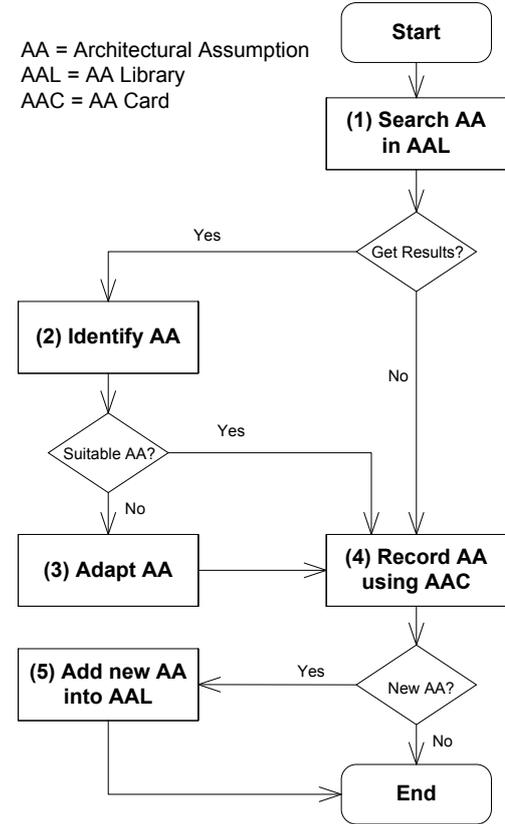


Fig. 2. Process of using AAL and AAC to identify and record AAs

## V. CASE STUDY

To evaluate our proposed method in an industrial context, we invited an architect from an industry project on Embedded Systems that employing agile development to use the simplified AA conceptual model and the process in the project to identify and record AAs. The collected AAs are limited because there is only one project involved, and we plan to further evaluate this method empirically with e.g., controlled experiments in the next step.

This project is an embedded system based on the Internet of Things techniques for managing fixtures in a big clothing factory. One fixture can be possibly moved on demand to everywhere in the factory which is composed of several workshops, and factory administration needs to find out where it is and its trajectory when they need it. They used readers, locators, electronic tags, and computers to build the hardware environment. The operation principles of the embedded system are described below:

- (1) Every fixture has a tag, providing its basic information, including a unique identity ID.

- (2) Every workshop has been divided into several areas and each of them has a locator. Position information of this area has been stored in locators, and they can activate tags through sending data with 125KHz (the frequency used in Radio Frequency Identification) in a fixed period.
- (3) Workshops have been divided and each of them has a reader. The reader can detect the tag and receive the data with position information from the tag.
- (4) Through the wireless network, the readers can send the data received from the tags to the computer.
- (5) After processing the data in the computer, users can retrieve the position information of the fixture.

We gave a short tutorial of AA to the architect of this embedded system, and asked him to construct an initial AAL using the template (as shown in TABLE I) based on his own experience from previous industry projects. More than 20 AAs were added in this initial AAL. During this project, the architect used the AAL to identify AAs and then recorded AAs using AAC. For example, the architect identified an architectural assumption “*using 433MHz model (the communication frequency between readers and tags) with 40 meters of the communication distance*” from the AAL, which is suitable for the current design context. Based on the actual environment (metal cover, signal interference, etc.) and his own experience, he adapted this AA into 433MHz model with 20 meters of the communication distance, and recorded this AA using AAC. The architect can use the AAs in AAL to reduce effort in architecture design. On the other hand, making these AAs explicit can help stakeholders to have a clear view in their perspectives: project managers and customers could know how many fixtures they need to cover the whole workshops; implementation consultant could figure out how to deploy the whole system; etc. Several examples of AA recorded by the architect using our method are listed in TABLE IV.

After the case study, we asked the architect to fill in a questionnaire to obtain a qualitative evaluation about our method. The questions and answers are shown in TABLE V.

The evaluation and feedback from the architect show that the process and model for identifying and recording AAs is easy to follow and takes acceptable effort to manage implicit AAs. The method has a positive impact on the development of project. He mentioned that some other activities related to AA are also important in agile development such as AA understanding, recovery, and evaluation. However, he suggested that we should consider the cost of introducing these AA activities into agile development and to provide appropriate tools to support the method. He also mentioned that whether this method can be suitable and readily used in an industrial setting depends on some factors, such as team organization, motivation of stakeholders, etc. Furthermore, he thought it was necessary to enact some shared strategies for AAs to help stakeholders easily apply the method in agile development.

According to his suggestions, we plan to classify AAs into some groups based on domain of projects or stakeholder role. This can be helpful to share AA information to the right

stakeholders and reduce the redundancy of use (users may not want to read AAs they do not care).

## VI. CONCLUSIONS AND FUTURE WORK

Architecture assumptions in agile development are normally retained in stakeholders’ mind which results in architectural knowledge vaporization [2]. It is important to manage them in an explicit way to avoid misunderstanding of architecture in architecting process. The major contributions of this work: (1) We propose that companies or organizations can build an Architectural Assumption Library to store their AAs which can be further used to identify AAs in agile development projects. (2) We propose to use Architectural Assumption Card to record AAs in agile development. Architectural assumptions can be explicitly written in a card or stored in collaborative tools to be better shared and communicated among stakeholders. (3) We propose a process of identifying and recording AAs using AAL and AAC with a simplified conceptual model of AA. (4) Finally, we report on an industrial case study, which was conducted to show the feasibility and value of our method in agile development.

We outline our future work in several points: (1) Empirically validate the proposed method in industrial agile projects using e.g., controlled experiments; (2) Provide supporting tools that help identify, record, and share AAs using the method; (3) Recover AAs and evaluate the quality of AAs which has a positive impact to the use of AAs in architecture design; (4) Classify AAs into different groups based on domain of projects or stakeholder role to easy their use in agile development.

## REFERENCES

- [1] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*, 3rd Edition. Addison-Wesley Professional, 2012.
- [2] J. Bosch. *Software architecture: The next step*. In: *Proceedings of the 1st European Workshop on Software Architecture (EWSA)*, pp. 194-199. Springer, 2004.
- [3] P. Abrahamsson, M. A. Babar, and P. Kruchten. *Agility and architecture: Can they coexist?*. *IEEE Software*, 27(2):16-22, 2010.
- [4] J. Erickson, K. Lyytinen, and K. Siau. *Agile modeling, agile software development, and extreme programming: The state of research*, *Journal of Database Management*, 16(4):88-100, 2005.
- [5] S. Augustine. *Managing Agile Projects*. Upper Saddle River, N.J.: Prentice Hall, 2005.
- [6] K. Beck et al. *Principles behind the Agile Manifesto*. <http://www.agilemanifesto.org/principles.html>, accessed on 2014-01-06.
- [7] R. C. de Boer, R. Farenhorst, P. Lago, H. van Vliet, V. Clerc, and A. Jansen. *Architectural knowledge: Getting to the core*. In: *Proceedings of the 3rd International Conference on the Quality of Software-Architectures (QoSA)*, pp. 197-214, 2007.
- [8] M. A. Babar, A. W. Brown, and I. Mistrik. *Agile Software Architecture: Aligning Agile Processes and Software Architectures*. Morgan Kaufmann, 2013.
- [9] R. Roeller, P. Lago, and H. van Vliet. *Recovering architectural assumptions*. *Journal of Systems and Software*, 79(4): 552-573, 2006.
- [10] G. A. Lewis, T. Mahatham, and L. Wrage. *Assumptions Management in Software Development*. Technical Report, CMU/SEI-2004-TN-021, 2004.
- [11] I. Ostacchini, and M. Wermelinger. *Managing assumptions during agile development*. In: *Proceedings of the 4th Workshop on SHaring and Reusing architectural Knowledge (SHARK)*, pp. 9-16, 2009.

[12] ISO/IEC/IEEE, ISO 42010:2011 International Standard, Systems and software engineering - Architecture description, 2011.

[13] D. Garlan, R. Allen, and J. Ockerbloom. Architectural mismatch: Why reuse is still so hard. *IEEE Software*, 26(4):66-69, 2009.

[14] P. Lago, and H. van Vliet. Explicit assumptions enrich architectural models. In: *Proceedings of the 27th International Conference on Software Engineering (ICSE)*, pp. 206-214, 2005.

[15] I. Rus, and M. Lindvall. Knowledge management in software engineering. *IEEE Software*, 19(3):26-38, 2002.

[16] J. Tyree, and A. Akerman. Architecture decisions: Demystifying architecture. *IEEE Software*, 22(2):19-27, 2005.

[17] Z. Li, P. Liang, and P. Avgeriou. Application of knowledge-based approaches in software architecture: A systematic mapping study. *Information and Software Technology*. 55(5):777-794, 2013.

[18] M. Shahin, P. Liang, and M. R. Khayyambashi. Architectural design decision: Existing models and tools. In *Proceedings of the Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture (WICSA/ECSA)*, pp. 293-296, 2009.

[19] P. Kruchten. An ontology of architectural design decisions in software intensive systems. In: *Proceedings of the 2nd Groningen Workshop on Software Variability Management (SVM)*, pp. 54-61, 2004.

TABLE IV. EXAMPLES OF USING AAC TO RECORD AA IN AN INDUSTRY PROJECT

ID	Name	Stakeholder	Description	Pros	Cons	Relationship(s)
1	433MHz Model with 20 Meters Distance	Wei Feng Wen (architect)	This model provides the ability of communication between readers and tags in a 433MHz frequency, which has 20 meters of the communication distance (in an open space).	<ul style="list-style-type: none"> <li>- Strong ability of fitting the environment.</li> <li>- It has a longer communication distance than 2.4GHz model.</li> </ul>	<ul style="list-style-type: none"> <li>- The power consumption and the cost are higher than 2.4GHz model.</li> </ul>	<ul style="list-style-type: none"> <li>- <i>Is an alternative to</i> 2.4GHz model</li> <li>- <i>Constrains</i> the communication between readers and tags</li> </ul>
2	STM32F107	Wei Feng Wen (architect)	This is the type of MCU of the reader. It can be replaced if necessary.	<ul style="list-style-type: none"> <li>- Compared to AT91 (another type of MCU), it has lower power dissipation, higher basic frequency and stability.</li> </ul>	<ul style="list-style-type: none"> <li>- A relatively high price.</li> </ul>	<ul style="list-style-type: none"> <li>- <i>Conflicts with</i> and <i>Is an alternative to</i> AT91</li> <li>- <i>Constrains</i> the operation of the reader</li> <li>- <i>Constrains</i> the reader code development</li> </ul>
3	Internet	Wei Feng Wen (architect)	This is the communication way between readers and computers.	<ul style="list-style-type: none"> <li>- High speed.</li> <li>- It can process macro data.</li> </ul>	<ul style="list-style-type: none"> <li>- Lower stability than serial port.</li> <li>- It needs an additional algorithm to prevent packet loss.</li> </ul>	<ul style="list-style-type: none"> <li>- <i>Comprises</i> using Internet access</li> <li>- <i>Comprises</i> using cable or WIFI</li> <li>- <i>Is an alternative to</i> serial port</li> </ul>
4	Cable	Wei Feng Wen (architect)	We use cable as the fixture but not WIFI to realize the internet.	<ul style="list-style-type: none"> <li>- Stable and long distance.</li> </ul>	<ul style="list-style-type: none"> <li>- The reader needs to have an Internet port (extra complexity).</li> <li>- The cable becomes complicated if there are many readers.</li> </ul>	<ul style="list-style-type: none"> <li>- <i>Is related to</i> Internet</li> <li>- <i>Is an alternative to</i> WIFI</li> </ul>
5	Reader Operation	Wei Feng Wen (architect)	We need an operation running in the readers.	<ul style="list-style-type: none"> <li>- High instantaneity and stability.</li> </ul>	<ul style="list-style-type: none"> <li>- This would result in some extra power and time consumption.</li> </ul>	<ul style="list-style-type: none"> <li>- <i>Is related to</i> STM32F107</li> </ul>

TABLE V. QUESTIONNAIRE AND ANSWERS

1. Do you clearly know what architectural assumptions are?

Fully understand     Somewhat understand     Do not understand

2. Do you agree that architectural assumptions need to be managed in agile development?

Yes     No     It depends

3. To what extent can you use our method to identify and record architectural assumptions in the agile project?

Easy to use     Need some effort but acceptable     Difficult to use

4. Do you agree that the efficiency in agile development can be improved by identifying and recording architectural assumptions?

Yes     No     I do not know

5. Do you agree that tools are necessary when identifying and recording architectural assumptions in agile development?

Yes     No     I do not know

6. What other activities related to architectural assumptions do you think are important in agile development (such as architectural assumptions recovery, etc.)?

Answer: understanding, recovery, and evaluation of architectural assumptions

7. What are your comments and suggestions for improving the method based on your experience on agile development?

Answer:

(1) Whether this method can be suitable and readily used in industry is related to some other factors, such as team organization, motivation of stakeholders, etc.

(2) It is hard to define what types of architectural assumptions should be shared to what stakeholders, because we cannot and it is not necessary to share all the assumption information to everyone, but we only need to share the information to the specific stakeholders who need it. For example, it is necessary to share the system response time to the customers, but it may not be necessary to share the technical assumptions with them.

# Extended DEVSML as a Model Transformation Intermediary to Make UML Diagrams Executable

Jianpeng Hu<sup>1,2</sup>, Linpeng Huang<sup>1</sup>, Bei Cao<sup>1</sup>, Xuling Chang<sup>1</sup>

<sup>1</sup>Dept. of Computer Science and Engineering  
Shanghai Jiao Tong University, Shanghai, China

<sup>2</sup>College of Electrical and Electronic Engineering,  
Shanghai University of Engineering Science, Shanghai, China  
mr@sues.edu.cn, huang-lp@sjtu.edu.cn, caobei.sjtu@gmail.com, changxl@sjtu.edu.cn

**Abstract**—The Unified Modeling Language (UML) has been widely used for software and system design. To reduce the cost and risk of the system development, it is very important to validate and evaluate the system precisely in early design phase. Many efforts were made to make UML executable by transforming single diagram to executable model such as Colored Petri Nets (CPN), however, approach like this could not provide more systematic and intuitive simulation of the entire system. Therefore we choose the Discrete Event System Specification (DEVS) as the target formalism and transform UML Diagrams to systematically simulatable models in this paper. To achieve this goal, we extend the DEVS modeling language (DEVSML) by enhancing its capability of describing complex behavior of systems, and provide an automated code generation process using Extended DEVSML (E-DEVSML) as a model transformation intermediary to help modelers to acquire the benefits of DEVS framework without delving in the DEVS theory.

**Keywords**- UML; DEVS; Model Transformation; Simulation; Executable Model

## I. INTRODUCTION

The Unified Modeling Language (UML) has become the de facto standard modeling language for software and system design. With extensions in different specific domain, such as Service oriented architecture Modeling Language (SoaML) in service engineering, the UML family has powerful ability to describe the systems from diverse viewpoints. However, systems described by those diagrams in UML are only static models, which are hard to validate and verify. Rigorous validation and verification of system specifications requires executable models, which typically come with various simulation capabilities to further support dynamic analysis and evaluation of the systems. A lot of researches are made to transform the UML models to executable discrete-event models. Colored Petri Nets (CPN) and Discrete Event System Specification (DEVS) [1] are two extensively referenced and used modeling and simulation formalisms. CPN are especially well suited to model workflows of systems where concurrent events can take place. Commonly a single UML diagram such as Sequence Diagram can be transformed into CPN to perform a validation [2]. As to DEVS which starts from general system theory, it may provide more systematic and intuitive simulation of the entire system by transforming a combination of several

diagrams to executable models. In addition, DEVS has been proven to be a universal formal mechanism to express a variety of discrete-event system subclasses, including Petri Net, Cellular Automata and Generalized Markov Chain [3]. Thus we argue that DEVS is a better choice if we need a systematic validation and evaluation of the system. In this paper our research devotes to realize seamless connection between UML and DEVS simulation by model transformation. To achieve this goal, we propose an Extended DEVSML (E-DEVSML) by enhancing its capability of describing complex behavior of systems, and provide an automated model transformation approach between UML and DEVS.

## II. PARALLEL DEVS FORMALISM

Parallel DEVS removes constraints in the classic DEVS that originated with the sequential operation and hindered the exploitation of parallelism [1]. DEVS models can fall into two categories: atomic and coupled. The atomic model is the irreducible model definition that specifies the behavior for any modeled entity. The coupled model represents the composition of several atomic and coupled models connected by explicit couplings. An atomic model  $M$  and a coupled model  $N$  are defined by the following equations:

$$M = \langle IP, OP, X, S, Y, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta \rangle \quad (1)$$

$$N = \langle IP, OP, X, Y, D, EIC, EOC, IC \rangle \quad (2)$$

In an atomic model,  $S$  is the state space;  $IP, OP$  are the set of input and output ports;  $X, Y$  are the set of Inputs/Outputs, which are basically lists of port-value pairs.

$X = \{ (p, v) \mid p \in IP, v \in X_p \}$ ,  $Y = \{ (p, v) \mid p \in OP, v \in Y_p \}$ , where  $X_p$  and  $Y_p$  are input/output values on port  $p$ .

$\delta_{int} : S \rightarrow S$  is the internal transition function;  $\delta_{ext} : Q \times X^b \rightarrow S$  is the external transition function, where  $Q = \{ (s, e) \mid s \in S, 0 \leq e \leq ta(s) \}$  is the total state set,  $e$  is the time elapsed since last transition, and  $X^b$  is a set of bags composed of elements in  $X$ ;  $\delta_{con} : S \times X^b \rightarrow S$  is the confluent transition function, which decides the order between  $\delta_{int}$  and  $\delta_{ext}$  in cases of collision between simultaneous external and internal events.

$\lambda : S \rightarrow Y^b$  is the output function, where  $Y^b$  is a set of bags composed of elements in  $Y$ ;  $ta(s) : S \rightarrow R_0^+ \cup \infty$  is the time advance function which decide how much time the system stays at the current state in the absence of external events.

This work was supported by the National Natural Science Foundation of China under Grant No. 61232007, 91118004

In a coupled model,  $IP$ ,  $OP$ ,  $X$  and  $Y$  have similar connotation as in atomic model, but mean external elements;  $D$  is a set of DEVS component models.  $EIC$  is the external input coupling relation,  $EOC$  is the external output coupling relation, and  $IC$  is the internal coupling relation. The coupled model can itself be a component of a larger coupled model giving rise to a hierarchical DEVS model construction.

### III. EXTENDED DEVS MODELING LANGUAGE

In practice, when we depict a system with previous DEVS Modeling Language (DEVSML) version [4] based on Parallel DEVS and Finite Deterministic DEVS, it is inconvenient to deal with nondeterministic state transitions, any state transition based on the message content is not realizable, and there may be a collision if two messages simultaneously arrive on more than two ports. Therefore we take appropriate measures to extend DEVSML to deal with these complex situations.

#### A. Separation of Message Processing From State Transition

Sometimes, simultaneous arrivals of inputs on different ports make it harder to describe the behavior of the external transition function, because only one transition can be invoked at the same time. If we choose one of the possible transitions to trigger, other inputs may be ignored or lost. Therefore we try to separate the processing of inputs from state transition. For example, we could first store these inputs in some variables before state transitions. From a functional perspective, an abstract description of an atomic model will be: accepting inputs or incentives and then generating an output with state changes. This behavior can be expressed with the two following formula:

$$y = \lambda(s_n) / s_n \in S, y \in Y^b \text{ and } s_n = \delta(x, s_{n-1}) / x \in X^b \cup \emptyset \quad (3)$$

where  $S = \{s_0, s_1, \dots, s_{n-1}, s_n\}$  is set of states. We could divide this input/output process into three parts: receiving a message, data processing during a series of transitions and sending a message. Three associate functions are defined as following:

$$v_x = Rev(x), y = Sed(v_y), v_y = DP(v_x) \quad (4)$$

where  $x \in X^b$ ,  $y \in Y^b$ ,  $v_x \in V_x$ ,  $v_y \in V_y$ , we define a set of input-port-associate variables  $V_x$  and a set of output-port-associate variables  $V_y$ . These variables may be simple data types (e.g. integer, floating number and string) or containers of simple data types (e.g. queue, stack and list). Deriving from equations 3 and 4, we can get equation 5:

$$y = \lambda(s_n, v_y) \text{ and } (s_n, v_y) = \delta(v_x, s_{n-1}) \quad (5)$$

When an external transition is triggered, the receiving function  $Rev$  stores the inputs into corresponding  $V_x$  by unpacking a message bag. And  $DP$  function for data processing from  $V_x$  to  $V_y$  could be implemented within external transitions or internal transitions. Similarly, the sending function  $Sed$  for processing of outputs may be independently implemented in output function  $\lambda$ . This separation saves modelers from digging into the DEVS semantics but only their own behavior description of the systems.

#### B. Support for Nondeterministic State Transitions

First, we'll show the differences between Nondeterministic State Transitions (NST) and Deterministic State Transitions

(DST) in formal definition. Based on basic transition functions, DST and NST can be defined as follow:

$$DST = \delta: S \times X^b \rightarrow S \quad (6)$$

$$NST = \delta: S \times X^b \rightarrow P(S) \quad (7)$$

where  $\delta(s, x) = \delta_{ext}(s, x) \cup \delta_{con}(s, x)$ , and  $\delta(s, \emptyset) = \delta_{int}(s)$ ;  $P(S)$  denotes the power set of  $S$ . In DST, the transition function  $\delta$  is a surjection. Thus the term deterministic means that there is at most one invoked transition to a target state from any source state with a specific input. On the contrary, NST has more alternatives for target state with the same inputs. In practice, it is very important to convert easier-to-implement NSTs into more efficiently executable DSTs. To achieve this goal, we defined a Conditional Transition (CT):

$$CT = \delta: S \times X^b \times C \rightarrow S \quad (8)$$

It shows the transition is triggered only when some pre-conditions are satisfied. It also can be described as  $\delta(s, x, c) = s'$ , where  $s, s' \in S$ ,  $c \in C$ ,  $x \in X^b \cup \emptyset$ ,  $C$  is a set of constraints. We could make an assumption that, for any target state we can find a disjoint pre-condition satisfied (may be specific inputs, variable correlation, time correlation, etc.) to invoke a deterministic transition from a source state, because we need ensure the system running as expected and there must be only one option of transitions in the same condition. So a NST described as:  $\delta(s, x) = \{s_1, s_2, s_3, \dots, s_i / s_i \in S\}$ , may be divided into equations as follow:  $\delta(s, x, c_1) = s_1$ ,  $\delta(s, x, c_2) = s_2, \dots$ ,  $\delta(s, x, c_i) = s_i$ . We assume that a specific input can also be viewed as a part of the constraint condition, while for internal transitions, the input is null. For unified implementation of internal transitions and external transitions, we simplify these equations:

$$\delta(s, c_1) = s_1, \delta(s, c_2) = s_2, \dots, \delta(s, c_i) = s_i \quad (9)$$

where  $c_i \in C$ ,  $s, s_i \in S$ . We could consider equation 9 as an approximate realization of a NST using a DST format.

#### C. Abstract Syntax of E-DEVSML

E-DEVSML is specified using Extended Backus-Naur Form (EBNF) notation like the previous version. Considering of both convenience of use and conformity with the DEVS formalism, we specify it with modular and object-oriented features. We realize this language by Eclipse Xtext which is a powerful framework for the development of a DSL. Models in E-DEVSML are divided into three primary elements: the *Entity*, the *Atomic* and the *Coupled*.

1) *Entity*: Message objects are exchanged according to the port-value pairs, and the datatype of a input value can be defined as an entity and reused by some ports. Figure 1 gives the definition of *Entity* in EBNF. We define a variable type named *container* which is a common data structure (e.g. queue) to store a series of entities. The keyword *default* assign values to variables when model is started or restarted.

2) *Atomic*: The Atomic is the most important and complicated part of DEVS. The Atomic model is specified in EBNF grammar as figure 2 and figure 3 showing. The keyword *vars* defines a set of variables. The *interfaceIO* specification gives the definition of ports with specific data

type which is referenced as an *Entity* type. And *state-time-advance* defines a set of states and the associated time-advances. The time-advance *TimeAdv* can have values of either *DOUBLE*, *infinity* or a *Variable* already declared. The *state-machine* contains the initial state *InitState* and four behavioral functions. The expression *(code=Code)?* implies that there may be a code block associated with setting up of the initial state.

```
Entity : 'entity' name=ID
('extends' superType = [Entity | QualifiedName])?
('{ (attributes += Variable)* '}' )? ;
Variable :
type = VarType name=ID ('default' deft=STRING)?;
VarType:
simple = ('int'|'double'|'String'|'boolean')|
container = ('queue'|'stack'|'list'|)
complex = [Entity | QualifiedName];
```

Figure 1. Definition of Entity in EBNF

```
Atomic : 'atomic' name=ID
('extends' superType=[Atomic | QualifiedName])?
('{ 'vars' '{(variables += Variable)*}'
'interfaceIO' '{(msgs += Port)*}'
'state-time-advance' '{(stas += STA)*}'
'state-machine' '{'start in ' init=InitState
(stm1=Delttext)? (stm2=Outfn)? (stm3=Deltint)?
(stm4=Confluent)?}' (func += UserFunction)* '}' );
Port : type = ('input'|'output')
ref=[Entity | QualifiedName] name=ID;
STA : name=ID timeAdv=TimeAdv;
TimeAdv: tav=DOUBLE|inf='infinity'|tvar=[Variable];
InitState: state=[STA ] (code=Code)?;
CodeTemplate: 'code-template' name=ID
'(' (codePara += Parameter)* ')'
'{' (templateBody += TemplateBody)+ '}' ;
TemplateBody: code=Code | para=[Parameter];
UseTemplate: 'use' ct=[CodeTemplate]
'['(para+=[Variable|QualifiedName])* ']' ;
UserFunction: code=Code ;
Parameter: str=STRING;
Code: '{' str=STRING '}' ;
```

Figure 2. Definition of Atomic in EBNF

```
Delttext: 'delttext' '{(rm += ReceiveMessage)+
(st += StateTransition)*}' ;
Deltint: 'deltint' '{(st += StateTransition)+}' ;
Outfn: 'Outfn' '{(so += StateOutput)+ '}' ;
Confluent: 'confluent' con=( 'ignore-input'|
'input-only'| 'input-first'| 'input-later' ) ;
SendMessage: 'send' '{'out=[Port]'}'sender=[Variable];
ReceiveMessage:
'receive' '{'in=[Port]('subMsg = QualifiedName)?}'
receiver = [Variable];
StateOutput: 'S:'state=[STA] '{(sm += SendMessage)*}' ;
StateTransition: 'S:' state=[STA]
'{'(ct += ConditionalTransition)+ '}' ;
ConditionalTransition:
('when' '{( condition=STRING ')}')?
'{'(dp += DataProcess)* act = Action '}' ;
Action: con='continue'|sig = SetSigma|ts=Transition;
SetSigma:
'sigma' '{(sdv= DOUBLE|inf='infinity'|v=[Variable])}' ;
Transition: 'goto' target=[STA] (sig=SetSigma)?;
DataProcess: code=Code | ut = UseTemplate;
```

Figure 3. Definition of behavior of an Atomic in EBNF

DEVS has four functions to specify the behavior of an atomic model which are the main featured parts of E-DEVSML: *Delttext*, *Deltint*, *Outfn* and *Confluent*. And the conditional transition, receiving function, sending function and DP function mentioned above are also implemented as shown in figure 3. The keyword *continue* allows the model to stay in the same source state and the *SetSigma* rule allows the resetting of time-advance. To enable interoperability between the coupled

model and its components, user-defined method defined by *UserFunction* in an atomic model is allowed to be called by itself or the coupled model comprising it. In addition, *UseTemplate* rule can reuse a code block with parameters which is already defined in *CodeTemplate* rule.

3) *Coupled*: The specification of a coupled model is shown in figure 4. To put it simply, the coupled model has the same interface specification as the atomic model. It is composed of a set of models which can be either atomic component or coupled component.

```
Coupled:
'coupled' name=ID
('extends' superType=[Coupled|QualifiedName])?
'{' 'vars' '{(variables += Variable)*}'
'models' '{(components += Component)* '}'
'interfaceIO' '{(ports += Port)*}'
'couplings' '{(couplings += Coupling)*}'
('user-code' userCode=Code)? '}' ;
Component: AtomicComp | CoupledComp;
AtomicComp:
'atomic' at=[Atomic | QualifiedName] name=ID ;
CoupledComp :
'coupled' cp=[Coupled | QualifiedName] name=ID;
Coupling : ic=IC | eoc=EOC | eic=EIC;
EIC : 'eic' 'this': srcport = [Port ] '->'
dest = [Component] ':' destport = [Port];
IC: 'ic' src = [Component | QualifiedName] ':'
srcport = [Port] '->'
dest= [Component | QualifiedName] ':'
destport = [Port];
EOC : 'eoc'
src = [ Component ] ':' srcport = [Port ] '->'
'this': destport = [Port];
```

Figure 4. Definition of Coupled in EBNF

#### IV. MODEL TRANSFORMATION USING E-DEVSML

This automated transformation process from UML to DEVS is a two-step approach. Due to the textual style of E-DEVSML, first we use Xpand, a language specialized on code generation, to implement transformation from UML to E-DEVSML. Then we can get executable codes through Xtext framework because Xtext is seamlessly integrated with the Eclipse Java framework by code generator with Xtend. At last, we use one of the open source DEVS simulator named DEVS-Suite [5] to validate the executable models.

First, all the data types inherit from UML::Class can be translated into Entities in E-DEVSML. Second, from structural perspective, Class Diagram gives definition of attributes in an atomic model, while Component Diagram provides structural information about a coupled model including components in coupled model and couplings between them. At last, from behavioral perspective, State Machine Diagram (SMD) is quite suitable for providing behavioral information of an atomic model. In this diagram, the guard conditions of transitions certainly are mapped onto the constraints of Conditional Transitions. The receiving function and DP function are derived from transition effects, and the sending function is derived from state exit effects. Note that, there is no concept can be mapped onto  $\delta_{con}$ . The default definition of confluent transition function simply applies  $\delta_{int}$  before  $\delta_{ext}$  in cases of collision between simultaneous external and internal events.

To illustrate the capability of the E-DEVSML, we are going to show an example of a multi-server architecture model “m”

as shown in figure 6-a. The coupled model “EFM” contains two models: the coupled model “ExpFrame” is an experimental frame for evaluating the performance of the other coupled model “m”. The generator “g” sends entity “job” at a periodic rate, and the transducer “t” keeps counting generated jobs as well as processed jobs. In the coupled model “m”, the coordinator “multiSco:serverCord” keeps track of the status of the servers in a queue called “freeServers”. Figure 5 shows a SMD of the coordinator “serverCord”: when a job arrives at the input port “in”, it is routed to the first “passive” server. If no server is free, the job will be stored in a queue “jobsToDo”. When a completed job returns on corresponding port “x”, the “serverCord” reroutes it to the output port “out.” Simultaneously arrived completed jobs will be stored in a queue “jobsDone”. Note that, “jobsToDo” and “freeServers” are input-port-associate variables, and “jobToDo” is an output-port-associate variable which is assigned in the DP function employing a code template “make\_output”, while “jobsDone” is not only a  $V_x$  but also a  $V_y$ . If we try to transform this diagram to the previous version of DEVSMML, it’s difficult to deal with the NST to different target states which are triggered by a message on the same port or simultaneous messages on different ports, while the E-DEVSMML solves this problem and makes the automated transformation process more smoothly.

After an automated transformation process from UML to DEVS, figure 6-c shows a comparison of the model “serverCord” in E-DEVSMML and java. When the final generated codes are executed in DEVS-Suite, we can get runtime animations shown in “simView” window (figure 6-a). After several simulations with different processing time for a job of single server, the statistics of system performance is shown in figure 6-b. All these features guarantee validation of the functional requirements and evaluation of nonfunctional requirements precisely in earlier design phase.

### V. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a DEVS based modeling language E-DEVSMML which can helps modelers to realize systematic simulation of the systems. We also presented an

approach to make UML diagrams executable through an automated model transformation process using E-DEVSMML. It also saves modelers from digging into the DEVS simulator codes but only their own behavior description of systems. In the future, practical application of E-DEVSMML in an industry case will be discussed. As the current status of DEVSMML is only support for static-architecture coupled models, our future work also includes description of DEVS dynamic-architecture, which permits the coupled model to evolve over time.

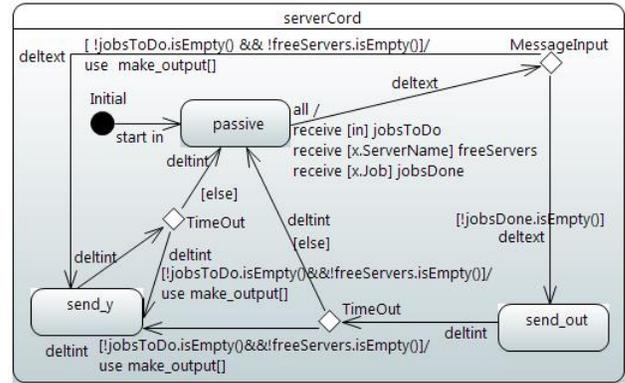


Figure 5. State Machine Diagram of the coordinator “serverCord”

### REFERENCES

- [1] Zeigler, B. P., T. G. Kim, and H. Praehofer. Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems Second Edition: Academic Press. 2000.
- [2] Wang, Renzhong, and Cihan H. Dagli. Executable system architecting using systems modeling language in conjunction with colored Petri nets in a model - driven systems development process. Systems Engineering 14.4 (2011): 383-409.
- [3] Vangheluwe, Hans LM. DEVS as a common denominator for multi-formalism hybrid systems modelling. Computer-Aided Control System Design, 2000. CACSD 2000. IEEE International Symposium on.
- [4] Mittal, S., and S. A. Douglass. DEVSMML 2.0: The Language and the Stack. Symposium on Theory of Modeling and Simulation, Spring Simulation Multiconference. Orlando, FL: SCS. 2012.
- [5] The Arizona Center for Integrative Modeling and Simulation (ACIMS), DEVS-Suite, available at <http://sourceforge.net/projects/devs-suitesim/>.

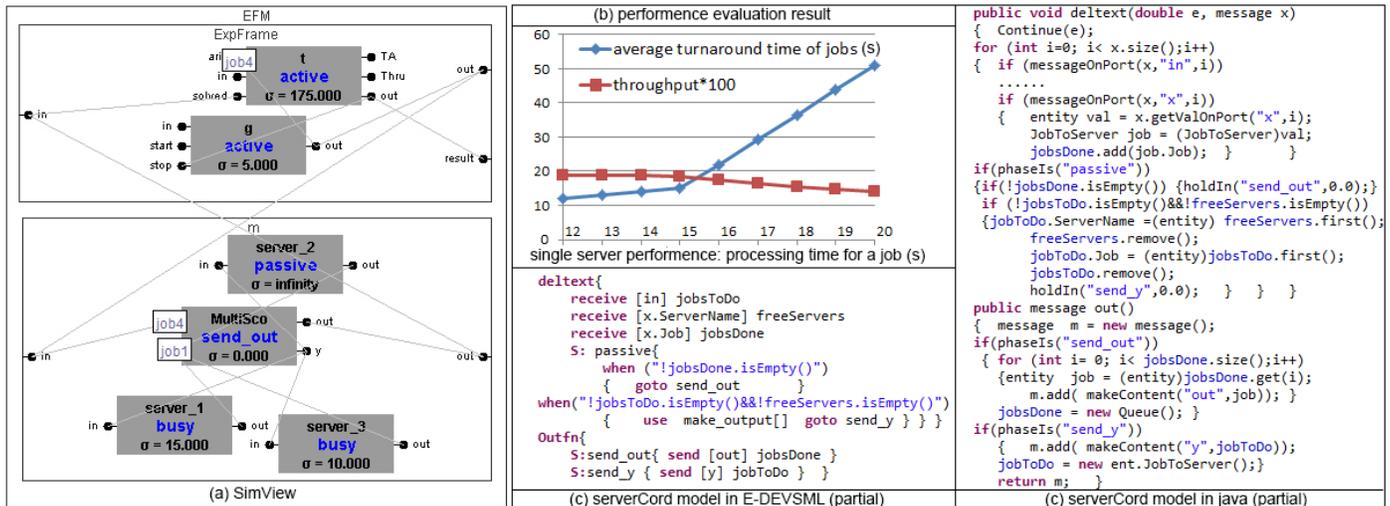


Figure 6. Simulation results of the multi-server architecture and some corresponding codes

# Detecting Semantic Equivalence in UML Class Diagrams

Valéria Oliveira Costa<sup>1,2</sup>

<sup>1</sup>Instituto Federal de Educação, Ciência e Tecnologia do  
Piauí, IFPI  
Teresina, Brasil  
valeria@ifpi.edu.br,

Rodrigo S. Monteiro<sup>2</sup>, Leonardo G. P. Murta<sup>2</sup>

Instituto de Computação<sup>2</sup>  
Universidade Federal Fluminense, UFF  
Niterói, Brasil  
{salvador, leomurta}@ic.uff.br

**Abstract**— When developing a system in Model-driven Engineering (MDE), developers change the same diagram in parallel. These changes generate different versions that may conflict. Conflicts can be syntactic, related to the structure of the diagram, or semantic, related to the meaning of the diagram. The detection of semantic conflicts in diagrams should take into consideration both the syntax and semantics. This is necessary because languages like UML provide different representations that produce the same meaning. Therefore, syntactically different diagrams may be semantically equivalent. Thus, conflict detection methods must understand the semantics of diagrams to reduce the occurrence of false positive and false negative conflicts. This paper presents a semantic conflict detection method for UML class diagrams. It verifies if two versions of a class diagram are semantically equivalent, if one version semantically contains the other, or if they are semantically conflicting.

**Keywords**— component; model version control; semantic equivalence; semantic conflict detection; UML class diagram.

## I. INTRODUCTION

The UML aims at assisting developers by providing tools to analyze, design, and implement systems [1]. One of the key diagrams in UML is the class diagram. A class diagram provides a static view of a system and consists of classes and relationships (association, aggregation, generalization, realization, etc.). According to [2], class diagrams are widely used for modeling systems and have several different structures with the same meaning. For example, we can represent a relationship amongst classes by means of an association or an attribute, setting its type as the opposing class, or even mixing both representations [3]. Therefore, the choice of which representation to use is the responsibility of the modeler [2]. Thus, different modelers can produce different class diagrams that have the same meaning.

When the development team uses Model-driven Engineering (MDE), the development of a system is done through the creation, modification, and evolution of models [4]. In this context, a model can contain or be restricted to a class diagram. During the system development, different versions of diagrams are generated in parallel by developers. This occurs for different reasons that range from time-to-market to isolation

of evolutive and corrective maintenance. These versions need to be controlled to ensure consistency. In this scenario arises the need of Model-driven Version Control System (VCS). A Model-driven VCS stores the versions of models and provide support for model comparison, detection and resolution of conflicts, and merging. Among the existings Model-driven VCS we can mention Odyssey-SCM [5], AMOR [6], and Mirador [7].

Model-driven VCS must check for the occurrence of conflicts amongst versions. Conflict is a set of contradictory changes where at least one operation performed by a developer does not comply with at least one operation performed by another developer [8]. A good conflict detection method should minimize the occurrence of false positive and false negative conflicts. The former are non-conflicting changes marked by the detection method as conflicts [9]. The latter are conflicting changes not marked by the detection method as conflicts [9]. False positives conflicts reduce the productivity of developers since the time taken to analyze them could be used for other tasks. False negatives conflicts produce even more harmful consequences, as they may be ignored by the development team, leading to inappropriate merges.

When the conflict detection method considers only the syntax of diagrams, it is restricted to the detection of syntactic conflicts. These conflicts are detected by structural comparison between the versions of diagrams [10][11]. As the method does not recognize the semantics of diagram, equivalent representations can be diagnosed as conflicting (false positives conflicts). Understanding the semantics of the diagram allows the identification of different representations that have the same meaning. Thus, the method becomes able to ascertain whether syntactically different versions are semantically equivalent. This helps in reducing false positives conflicts.

Moreover, understanding the semantics of diagram allows the detection of semantic conflicts. A semantic conflict occurs when changes performed in parallel not only interfere with the modified element, but also in others. For example, semantic conflicts may occur when a developer modifies an element that depends on another element modified by other developers [10]. As semantic conflicts are more difficult to detect, they can generate false negative conflicts. Therefore, understanding the

semantics of the diagrams can also help reducing false negative conflicts.

This paper presents a semantic equivalence detection method for class diagrams, called Rainbow, to solve the aforementioned problems. Our method investigates the semantic equivalence of class diagrams in order to reduce false positive and false negative conflicts. It receives an original version and two revisions of it. These three versions are exported as Prolog facts and enriched with semantic rules. Afterwards, the differences between the original version and its revisions are computed. These differences consist of the items added or deleted in a revision from original version. The differences are checked for semantic equivalence and semantic conflicts are detected. In a previous work [12], we present our semantic conflicts detection method for UML use case diagram. In this paper, we extended the method for UML class diagram.

The remaining of the paper is organized as follows: section II presents important concepts about Model-driven VCS; Section III explains how the Rainbow works; Section IV shows an example of Rainbow in action; Section V discusses the technologies used to implement a tool that automates the method; Section VI presents some related works; and Section VII presents the conclusion and future work.

## II. BACKGROUND

Rainbow assumes a development environment that uses MDE and an optimistic Model-based VCS. As we want to add no extra burden to the development environment, we also assume that the VCS is loosely coupled to it. This way we focus on a three-way state-based version control [13].

In three-way state-based version control, the merge algorithm receives three versions: the version that the developer wants to put in the repository (*developer*), the latest version stored in the repository (*current*), and the common ancestor version from which both derived (*base*). In this approach, in general, developers change the model on their workspace, independently and in parallel, until they decide to socialize their contributions [13]. When the developer takes this decision, he or she sends their *developer* version to the VCS. Then, the VCS checks for conflicts with the *current* version and generates a *merged* version.

The complete process of the three-way merge has four phases [14]: comparison, conflict detection, conflict resolution, and merge. The **comparison phase** contrasts the *developer* and *current* versions in order to calculate their differences. The information contained in *base* version is taken into consideration to help inferring the operations performed in each version. This phase is critical to the final result of the merge process because the other phases are based on information collected by it. In state-based merge, the comparison algorithm relies only on the input versions (*base*, *current*, and *developer*) to compute the differences between versions. Therefore, the method requires matching techniques to check if an element corresponds to another in different versions.

Next, the **conflict detection phase** verifies whether the differences contain conflicts. There are various types of conflicts such as syntactic and semantic conflicts. This work

focuses on the detection of semantic conflicts. During the analysis of differences, if *developer* and *current* versions are equal or semantically equivalent then there are no conflicts. In this case, the **merge phase** can be performed. Otherwise, the existing conflicts must be resolved (**conflict resolution phase**) before continuing the merge process. After resolving the conflicts, the merge phase can be performed. At this phase, the *base* and *developer* versions are combined into a single version (*merged* version).

## III. RAINBOW

Rainbow considers a subset of relationships and elements that exist in the specification of the UML class diagram (CD) [1]. This subset contains the following elements: class, attribute, operation, multiplicity, role, association, aggregation, composition, interface, realization, generalization, enumeration, and abstract class.

Rainbow is composed of three phases as shown in Fig. 1 [12]. These phases are: translation, semantic enrichment, and conflict detection. In the **translation phase**, our method receives the three versions of CD to be analyzed (*base*, *current*, and *developer* versions) and transforms them into a set of Prolog facts. Each Prolog fact represents an element or relationship in a CD version. Tab. I shows some examples of Prolog facts generated in this phase.

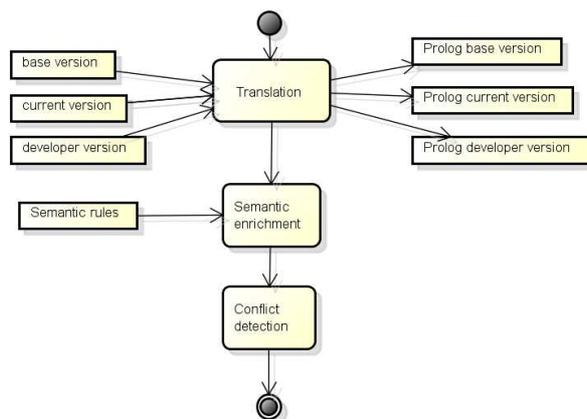


Figure 1. Rainbow phases

TABLE I. MAPPING FROM CLASS DIAGRAM TO-PROLOG FACTS

Diagram class item	Prolog fact
Class	class(className).
Abstract class	isAbstractClass(className).
Generalization	inheritance(subclassName, superclassName).
Attribute of class	attribute(className, attributeName).

During the **semantic enrichment phase**, Rainbow analyzes each Prolog fact generated during translation phase to infer indirect relationships. This inference generates new Prolog facts that are added to those created in the previous phase. To accomplish this, the class diagram specification has been

analyzed and we generated a set of rules that describes the semantics of the class diagram according to the UML metamodel. Tab. II shows some specific semantic rules for the UML class diagram used by Rainbow.

Next, the **conflict detection** phase starts. Rainbow compares the enriched set of Prolog facts to identify if they are semantically equivalent or if one version semantically contains the other. If one of these two alternatives is true, it means that there are no semantic conflicts. If the first alternative is true, it means that any of the versions can be chosen. If the second alternative is true, then the method should choose the most complete version. Otherwise, the versions should be analyzed for occurrences of semantic conflicts.

The semantic conflict detection consists of executing two diff operations. The first is performed between *base* and *developer* versions while the second is performed between *base* and *current* versions. Each diff operation computes the additions and deletions that transformed the *base* version in the version in comparison. The additions form the *add* set and the deletions form the *del* set.

TABLE II. SEMANTIC RULES FOR UML CLASS DIAGRAM

#	Rule
1	In a generalization relationship, for each attribute in the superclass, an equivalent attribute is inferred for the subclass unless the attribute in question has a private visibility.
2	In a generalization relationship, for each operation in the superclass, an equivalent operation is inferred for the subclass unless the operation in question has a private visibility.
3	In a generalization relationship, for each association in the superclass, an equivalent association is inferred for the subclass.
4	In a realization relationship, for each attribute of the interface, an equivalent attribute is inferred for the class.
5	In a realization relationship, for each operation of the interface, an equivalent operation is inferred for the class.
6	If a class is abstract and all its methods are abstract then, if there is an interface that has the same name and the same elements, they are semantically equivalent.

Next, the *add* and *del* sets of each diff operation are checked for conflicts. A conflict occurs if a model element appears simultaneously in the set of additions of the first diff operation and in the set of deletions of the second diff operation, or vice versa. For a better understanding of the functioning of Rainbow, section IV presents an example of our method in action.

#### IV. RAINBOW IN ACTION

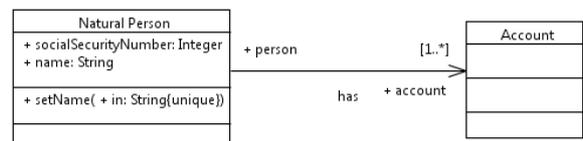
Consider the development of a system for bank control, and the modeling of their CD. Fig. 2 shows three simplified versions for such CD. Fig. 2.a shows the *base* version, Fig. 2.b presents the *current* version, and Fig. 2.c shows the *developer* version. In the proposed method, all three illustrated versions have their files submitted to the translation phase as explained in Section III. After the execution of this phase, three Prolog sets are generated, representing all relevant information about the input versions. During this phase, only the syntax of CD is

considered. Due to space constraints, this paper does not show all the generated facts during the translation phase.

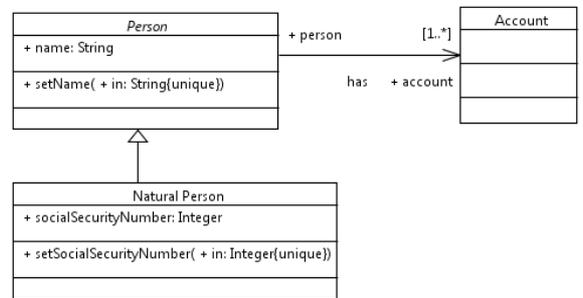
Next, the semantic enrichment phase is performed. In this phase the Prolog facts are enriched with semantic rules specific for UML CD. These rules infer indirect relationships through direct relationships amongst CD elements, conciliating the semantics of different representations.

In order to illustrate the semantic enrichment performed by the semantic rules, consider the figures that compose Fig. 2, more specifically Fig. 2.a and 2.b. Analyzing the differences between these two figures, we can observe that the developer made the following changes: he or she created an abstract class, named *Person*, and added a generalization between *Natural Person* and *Person* classes. He or she chose to put the attribute name and operation setName in the superclass. Another change made in this version was the addition of the association, named *has*, between *Person* and *Account* classes.

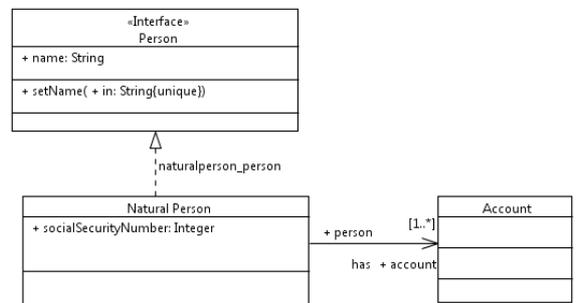
Furthermore, he or she added setSocialSecurityNumber operation in *Natural Person* class.



a. base version



b. current version



c. developer version

Figure 2. Model Versions for Bank Control System.

With respect to Fig. 2.b, the generalization between Natural Person and Person classes determines that Person is the superclass of Natural Person. Due to that, Natural Person inherits all attributes and operations that exist in Person whose visibility is not private. The same is also true for the associations.

These details justify the creation of new facts that represent these implicit behaviors (semantics) of the UML CD. Thus, in addition to the syntactic facts created in the translation phase, new semantic facts are created for both *current* and *developer* version from semantic rules shown in Tab. II.

Regarding the *current* version, by applying rule 1, the name attribute is inferred to Natural Person class and now this class has two attributes: name and socialSecurityNumber. Furthermore, by applying rule 2, the operation setName is inferred to Natural Person class and now this class has two operations: setName and setSocialSecurityNumber. Through the application of rule 3, associations are also inherited from Person. Then Natural Person is associated with class Account.

Consider now the figures 2.a and 2.c. Fig. 2.c was created from Fig. 2.a. Checking for changes made by the developer, it can be noticed that he or she created the Person interface. In Object Oriented paradigm, interfaces are public and cannot be instantiated. These structures specify contracts. Thus, the class that implements an interface contains all elements contained in

that interface. In UML, the interface implementation is represented by a realization relationship.

Checking Fig 2.c, it is possible to note that the developer created a realization relationship between the Natural Person class and the Person interface. This relationship indicates that the Natural Person undertakes to implement the Person interface. When there is a relationship of realization between a class and an interface, the class in question contains not only the attributes specified in the class itself, but also the attributes and operations contained in the implemented interface.

These changes lead to the inference of new Prolog facts from UML semantic rules shown in Tab. II. Once again, it is worth mentioning that these new facts are added to the syntactic facts created in translation phase. In this case, these new facts were appended to the *developer* version by applying the semantic rules 4 and 5 of Tab. II. Through the application of rule 4, the Natural Person class has facts that correspond to name and socialSecurityNumber attributes. The name attribute comes from the Person interface. By rule 5, the setName operation also is inferred to Natural Person class.

After the end of the semantic enrichment phase, Rainbow starts the conflict detection phase. Analyzing this scenario, we can observe that two developers changed in parallel the same class diagram. Then, we need to check if these changes generated equivalent or conflicting versions.

TABLE III. ADD SETS FOR CURRENT AND DEVELOPER VERSIONS

Add <sub>Current</sub>	Add <sub>Developer</sub>
class(person).	
visibilityclass(person,public).	
isabstractclass(person).	interface(person).
inheritance(natural_person,person).	realization(naturalperson_person,natural_person,person).
attribute(person,name).	attributeinterface(person,name).
typeattribute(person,name,string).	typeattributeinterface(person,name,string).
visibilityattribute(person,name,public).	visibilityattributeinterface(person,name,public).
lowervalueattribute(person,name,1).	lowervalueattributeinterface(person,name,1).
uppervalueattribute(person,name,1).	uppervalueattributeinterface(person,name,1).
operation(person,setname).	operationinterface(person,setname).
visibilityoperation(person,setname,public).	visibilityoperationinterface(person,setname,public).
typereturn(person,setname,unset).	typereturninterface(person,setname,unset).
isabstractoperation(person,setname).	isabstractoperationinterface(person,setname).
operationparameter(person,setname,name,string).	operationparameterinterface(person,setname,name,string).
operation(natural_person,setsocialsecuritynumber ).	
visibilityoperation(natural_person, setsocialsecuritynumber, public).	
typereturn(natural_person, setsocialsecuritynumber,unset).	
operationparameter(natural_person, setsocialsecuritynumber, socialsecuritynumber, integer).	
association(has,person,account).	
roleassociation_end_a(has,person,person).	
aggregationassociation_end_a(has,person,none).	
lowervalueassociation_end_a(has,person,1).	
uppervalueassociation_end_a(has,person,1).	
roleassociation_end_b(has,account,account).	
aggregationassociation_end_b(has,account,none).	
lowervalueassociation_end_b(has,account,1).	
uppervalueassociation_end_b(has,account,-1).	
isnavigableassociation_end_b(has,account).	

In this phase, Rainbow executes the diff operations in the two pairs: *base* and *current* versions, and *base* and *developer* versions. For each pair, the *add* and *del* sets are computed. First, we consider the first pair (*base-current* diffs). As previously mentioned, the *current* version has only additions. Therefore, the *del* set is empty. Tab. III shows the result of this diff operation and contains all differences found in this scenario. The  $Add_{current}$  column presents the result of the diff operation that calculates the additions made in the *current* version in relation to *base* version.

Considering the second pair formed by *base* and *developer* versions, it is possible to notice that, as in the first pair, the developer only added elements to the *base* version. Therefore, the *del* set is also empty. The *add* set is shown in Tab. III in the  $Add_{Developer}$  column.

After the creation of the *add* sets, the differences reported must be analyzed. In summary, one can say that the *current* version:

- a) Created Person class, which is an abstract and public class. This class has an attribute called name and an abstract operation called setName. The subset of facts featured in Tab III, shows the specification of these concepts.
- b) Added a generalization between Person and Natural Person classes. In this relationship Person is superclass of Natural Person.
- c) Added an operation called setSocialSecurityNumber and an association between Person and Account class.

On the other hand, one can say that the *developer* version:

- d) Added an interface called Person. This interface has an attribute called name and a setName operation. The setName operation is public and abstract.
- e) Added a realization between the Person interface and Natural Person class.

Now, consider the Prolog facts represented in Table III. In  $Add_{current}$  column, the highlighted facts correspond to the a) and b) differences mentioned above. Likewise, in  $Add_{Developer}$  column, the highlighted Prolog facts correspond to the d) and e) differences found in the *developer* version.

By rule 6 of Tab. II, it is concluded that Person interface and Person abstract class are equivalent since all methods of this class are abstract and public. Moreover, the name attribute has the same characteristics in both the interface and abstract class. The same happens with the setName operation. Thus, the Prolog facts that match the Person abstract class and all its elements, as well as those related to the Person interface, should be removed from both sets of difference. This removal operation makes the  $Add_{Developer}$  set empty.

From this information, one can conclude that the *current* version includes all semantic which exists in the *developer* version. Therefore, Rainbow chooses the *current* version because this version is the most complete.

This type of semantic equivalence is not detected when only the syntax of the CD is analyzed. When only the syntax of the CD is considered, the method can identify these differences as conflicts and generate false positive conflicts. Also, not taking into account the semantic equivalence can lead to erroneous merged version.

As shown in the example, Rainbow contributes to identify syntactically different CD versions presenting the same semantics. This helps to reduce the amount of false positive

conflicts and increases the efficiency of the method of conflict detection. This feature further reduces the rework generated for the team.

Moreover, understanding the semantics allows for detecting semantic conflicts. This type of conflict cannot be detected when only the syntax is considered. Thus, this detection can help reducing false negative conflicts. This also prevents erroneous merged version.

## V. PROTOTYPE IMPLEMENTATION

We are implementing a tool to automate the Rainbow method. It currently supports the translation phase and we are implementing the semantic enrichment phase. When completed, Rainbow implementation will be expanded to other UML diagrams and, in the future, we will check for conflicts amongst different diagrams such as class diagrams and use cases.

The example of Section IV had its CD versions designed using Papyrus<sup>a</sup>. For each CD version, Papyrus generates a file containing its XML Metadata Interchange (XMI) file. This file is used as input for Rainbow and translated into Prolog facts. We adopted the OMG Model to Text (M2T) standard to transform XMI files into a set of Prolog facts. The translation is made by means of Acceleo<sup>b</sup>. Finally, the semantic enrichment phase uses TuProlog<sup>c</sup> library, integrated to Java, to infer new Prolog facts.

## VI. RELATED WORK

Odyssey-VCS [15] is a Model-driven VCS that allows the use of fine granularity for UML 2 models. The system uses Three-Way merge and state-based versioning. The conflict detection uses existence analysis of elements and processing of attributes and relationships. This VCS takes into account only the syntax of UML models. As it does not take into account the semantics of the models, it cannot detect semantic equivalence and semantic conflicts.

In [16] is proposed a domain-specific language that defines and manages syntactic and semantic conflicts. In this approach, the differences between the compared versions are represented in models of difference. The difference metamodel is generated from the base metamodel. For each metaclass of the base metamodel are generated three new classes: AddedMC, DeleteMC and ChangeMC. These artifacts aim at representing the added, deleted and modified classes of a version. It is applied the same reasoning for attributes, associations, methods and parameters. Moreover, the approach uses criteria resolution written in OCL (Object Constraint Language) and rules. This approach detects and resolves previously known semantic conflicts. However, we are unable to identify how semantic equivalence is treated.

Smoover [17], presents a method of semantic conflicts detection. Smoover is based on semantic views and inspection strategies of elements. The latter can be configured by the user. A semantic view maps a version from a source metamodel to target metamodel. This mapping is based on

<sup>a</sup> <http://www.eclipse.org/papyrus/>

<sup>b</sup> <http://www.eclipse.org/acceleo/>

<sup>c</sup> <http://tuprolog.alice.unibo.it/>

relevant aspects of the source. The result of the transformation is a model in conformance with the target metamodel that contains the aspects of interest. The conflicts found in the source metamodel are syntactic conflicts, and those found in the mapped metamodel are semantic conflicts.

CDDiff [18] is a semantic diff operator created for UML CD. The operator receives two versions and executes its semantic comparison. The result of this operation is a set of diff witnesses. Each diff witness is an object model that is possible in the first version and is not possible in the second version. This operator uses a Two-Way merge, therefore does not detect semantic conflicts.

## VII. CONCLUSION

This paper presents a semantic equivalence detection method for class diagrams called Rainbow. This method uses the three-way merge. Thus, it receives three versions and verifies if they are equivalent, if one version contains the other version, and if there are conflicts to be resolved.

Each submitted version is transformed into Prolog facts. These Prolog facts are semantically enriched by means of UML specific rules. Next, conflict detection phase checks for occurrences of semantic conflicts.

We also present an example that shows that syntactically different versions can be semantically equivalent. Due to the difficulty in identifying this type of equivalence, the method helps on reducing the amount of false positives conflicts. Thus, it increases the efficacy of the conflict detection method as a whole.

Moreover, understanding the semantics allows for detecting semantic conflicts. This type of conflict can not be detected when only the syntax is considered. Thus, this detection can help to reduce false negatives conflicts. This prevents that false negatives conflicts are not resolved which can lead to an erroneous merged version.

As future work, we intend to support additional UML diagrams (beyond the CD and use cases diagrams). Furthermore, we intend to expand the method to work in detecting conflicts between different diagrams. Finally, we are planning to run some experimental studies with the proposed method. We also intend to perform experiments on real case studies. Another future work is to allow the visualization of conflicts, changes, and merge by the developer.

## ACKNOWLEDGMENT

We would like to thank Rhaylson Silva do Nascimento for his valuable contribution during the implementation phase of this work.

## VIII. REFERENCES

- [1] O. M. G. Specification, "A UML Profile for MARTE, Beta 1," *Object Management Group pct/07-08-04*, 2007.
- [2] H. C. Purchase, L. Colpoys, M. McGill, D. Carrington, and C. Britton, "UML class diagram syntax: an empirical study of comprehension," in *Proceedings of the 2001 Asia-Pacific symposium on Information visualisation-Volume 9*, 2001, pp. 113–120.
- [3] C. Larman, *Utilizando UML e Padrões*. Bookman, 2007.
- [4] A. Cicchetti, F. Ciccozzi, and T. Leveque, "On the concurrent Versioning of Metamodels and Models: Challenges and possible Solutions," 2011, pp. 16–25.
- [5] L. Murta, H. Oliveira, C. Dantas, L. G. Lopes, and C. Werner, "Odyssey-SCM: An integrated software configuration management infrastructure for UML models," *Sci. Comput. Program*, vol. 65, no. 3, pp. 249–274, 2007.
- [6] K. Altmanninger, G. Kappel, A. Kusel, W. Retschitzegger, M. Seidl, W. Schwinger, and M. Wimmer, "AMOR—towards adaptable model versioning," *1st International Workshop on Model Co-Evolution and Consistency Management, in conjunction with MODELS*, vol. 8, pp. 4–50, 2008.
- [7] S. Barrett, P. Chalin, and G. Butler, "Table-driven detection and resolution of operation-based merge conflicts with mirador," in *Modelling Foundations and Applications*, vol. 6698, Birmingham, UK.: Springer Berlin Heidelberg, 2011, pp. 329–344.
- [8] R. Conradi and B. Westfechtel, "Version Models for Software Configuration Management," *ACM Computing Surveys (CSUR)*, vol. 30, no. 2, pp. 232–282, 1998.
- [9] K. Altmanninger and G. Kotsis, "Towards accurate conflict detection in a VCS for model artifacts: a comparison of two semantically enhanced approaches," presented at the Proceedings of the Sixth Asia-Pacific Conference on Conceptual Modeling, 2009, pp. 139–146.
- [10] K. Altmanninger and A. Pierantonio, "A categorization for conflicts in model versioning," *e & i Elektrotechnik und Informationstechnik*, vol. 128, no. 11–12, pp. 421–426, Dec. 2011.
- [11] K. Altmanninger, "Models in conflict- towards a semantically enhanced version control models," in *Models in Software Engineering*, vol. 5002, Nashville, TN, USA.: Springer Berlin Heidelberg, 2008, pp. 293–304.
- [12] V. O. Costa, J. M. B. Oliveira Júnior, and L. G. P. Murta, "Semantic Conflicts Detection Method in Model Driven Engineering," in *Proceedings of the Twenty-fifth international conference on Software Engineering & Knowledge Engineering (SEKE 2013)*, Boston, USA, 2013, pp. 656–661.
- [13] T. Mens, "A state-of-the-art survey on software merging," *IEEE Transactions on Software Engineering*, vol. 28, no. 5, pp. 449–462, 2002.
- [14] K. Altmanninger, M. Seidl, and M. Wimmer, "A survey on model versioning approaches," *International Journal of Web Information Systems*, vol. 5, no. 3, pp. 271–304, 2009.
- [15] L. Murta, C. Corrêa, J. G. Prudêncio, and C. Werner, "Towards Odyssey-VCS 2: Improvements over a UML-based Version Control System," in *Proceedings of the 2008 international workshop on Comparison and versioning of software models (CVSM '08)*, Leipzig, Germany, 2008, pp. 25–30.
- [16] A. Cicchetti, D. Di Ruscio, and A. Pierantonio, "Managing model conflicts in distributed development," in *Proceedings of the 11th international conference on Model Driven Engineering Languages and Systems (MoDELS '08)*, Toulouse, France, 2008, pp. 311–325.
- [17] K. Altmanninger, W. Schwinger, and G. Kotsis, "Semantics for accurate conflict detection in smover specification detection and presentation by example," *International Journal of Enterprise Information Systems (IJEIS)*, vol. 6, no. 1, pp. 68–84, 2010.
- [18] S. Maoz, J. O. Ringert, and B. Rumpe, "CDDiff: Semantic Differencing for Class Diagrams," in *Proceedings of 25th European Conference on Object Oriented Programming (ECOOP'11)*, Lancaster, UK, 2011, vol. 6813, pp. 230–254.

# Behavioral Model Generation from Use Cases Based on Ontology Mapping and GRASP Patterns

Nurfauza Jali, Des Greer and Philip Hanna  
 School of Electronics, Electrical Engineering & Computer Science  
 Queen’s University Belfast  
 Belfast BT7 1NN  
 {njali01|des.greer|p.hanna}@qub.ac.uk

**Abstract**— This paper contributes a new approach for developing UML software designs from Natural Language (NL), making use of a meta-domain oriented ontology, well established software design principles and Natural Language Processing (NLP) tools. In the approach described here, banks of grammatical rules are used to assign event flows from essential use cases. A domain specific ontology is also constructed, permitting semantic mapping between the NL input and the modeled domain. Rules based on the widely-used General Responsibility Assignment Software Principles (GRASP) are then applied to derive behavioral models.

**Keywords**- Requirement Engineering, Ontology, Requirement Specification, Natural Language Processing, Software model, UML, Software Design Pattern

## I. INTRODUCTION

One of the biggest challenges in Requirement Engineering (RE) is managing changes. Requirements are most often written and communicated in Natural Language (NL)[1][2] but unfortunately moving from NL to software design is a difficult and time consuming process.

The purpose of this work is to increase the efficiency of software development based on the automated transition from textual to a conceptual visualization of the dynamic software model. An approach is described, where starting with the text-based use-case specifications, objects inside the use-case specifications are elicited and object properties extraction is affected using an ontology representing the domain model. Eventually, behavior is modeled and visualized as Unified Modeling Language (UML) sequence diagrams.

The next section provides an overview of related research. Section III gives a description of the problem under investigation. Section IV demonstrates the solution approach using a ‘Point of Sale’ illustration and the final section discusses the results obtained.

## II. BACKGROUND

There is undoubtedly a challenge in translating a natural language description into usable software. NL descriptions of problem domains captured from the stakeholder are often complex, vague and ambiguous leading to multiple interpretations [3]. Saeki *et al.* [4], and Rupp [5] have analyzed the requirement specification document from a linguistic aspect.

Saeki *et al.*[4], Carasik *et al.* [6], Cockburn [7], Boyd [8] and Juristo *et al.*[9] have explored the use of language as a metaphorical basis (analogical source) for discovering the structure (syntax) of objects and object messages, and for the naming (semantics) of software components. Their work has focused on the production of UML static diagrams (e.g. Use Case diagram and Class diagram). Other tools such as NL-OOP[1], RECORD [10], CM-Builder [11], LIDA [12] and UCDA [13] also only generate static diagrams, probably because this is less complex than building dynamic diagrams. GOOAL [14], CIRCE[15], and a-Toucan [16] are tools that claim to produce UML sequence diagrams besides generating other UML diagrams (use case diagram, activity diagram and class diagram).

Table 1.0 summarizes the types of UML diagrams produced from each of the research projects mentioned above. The tick (✓) symbol represents diagrams that had been implemented in the existing research projects. All the tools investigated are able to identify attributes, objects and methods from requirements text for the production of UML diagrams. However, they require human intervention to interpret the correctly extracted annotation of OO concepts before building a particular diagram.

The existing tools need to enhance their NLP system to produce high quality analyses by allowing for dependable deep semantic analysis and adequate syntactic analysis, in order to produce high quality diagrams. The ontology will be used to ensure that software engineers have a shared understanding of the problem domain with the stakeholders as well as to promote reusability.

TABLE I. CHECKLIST OF UML DIAGRAMS PRODUCED FROM EXISTING RESEARCH PROJECTS.

Diagram Type	NL-OOP	RECORD	D-H	CM-Builder	LIDA	GOOAL	CIRCE	UCDA	UCD-G	UML-SDG	UMGAR	a-Toucan
<b>External</b>												
Use case	X	✓	X	X	X	X	✓	✓	✓	✓	X	✓
<b>Internal &amp; Static</b>												
Class Diagram	X	X	X	✓	✓	✓	✓	✓	X	X	✓	✓
<b>Internal &amp; Dynamic</b>												
Object Diagram	✓	✓	✓	X	✓	X	✓	✓	X	X	✓	X
Sequence Diagram	X	X	X	X	X	✓	✓	X	X	✓	X	✓
Collaboration Diagram	X	X	X	X	X	X	✓	✓	X	X	✓	X
State Transition Diagram	X	X	X	X	X	X	✓	X	X	X	X	X
<b>External &amp; Dynamic</b>												
Activity Diagram	X	X	X	X	X	X	X	X	X	X	X	✓
<b>Prototype</b>												
Fully-Automated	X	X	X	X	X	✓	✓	✓	X	X	X	✓
UC Scenario Specification	X	X	X	X	X	X	X	✓	X	X	X	X

In the production of sequence diagrams, various synthesis techniques were used to build these models from use case

specification scenarios. Work by [17][18][19] were refereed and their approaches will be enhanced to meet the goal of building a precise dynamic model.

### III. PROPOSED APPROACH

We propose an architecture for a toolset ‘Use Case specification to Sequence Diagrams’ (UC2SD) which allows us to produce UML sequence diagrams from the text requirements provided by the stakeholder(s).

This architecture will focus on the modeling aspects of the process, largely where the end result is to generate diagrams and software code to represent the solution. Requirements analysis will involve an automated Natural Language Processing (NLP) process. In turn, the requirements analysis to design phase will involve the extraction of object model elements such as classes, attributes, methods and relationships derived from the NLP. The inclusion of knowledge in related domain ontologies will help to refine the object and properties candidates. In the software design and the implementation phases, these components will assist in building software models such as UML diagrams and software code. The data verification for each module will be evaluated by human experts. Data correction is a part the verification process. Thus we are aiming at design support rather than complete automation. Nonetheless, it is hoped that the NLP can handle the majority of the process in our approach. In what follows, we will describe the research justification for each module in the architecture, its role in achieving the research goal.

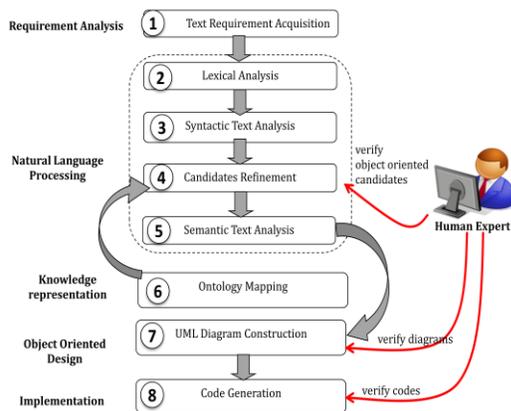


Figure 1. Architecture of Use Case specification to Sequence Diagrams (UC2SD)

#### A. Requirement Analysis Module

##### 1) Text Requirement Acquisition

At this stage, requirements having been elicited are written using specified template rules. By template rules, we mean the use of guidelines or approaches to follow in developing use case specifications descriptions from system requirements [20][21]. The use case is used to capture

system’s behavioral requirements by detailing event-driven threads. In this module, such a template will permit the user to clearly and succinctly input the important use case specification element(s). We have utilized the concept of an essential use case [22], where the use case form is split into user intentions and system responsibilities and target applications that can be specified using this form. In implementing this restriction, we reduce the probability of misinterpretations by an NLP system and at the same time ensure that the user has written the use case in a way that makes it possible to identify user actions and system responsibilities. Later this should help to reduce the need for human intervention. In realizing this, a set of rules in the use case content structure are introduced.

#### B. Natural Language Processing Module

Liddy *et al.* [23] has defined Natural Language Processing or NLP as “a theoretically motivated range of computational techniques for analyzing and representing naturally occurring texts/speech at one or more levels of linguistic analysis for the purpose of achieving human-like language processing for a variety of tasks or applications.” The NLP goal is to transform text information to some internal data model which synthesizes from a data form into a NL surface form [24]. In Linguistic analysis of NL text, Li *et al.* [3] have identified three main components: word-tagging; syntactic analysis; and semantic analysis. In the proposed architecture, the text will be processed in four stages: Lexical Analysis; Syntactic Text Analysis; Candidate Refinements; and Semantic Text Analysis.

##### 2) Lexical Analysis

This stage involves the text tokenization and lexical pre-processing of the input text. Before the Part-of-Speech (POS) tagging process, the input text from use case specification has to be tokenized. Part-of-Speech determines word type and the role a word plays in the phrase structure. This is where the phrase or text is split up into a set of labeled tokens (i.e.: nouns, verb, adjective, adverb, etc...) [21]. In this context, we are using Penn Treebank Tagset [25] as a default set of grammar rules for each tagged token.

##### 3) Syntactic Analysis

Syntactic analysis involves determining the structure of the input text. A single sentence is typically the largest modeled structure within a portion of text, whilst the smallest modeled structures are the basic symbols (i.e. mostly words) within the input text. In contrast with lexical analysis, syntactic analysis takes into account the sentence structure, whereby each lexical token is assigned one or more Part-of-Speech tags. By identifying domain dependent terminology, an initial attempt is made at this point to extract and build a domain model within this stage.

#### 4) Candidates Refinement

The output of POS tagging will tag a set of preliminary noun and verb phrase candidates. It may be that preliminary candidates are poorly defined and often not related to the problem domain. At this stage, the collected candidates can be further analyzed to discover details such as “What are the refined candidate classes?”, “What might be the attribute value?”, and “What kind of relationships hold between the classes?” The solution for these questions will be sought in an iterative manner. This would lead to more detailed and refined object properties which can then be an input to the production of behavioral models.

Domain ontology analysis will facilitate the process of identifying the relevant candidate classes aside from the verification by human experts. In order to refine a preliminary candidate class to relevant classes, the results from the parser will be matched with the concepts and structures defined within the ontology which will be described in the Knowledge Representation Module, later. The algorithm to refine the candidate classes is listed below followed by the identification of attributes and methods.

##### Step 1: Pre-Class identification

- i. Identify candidate classes from the common nouns (e.g. things, persons, places)
- ii. Identify candidate classes from nouns which follow the preposition ‘a’ or ‘the’ (e.g.: a *sale*, the *customer*)
- iii. Identify candidate classes from the ‘IsA’ relationship (instantiation and inheritance)  
e.g.: Credit card is a Payment type.  
Credit Card is a subclass of Payment

##### Step 2: Attribute Identification

- i. Identify attributes from common nouns (e.g.: Class Person attributes are *first name*, *last name*, *address* etc...)
- ii. Identify attributes from adjectives.  
e.g.: sale line item **with** description, price and total.  
Class: *sale line item*  
Adjective: *with*  
Attribute: *description, price and total*
- iii. Identify attributes from the ‘HasA’ relationship (aggregation)  
e.g.: Payment **has an** amount.

##### Step 3: Operation and Relationship Identification

- i. Identify the operation or method from action verbs (e.g.: *calculate*, *start*, *enter*)
- ii. Identify the relationship from static verbs which describe the association relationship.  
e.g.: Class Cashier *works* for Class Manager.  
Class System *records* the Class Sale Line Item.

The outcomes for this process will produce a refined set of candidate classes and eliminate a number of irrelevant classes, which then might be acted as the attributes of the classes. The responsibility of a class is defined in its methods

and fulfilled by collaborating with other classes. This will be further described in the next module.

#### 5) Semantic Text Analysis

In semantic text analysis, the sentence level syntactic text analysis will be combined with the semantic items identified within the ontology to map them onto Object Oriented elements, namely classes, attributes, methods and relationships among the classes. Here, a rule-based approach is followed for identifying actors, objects, class attributes, messages etc. Behavior of each object are identified using the Subject-Verb-Object (SVO) approach, based on a word order of a typical sentence pattern. The verb carries the action across to a target or receiver. To illustrate, the SVO approach can be applied to a use case sentence as follows: “a customer (*subject*) creates(*verb*) account(*object*)”.

In particular, a set of syntactic rules is proposed to assist the software developer in writing and normalizing the use case specification. The behavior element written in the specification statements can be read by machines using this syntactic pattern matching. At this stage, the meaning and relationship of each sentence will be analyzed iteratively. The semantic checking is performed here to resolve ambiguities.

#### C. Knowledge Representation

In this module, the business domain knowledge is represented using suitable business ontology. We have selected the Business Management Ontologies (BMO) [26] as suitable for application to our case study. The current version of BMO has about 40 ontologies with around 1300 classes designed to allow the user to define private, public and collaborative business processes (using Business Process Modeling Notation). The content includes the data (instances) and definition where the higher-level ontology may import one or more lower level ontologies.

This mapping process will be combined with the result from the candidate refinement stage using the POS-BMO ontology with the Protégé-OWL tool[27].

#### D. Object Oriented Design Module

The output from the NLP processor will be used to generate sequence diagrams. Participating actors/ objects/ classes, messages/ methods and attributes are mapped respectively with nouns, verbs and adjectives and are then translated into UML sequence diagram constructs. At the beginning the system sequence diagram (SSD) will be generated, followed by the detailed sequence diagram.

To move to detailed sequence diagrams, we must consider how objects collaborate with other objects to implement system operation. To achieve this we have applied the General Responsibility Assignment Software Principles (GRASP) [2], including Creator, Information Expert, Controller, Low Coupling and High Cohesion principles. These are used as follows:

- i. Creator – to determine who should be responsible for creating a specific object.

- ii. Information Expert - should be responsible for a responsibility based on it having the necessary data.
- iii. Controller - to determine which should be the first object to receive a message from an external actor
- iv. Low Coupling – used to choose between objects for responsibility assignment, based on interaction between objects
- v. High Cohesion – used to choose between objects for responsibility assignment and it measures how strongly related and focused are the responsibilities of each class.

The goal of applying these principles is to identify object’s responsibility which in turn establishes its collaboration.

### E. Implementation

Once the sequence diagrams have been generated, they can be checked and verified using human experts. Indeed, compilable code can be created from the sequence diagrams, if desired. This phase will not be discussed in this paper as the work is still in progress.

## IV. DEMONSTRATION AND ANALYSIS

A Use Case specification to Sequence Diagrams (UC2SD) generator was designed and constructed in order to demonstrate the approach. We used the processing resources that GATE [28] provides, which are made available in the form of plug-ins. GATE makes it possible to use the Java Annotations Pattern Engine (JAPE) transducer which provides a way to process text over specified annotations and to further identify patterns or entities in text.

Input text is from the use case provided by the user, which needs to be tokenized and split into sentences. Each token (i.e. number, word, punctuation) is then assigned with Part-of-Speech (POS) tags where the grammars are based on Penn Treebank Tagset which applies the Hepple's Brill-style tagger. This process is assisted by a morphological analyzer which involves lemmatization or word stemming.

Next, the JAPE transducer will trigger the grammar rule to identify and annotate objects and messages from the given Syntactic Rules (SRs), discussed earlier. The JAPE syntaxes have been developed for all of the SRs.

Thus, before the XML is produced, a frequency analysis step is carried out to produce frequency lists of overall word form. The selection of candidate classes are based upon the frequency of the nouns’ appearance and the result will then be verified by the user. This object property extraction is used to construct a System Sequence Diagram (SSD). The SSD is a sequence diagram that shows the event interaction between external actors with the system object. Figure 2 and 3 illustrate the Point of Sale (POS) system specification for process sale use case and the SSD generated. This SSD describes: (1) each method with a sequence number label above the arrow; (2) method parameters in brackets for each message; (3) message represented as solid arrows and returns represented as a dotted line arrow.

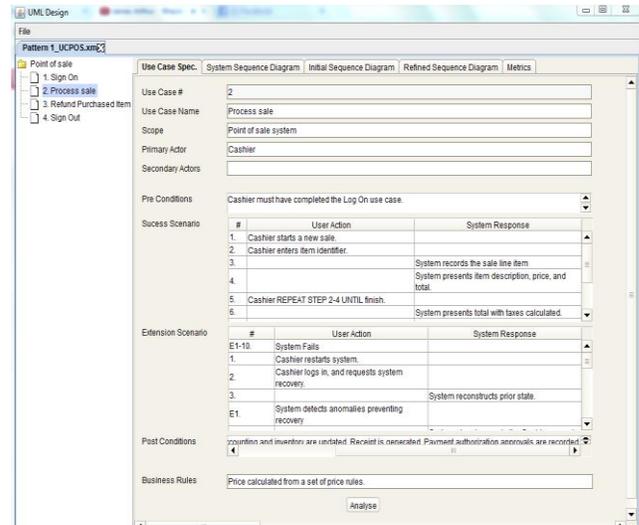


Figure 2. UC2SD Automation of System Sequence Diagram (SSD) production

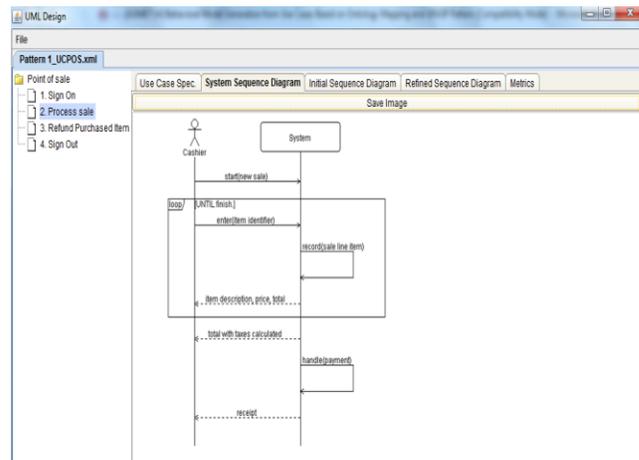


Figure 3. System Sequence Diagram (SSD) of Process Sale use case generation

The SSD generation is a relatively straightforward task as it only involves the external actors, message flow and System object. However, a more detailed system design of SD can be derived with potential classes involving three common stereotypes (boundary, controller and entities). Thus, to construct a refined Sequence Diagram (rSD), we finalized the potential classes and determined responsibilities of objects within the system. To achieve this, we use the POS-BMO ontology to map the extracted object properties to appropriate objects.

First it adds all the entity objects from preceding ontology analysis and then it transforms some individual messages on the GRASP rules. Finally it divides the System class into UI, Controller and Entity classes and as well adjusts the messages accordingly. All of this is currently done against a representation of the sequence diagrams in XML, via the Document Object Model (DOM) API.



Figure 5 illustrates the refined Sequence Diagram (rSD) that been generated from the tool based on the rules implemented. From the generated diagram it is clearly has similar result with the sample given by the expert (based on Larman's book) because of the strong semantic support and rules constructed applied to the system tool.

#### CONCLUSION AND FUTURE WORKS

In automatically producing behavioral models from text the following tasks have been accomplished:

- A demonstration of linguistic algorithms to identify the proper object, classes, attributes relationships and so forth for building a System Sequence Diagram and Refined Sequence Diagram by applying rules based on GRASP Principles.
- A process has been developed for building an ontology to help improve the mapping process of words terms nouns/verbs etc.) to UML notations, particularly relating to behavior (objects/messages etc.).

One of the limitations of this work is that, since ontology practice is still immature, it is hard to find comprehensive ontology resources. In order to refine the current ontology, it may be worth looking at how well BMO and *GoodRelations* [29] could be aligned with each other.

In future work, we will further derive and formulate the remainder of the GRASP principles, particularly low coupling and high cohesion. While there still remain many challenges in deriving compilable code from use case specifications, the advances in NLP theory and tool support offer the possibility of moving closer to this goal.

#### ACKNOWLEDGMENT

The authors gratefully acknowledge Ministry of Higher Education (MOHE) Malaysia, as part of the first author's PhD studies scholarship.

#### REFERENCES

[1] L. Mich, "NL-OOPS: from natural language to object oriented requirements using the natural language processing system LOLITA," *Natural Language Engineering*, vol. 2, no. 2, pp. 161–187, 1996.

[2] C. Larman, *Applying UML and Patterns*, 2nd ed. Prentice Hall, 2001.

[3] K. Li, R. G. Dewar, and R.J.Pooley, "Object-Oriented Analysis Using Natural Language Processing," in *Linguistic Analysis*, 2005.

[4] H. Saeki, Motoshi. Horai, Hisayuki. Enomoto, "Software development process from natural language specification," *Proc. 11th International Conference in Software Engineering - ICSE '89*, pp. 64–73, 1989.

[5] C. Rupp, "Requirements Templates — The Blueprint of your Requirement," *Requirement Engineering*, 2004.

[6] Robert P Carasik, S. M. Johnson, D. A. Patterson, and G. A. Von Glahn, "Towards a Domain Description Grammar: An Application of Linguistic Semantics," *Engineering*, vol. 15, no. 5, 1990.

[7] A. Cockburn, "Using natural language as a metaphoric base for OO.," *SIGPLAN OOPS*, vol. 4, pp. 187–189, 1993.

[8] N. Boyd, "Using Natural Language in Software Development," *J. Object Oriented Program.*, 1999.

[9] N. Juristo, A. M. Moreno, and M. López, "How to Use Linguistic Instruments for Object-Oriented Analysis," *IEEE Software*, no. June, pp. 80–89, 2000.

[10] J. Börstler, "User-Centered Requirements Engineering in REquirements COllection Reuse and Documentation (RECORD) - An Overview," in *Proceedings Nordic Workshop on Programming Environment Research (NWPER '96)*, 1996, pp. 149–156.

[11] H. M. Harmain and R. Gaizauskas, "CM-Builder: an automated NL-based CASE tool," *Proc. ASE 2000. Fifteenth IEEE International Conference Automation Software Engineering*, pp. 45–53, 2000.

[12] S. P. Overmyer, L. Benoit, and R. Owen, "Conceptual modeling through linguistic analysis using LIDA," in *Proceedings of the 23rd International Conference on Software Engineering (ICSE)*, 2001, pp. 401–410.

[13] K. Subramaniam, D. Liu, B. H. Far, and Armin Eberlein, "UCDA: Use Case Driven Development Assistant Tool for Class Model Generation," in *The Sixteenth International Conference on Software Engineering and Knowledge Engineering (SEKE '04)*, 2004, pp. 1–6.

[14] H. G. Perez-gonzalez and J. K. Kalita, "GOOAL: A Graphic Object Oriented Analysis Laboratory," in *OOPSLA '02*, 2002, pp. 38–39.

[15] V. Ambriola and V. Gervasi, "The Circe approach to the systematic analysis of NL requirements," 2003.

[16] T. Yue, L. C. Briand, and Y. Labiche, "Automatically Deriving UML Sequence Diagrams from Use Cases," Norway, 2010.

[17] Z. Ding and M. Jiang, "Model Driven Synthesis of Behavioral Models," *Informatics Control. Autom. Robot.*, vol. 132, pp. 713–717, 2011.

[18] C. Damas, B. Lambeau, P. Dupont, and A. Van Lamsweerde, "Generating Annotated Behavior Models from End-User Scenarios," *IEEE Trans. Softw. Eng.*, vol. 31, no. 12, pp. 1056–1073, 2005.

[19] Z. Ding and M. Jiang, "Model Driven Synthesis of Behavioral Models from Textual Use Cases," in *Informatics in Control, Automation and Robotics*, Springer, 2011, pp. 713–717.

[20] M. G. Ilieva and O. Ormandjieva, "Models Derived from Automatically Analyzed Textual User Requirements," *Fourth International Conference Software Engineering*, pp. 13–21, 2006.

[21] L. Kof, "Text Analysis for Requirements Engineering," Technischen Universitat Munchen, 2005.

[22] E. T. Robert Biddle, James Noble, "Essential Use Cases and Responsibility in Object Oriented Development," 2001.

[23] E. D. Liddy, E. Hovy, J. Lin, J. Prager, D. Radev, L. Vanderwende, and R. Weischedel., *Natural Language Processing*, 2nd ed. Marcel Decker, Inc., 2003, pp. 2126–2136.

[24] E. Brill and R. J. Mooney, "An Overview of Empirical Natural Language Processing," *AI Magazine*, vol. 18, no. 4, pp. 13–24, 1997.

[25] C. D. Manning, "Part-of-Speech Tagging from 97 % to 100 %: Is It Time for Some Linguistics?," in *12th International Conference, Computational Linguistics and Intelligent Text Processing, (CICLING 2011)*, 2011, pp. 171–189.

[26] Jenz and P. Gmbh, "Business Management Ontology ( BMO ) Version 1.0," Germany, 2004.

[27] "The Protege Ontology Editor and Knowledge Acquisition System," *Research, Stanford Center for Biomedical Informatics*, 2010. [Online]. Available: <http://protege.stanford.edu/overview/protege-owl.html>. [Accessed: 02-Nov-2011].

[28] K. Bontcheva, V. Tablan, D. Maynard, and H. Cunningham, "Evolving GATE to meet new challenges in language engineering," *Natural Language Engineering*, vol. 10, no. 3–4, pp. 349–373, 2004.

[29] M. Hepp, "GoodRelations: An Ontology for Describing Web Offerings Final Version The GoodRelations Ontology," *Information System Journal*, p. 105, 2008.

# *Semantic-based Repository of Agent Components*

Merlin Parra Jiménez, Andrew Diniz da Costa, Carlos José Pereira de Lucena  
Software Engineering Laboratory  
Pontifical Catholic University of Rio de Janeiro  
Rio de Janeiro, Brazil  
merlin@les.inf.puc-rio.br  
{acosta, lucena}@inf.puc-rio.br

**Abstract**— We posit that a robust development environment for the construction of agent-oriented software systems must be enhanced by advanced reuse methods. However, research addressing agent reuse is meager and does not tackle the problem of organizing and storing agent-oriented artifacts according to the software engineers' needs. Therefore, the agent retrieval process turns into an important challenge to be overcome in agent-oriented software engineering. In this context, this paper proposes a repository prototype based on semantic web technologies that support reuse for developing agent systems. The repository includes: (1) a meta-model for representing the agents, represented by means of an ontology, (2) a taxonomy to classify agents according their application domains, (3) a recommendation system that allows end-users to discover reusable interrelated agents, and (4) enhanced search and browsing methods for agents. Finally, we discuss an illustrative example that shows the proposed reuse method is an improvement in terms of the relevance of retrieved agent-oriented artifacts.

*Software agent reuse; agent component; agent-oriented artifact; agent repository; semantic information retrieval; agent-oriented software engineering*

## I. INTRODUCTION

Nowadays the development of complex systems is extremely common. Considering this context, the multi-agent system (MAS) paradigm [1] has been used especially when distributed, autonomous and pro-active entities are represented.

A MAS can be composed of several software agents [1], which interact among them to achieve common goals. Developing an agent-based system from the ground up is a difficult, expensive, and lengthy software-engineering activity due to the various kinds of expertise necessary. In developing a multi-agent system, it is necessary to define multiple agents that communicate and cooperate among themselves while engaging in group decision making as well as competitive behavior. The benefits of applying software-engineering principles to guide problem solving have not been fully appreciated and applied by the agent-development community.

According to [2], reusing agents created previously for other systems is a crucial point that must be considered in the agent-oriented software engineering (AOSE) [3]. It brings several advantages to multi-agent systems such as: (1) construction of reliable and consistent software since it uses code created and tested, (2) reduction of cost and time

developing agents, and (3) increased flexibility of the software to facilitate its maintenance and evolution.

Today, although software artifact reuse is already established in the literature on software engineering, the work addressing agent reuse is meager and does not tackle the problem of identifying, organizing and storing agent-oriented artifacts for reuse. Hence, the process of retrieving existing reusable agent-oriented artifacts from different application domains, or *agent retrieval*, is limited by the absence of appropriate mechanisms and standards. In this context, the entire agent retrieval process, which includes indexation, identification, storage and recovery, turns into a crucial impediment to be overcome in AOSE.

In order to provide a solution, we propose a repository prototype based on semantic web technologies that identifies and locates appropriate reusable agent components according to user needs. A (software) agent component is a software component [4] that takes advantage of both software agent and component techniques (reusability, economy, extensibility). The interface of an agent component is part of a component that defines explicitly and semantically its access points (e.g., communication protocol), which make the agent services available to the environment and accessible by other agents.

For a software agent to be effectively reused, its specifications must be flexible enough and easy to adapt to the many variations that can exist in an application domain. Consequently, we created an ontology to model the agent components based on characteristics stipulated by the Foundation of Intelligent Physical Agents (FIPA) [5]. Ontologies have emerged as a powerful tool to enable knowledge representation and knowledge sharing, by a community of a target domain in an explicit and formal way, and to achieve semantic interoperability among heterogeneous distributed systems [6]. Thus, ontologies enable knowledge reuse and a standardized model of software artifacts.

## II. RELATED WORK

There have been some initiatives on exploiting software reuse in agent technology [7][8]. The works [2][4] discuss the need of toolkits and libraries of agents, agent parts or pre-connected agent societies, that could allow their use in different systems. Researchers have explored the possibility of using agents as advanced reusable artifacts of software. In this scenario, agents should exhibit at least the same properties of

software components, adding interesting characteristics like autonomy, reasoning and goal-direct behavior [9][2].

We realized which difficulties exist to reusing software agents, such as the lack of mechanism or standards to represent and retrieve agent-oriented artifacts, and the lack of architectures and programming languages that allow MASs to implement all the features of agents. The capabilities of existing component repositories are not sufficient to provide for the reuse of heterogeneous agent-oriented artifacts, since they do not support a model of agents that includes their dimensions and interactions, or their retrieval according agent characteristics. Therefore, we propose a repository prototype based on semantic web technologies that exploits reuse in agent-oriented development among different agent architectures, platforms and languages in diverse application domains.

### III. REPOSITORY OF AGENT COMPONENTS

The repository provides a web interface that implements the searching and browsing mechanisms to allow the user to view which agents offer certain functions in a particular category or with particular specification, furthermore submit search queries. The feature of searching artifacts is supported by a recommendation system.

In order to register an agent component, a description of the agent has to be provided by the user. Users can edit only the agent components added by them. Users can download an originally created agent component from the repository, a newer version of existing component or its entire predecessor's history. These functionalities are supported by the core of the repository, whose main characteristics are described next.

#### A. Modeling Agent Components

Although agents are developed under diverse architectures, platforms, languages, and agent components are different in their interfaces, internal architecture and functionalities, we observed they could be modeled within a standard and intelligible representation.

We defined a meta-model to translate the dissimilar agent components into this representation. It establishes a precise and formal description that shares the understanding of agent components regardless of the types of components that are in the repository. Therefore, heterogeneous agent components can be stored and retrieved without losing any of their own characteristics.

For structuring the meta-data with the information of agents provided by the user, the repository uses an ontology. An important advantage of the ontology to maximize the reusability is to allow the extensibility of agents, i.e., to extend features of an agent (e.g. collaboration protocols or knowledge) without breaking the previous architecture or the reusability of the agent in the system that contains it. Thus, the ontology is open for anyone to use and explore.

After a thorough examination of available research on ontologies that describe software agents (functionalities, structure and interfaces), no appropriate result for modeling the common structure of heterogeneous agent components was found. Hence, we complement existing ontologies since some

agent characteristics are not still covered, for example the interfaces of heterogeneous agent components. Therefore, we propose the ontology depicted in Figure 1 to formalize the description of dissimilar agent components.

A reusable agent component should have a number of attributes that are essential in determining the appropriateness of this construct in the reuse process. These attributes, which are considered as meta-data for the agent component specification are the following:

*a) ID:* Identifier given to the agent. It is unique in the system.

*b) Name:* Name given to the agent.

*c) Description:* Description of the agent in natural language, which can refer to whatever information related to the agent.

*d) Version:* Developmet version of the agent.

*e) Previous Version:* If the version is not the first one, the agent is a new one based on a previous version.

*f) Date:* Date when the agent was developed.

*g) Language:* Programming language the agent was developed with.

*h) Platform:* Platform the agent was developed with.

*i) User:* Developer or team responsible for implementing the agent.

*j) Dimensions:* Characteristics of the agent according to its nature, like autonomy, reactivity, etc.

*k) Roles:* Roles the agent performs.

*l) Operations:* Operations each role executes.

*m) Parameters:* Requirements an operation has to execute.

*n) Categories:* Application domains associated to kind of roles.

*o) Tags:* Tags the agent can be described with a few of words. Each tag has its own weight in the system.

*p) File:* File (.JAR or .ZIP for example) which contains the agent component.

*q) Related:* There are agents related to the current one respect to some characteristics (description, roles, previous version, interfaces, categories and tags).

*r) Interfaces:* Unlike software components, there is not a standard format to document agents' interfaces based on the interfaces of public functions that include restrictions on the behavior of objects, such as the order in which the functions/operations should be invoked. To support communication, compatibility or interoperability among all of these heterogeneous agents, we propose an interface model that describes the context (cooperation, coordination, and negotiation) of relations among the agents, the types of messages the agent sends or receives based on the communicate acts of FIPA [5], the content of the message that can include ontologies, and the other agent participants.

*s) Diagram:* Image file to illustrate the structure of the agent showing its class and its relationships in its environment.

*t) Documentation:* URL or plain text that explains how an agent interacts in its environment or how it should be used.

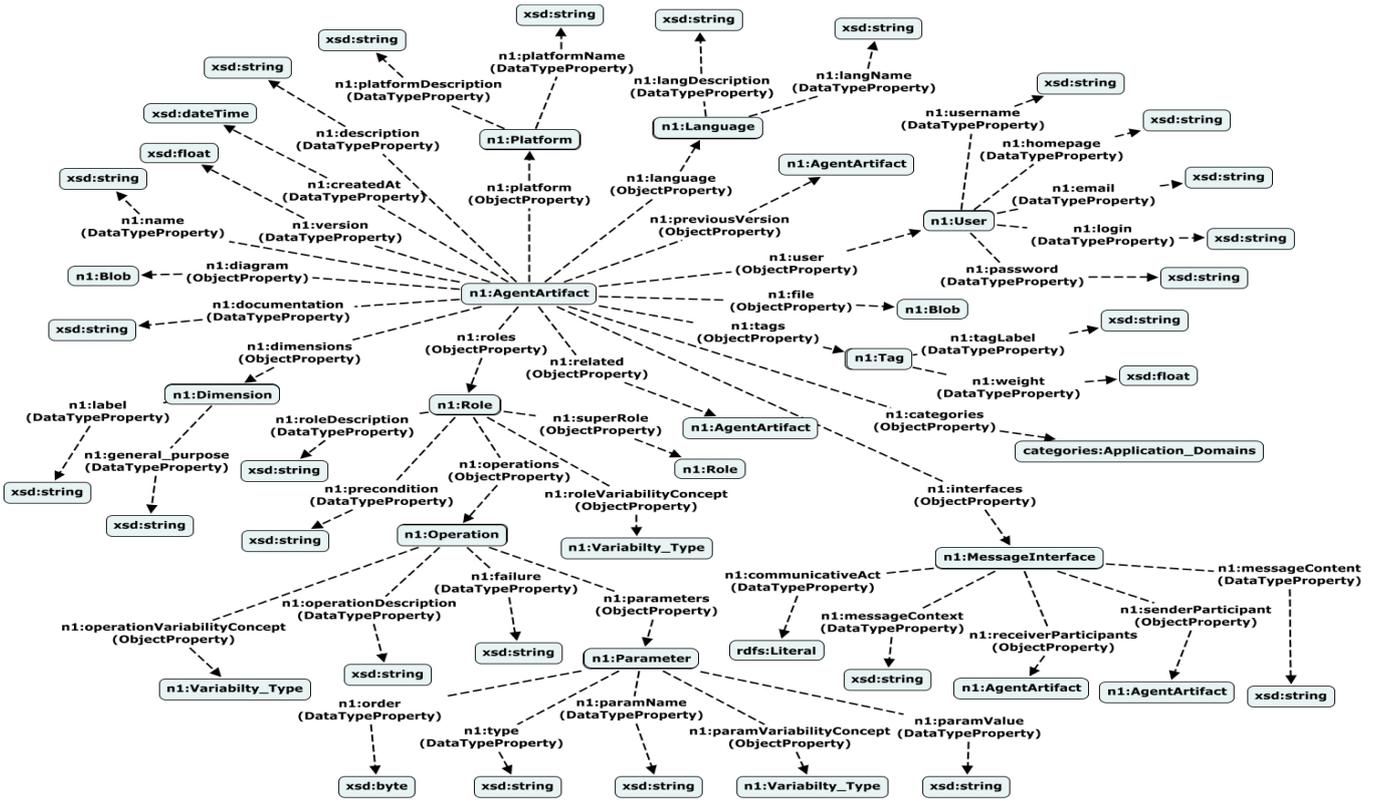


Figure 1. Ontology to Model an Agent Component.

An individual example of this ontology is depicted in [10]. It consists on an agent that buys a specific book at the lower price at online seller libraries.

### B. Classifying Agent Components

Classifying agent-based artifacts allows software engineers to organize collections of them into structures that they can look easily for in future searches. In order to perform such classification, a hierarchical taxonomy of application domains was adopted. However, there are several perspectives to classify existing software agents, which are not unique.

The taxonomy facilitates to the user during a search, to access not only the agents related to keywords in the query, but also those agents that are interrelated semantically to them, such as those with equivalent characteristics or terms. It reduces the search space and makes the search results more relevant. In [10] there is a graphical subset of our taxonomy and the complete version in owl format. It is built with several facets that compose the agents' domains, using a common vocabulary, derived and defined by the agent developers/users.

### C. Recommendation System

To support the search in the repository, we created a recommendation system (RS) that allows end-users to discover the existence of reusable interrelated agents, and to learn new information about agents as needed, improving user productivity and promoting agent reuse. To measure the degree of similarity or relationship among the artifacts in the repository, the RS adapts the tf-idf algorithm [11], to compute weights of terms belonging to the agent components, and

advises the user which agents would be related. Later, the user can establish the relations, and a degree of relationship is assigned automatically. The RS tracks usage histories of a group of agents to recommend agents expected being needed by that user. In that way, an ontology network, that contains the discovered data and their associative relations, is constructed.

The attributes taken into account for the recommendation are the description, roles, interfaces, ancestor, categories and tags.

### D. Semantic-based Search System

For retrieving suitable agent-based artifacts, we developed a suite of search methods that utilizes the semantics of the agent descriptions, streamlining and reducing the search time.

The whole search process combines these phases: (1) pre-processing the agent's specifications and queries i.e., conflating related words to a common word stem, (2) interpreting user queries to choose the specific domains and establish the search method and parameters in terms of the characteristics of agents, (3) splitting these domains into sub-queries and executing them, (4) retrieving the agent components, rating and filtering them according to their semantic relevance with user parameters, (5) analyzing and classifying the results to infer if there is new relevant information that can enrich the semantic knowledge base, and (6) presenting the results to the user in the way further exploration of each component is enabled. The semantic knowledge base contains the data from the ontology and taxonomy, the indexes, the index terms and the learned and inferred knowledge during the search process. The indexation is performed by means of a lexical database of English.

The different ways to retrieve agents in the repository are described below.

1) *Keyword-based Searching*: Initially, we create a vector for the query with its terms. Afterwards, we calculate the cosine similarities [11] between the query vector and the vectors that represent the index terms of all agent components already stored in the repository.

2) *Tag-based Searching (Tag Cloud)*: It is a weighted list that exemplifies the density of keywords present according to its relevance on the agent characteristics using a variety of fonts in the visual design. This allows a user to quickly identify what archetypes of agents are more common in the repository. Selecting a tag, the tag cloud will work as a browser leading to a collection of agent components that are associated with that tag.

3) *Custom Search Facet*: A tree-structure-based search representation model is conceived to allow users to browse, locate and filter their search intention by functionality. After a user lists all agent components related to an application domain, he can refine that result set. For this, all tags of all resulting agents are showed and the user under that category can mark the tags that he is interested on. Thus, the agent component result set is refined.

4) *Platform-based Searching*: It is looked up on the ontology all agent components already stored in the repository that were developed in the platform passed as parameter. It is an analogous process respect to the programming language.

5) *Interface-based Searching*: It is looked up all the agents that interact in a certain context, e.g. cooperation, coordination and negotiation, to achieve a specific role. In addition, it can be looked up the specification of message exchange or the description of the participants to accomplish this interaction.

There is a user manual in [10] that contains all essential information for the user to make full use of the repository system.

#### IV. ILLUSTRATIVE EXAMPLE

To evaluate correctness (task-performance) and effectiveness (time-performance) of a method of retrieval software artifacts in the repository, we rely on two metrics that are standard practice for retrieval, recall and precision [11]. Recall means to get all the relevant components, while precision means that all the retrieved components exactly match the query submitted by a user.

To evaluate our information model, we developed a study that involves twenty-eight agent components stored in the repository [10] and a produced test collection. A test collection is a set of elaborated queries that cover all the components and the associated set of relevant components that is known a priori. These artifacts were implemented using Java language but with dissimilar platforms in different application domains by unrelated developers.

Then, the search of each query is performed using mainly the keyword-based, tag-based and interface-based approaches, with a fixed minimum relevance acceptance value of 0.6 [11].

Since we know a priori which agents are relevant among all that are returned to that query and how many relevant artifacts are in the repository, we can calculate the recall. We also know the quantity of agents retrieved, making it possible to calculate the precision.

We realized that the values of recall and precision have the same behavior in the case of the searches based on language, platform, categories and tags. Just in this particular case, an exact match of the query and the respective attribute was made.

To conclude, we find that a semantic approach can significantly improve precision and recall of search methods.

#### V. FINAL REMARKS

This paper presents a prototype repository to support reuse for developing agent-oriented systems. The repository relies on semantic web technologies for the description and retrieval of the components: (1) an ontology to set up all the characteristics associated with heterogeneous software agents, and (2) a taxonomy to classify these components according to their application domains.

We aim (1) to implement some combination of the search techniques. For instance, we could combine a keyword-based approach with the facet-based search or consider combinations of attributes as input data, e.g. fixed platform and programming language; (2) to represent the (semantic) intersection among application domains with the taxonomy. We will have to analyze if both of these improve the relevance of the searches. In addition, we must evaluate the performance of the search system taking into consideration the continual agent component registration, since new concepts can be included dynamically in the taxonomy and the tag cloud.

#### REFERENCES

- [1] M. Wooldridge, "An Introduction to Multi Agent Systems," Wiley, 2002.
- [2] M.L. Griss, "Software Engineering with Java Agent Components," November 2003. Available at <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.85.1125>.
- [3] N.R. Jennings, "Agent-based Computing". Available at <http://users.ecs.soton.ac.uk/nrj/download-files/ifip-02.pdf>.
- [4] M.L. Griss, and R.R. Kessler, "Achieving the Promise of Reuse with Agent Components," Software Engineering for Large-Scale Multi-Agent Systems, Springer-Verlag, 2003, pp. 139-147.
- [5] FIPA, <http://www.fipa.org/>.
- [6] N. Guarino, "Formal Ontology and Information Systems," Proceedings of the 1<sup>st</sup> International Conference, June 1998, Italy.
- [7] F. Bergenti, "Formalizing the Reusability of Software Agents." vol. 3071, Springer, 2004, pp 246-257.
- [8] R. Girardi, "Reuse in Agent-based Application Development," 2002. Available at <http://maae.deinf.ufma.br/Ensino/ES/CGCC/Reuse%20in%20Agent-based%20Application%20Development.pdf>.
- [9] F. Bergenti, and M.N. Huhns, "On the Use of Agents as Components of Software Systems," Available at <http://www.cse.sc.edu/~huhns/chapters/BergentiHuhnsAOSE.pdf>.
- [10] Semantic-based repository of agent components. Available at [http://www.les.inf.puc-rio.br/wiki/index.php/Semantic-based\\_Repository\\_of\\_Agent\\_Components](http://www.les.inf.puc-rio.br/wiki/index.php/Semantic-based_Repository_of_Agent_Components).
- [11] C.D. Manning, P. RAGHAVAN, and H. Schütze, "An Introduction to Information Retrieval," Cambridge University Press, April 2009.

# A Multi-Agent-Based Approach for Autonomic Data Exchange Processes

*Hicham Assoudi*

LATECE, Département d'Informatique  
Université du Québec à Montréal  
Montréal, Canada  
*assoudi.hicham@courrier.uqam.ca*

*Hakim Lounis*

LATECE, Département d'Informatique  
Université du Québec à Montréal  
Montréal, Canada  
*lounis.hakim@uqam.ca*

**Abstract**—In this paper, we present a prototype for our solution called Data Exchange Autonomic Manager (DEAM) [1] which has as main goal to turn Data Exchange processes into self-managed systems. We believe that providing data exchange processes with self-healing autonomic capability is a promising approach toward reliable self-managed and resilient data exchange processes. We describe the high level architecture of DEAM prototype which leverages well established techniques and technologies for Autonomic Computing (Multi-Agent Systems) and Schema Matching (Automatic Schema Matching and Mapping).

**Keywords**—*data exchange; autonomic computing; self-managed systems; dependability; fault tolerance; sufficient correctness; schema matching; schema mapping; mapping adaptation; multi-agent systems; agent based modelling and simulation*

## I. INTRODUCTION

In a previous paper, we proposed a self-healing based approach, called Data Exchange Autonomic Manager (DEAM) [1], to answer the following question: How data exchange processes can be “autonomically” resilient to schema evolution changes? The aim of the proposed solution, DEAM, is to tackle to the problem of reducing the human intervention needed to maintain the data exchange processes after a schema evolution [2] (changes impacting source or target system schemas participating in a data exchange scenario). DEAM prototype leverages well established techniques and technologies for Autonomic Computing (Multi-Agent Systems) and Schema Matching (Automatic Schema Matching and Mapping). As far as we know, there is no previous literature describing a solution for self-managed or autonomic data exchange processes.

## II. RELATED WORK

Data exchange problem is defined by Arenas et al. [3] as the problem of finding an instance of a target schema, given an instance of a source schema and a specification of the relationship between the source and the target. Data exchange is strongly related to two distinct but complementary concepts: Schema matching and mapping. Those two concepts are often confused and discussed under the single name "Schema Matching" [4].

Schema matching (including its ontology matching variant) has been a very active research area, especially in the last decade, and numerous techniques and prototypes for automatic matching have been developed [5], [6]. Unfortunately, schema matching remains largely a manual and time consuming process [7]. Schema matching has been used as a first step to solve data exchange, schema evolution, or data integration problems [8]. It has also been used for resolving Schema Evolution problems [2] by Maintaining the consistency of mappings under schema changes by finding rewritings that try to preserve as much as possible the semantics of the mappings (Mapping Adaptation [9], [10]).

One of the approaches or paradigms to automate the management of IT solutions, inspired by the nervous system, was proposed by IBM under the name of "Autonomic Computing" [11], [12]. The systems use autonomic strategies and algorithms to handle complexity and uncertainties with minimum human intervention. An autonomic system is a collection of autonomic elements, which implement intelligent control loops to monitor, analyze, plan and execute using knowledge of the environment. One of the main four properties defining an autonomic system is Self-healing. Ghosh et al. [13] provide a definition of self-healing system: “A self-healing system should recover from the abnormal (or “unhealthy”) state and return to the normative (“healthy”) state, and function as it was prior to disruption.”

Agent-Based Modelling and Simulation (ABMS) is a very natural and flexible way to model today's complex, distributed, interconnected and interacting industrial systems: the self-X principles can be mapped on the explicit notion of autonomous agents, modelling a system as a group of interacting agents maps directly onto the distributed and communicative nature of today's systems [14], [15].

## III. DEAM PROTOTYPE

This section presents the overall architecture of DEAM and detailed descriptions of agents responsibilities. As we already mentioned in the introduction, DEAM leverages well established techniques and technologies from different research areas:

- Autonomic Computing and Multi-Agent Systems for automatic fault detection, diagnostic and repair (Self-Healing capabilities based on autonomic computing principles);
- Schema Matching for automatic Schema Matching and Mapping (generation of new mapping to recover from mapping degradation after a schema evolution).

In order to implement the autonomic computing principles (i.e. MAPE-K loop) we are leveraging the open source multi-agents platform Jadex which is an extension of the popular and mature java multi-agents platform Jade. Jadex allows the creation of BDI agents supporting the full BDI reasoning cycle with goal deliberation and means-end reasoning [16].

For schema matching, DEAM is leveraging COMA (COMA 3.0 Community Edition). COMA is (previously named COMA and COMA++) is a generic prototypes for schema and ontology matching and it was among the first systems to successfully support the multimatcher architecture and match workflows [17].

### A. High-level Architecture

The figure below shows the DEAM architecture, including agents, human actors, and the relationships between agents.

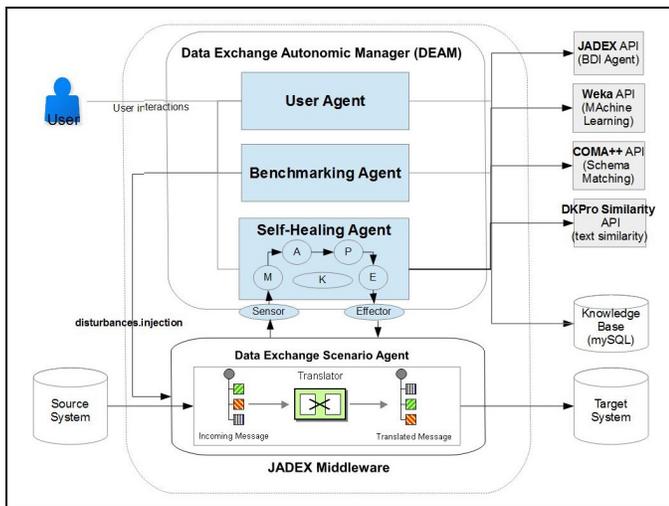


Figure 1. DEAM High level architecture

The the main logical components of DEAM are:

- Data Exchange Scenario Agent: This component represent the data exchange scenario implemented as an active Component in Jadex. Its responsibility is to receive an inbound XML message (from the source system), transform it to the target system format and deliver it to the target system.
- Self-Healing Agent: Jadex BDI agent implementing the MAPE-K loop and responsible for the whole self-healing process. This agent will have to perform the following steps as part of its self-healing process:

- Detects a failure at the data exchange process and translate it to a standard event format (e.g. Detect a change at the source XSD file after schema evolution).
- Analyze the event, diagnose a mapping degradation issue (correlation rules) and isolate the broken mappings. During this step the original and the evolved schemas will be compared to capture the changes and to classify them into a set of primitive actions [10] (e.g. adding/deleting elements, merging/splitting elements, etc.).
- Plan a mapping adaptation by selecting, based on Knowledge Base, the best mapping adaptation approach.
- Apply the mapping adaptation plan. During this step DEAM will have to modify the data exchange artifacts such as schema definition file and mapping rules file (i.e. XSD, XSLT).

- Benchmarking Agent: This BDI agent provides verification capabilities for the evaluation of the correctness of the healing plans generated by DEAM. It is responsible for automatically generating and injecting perturbations into existing schemas (i.e. DisturbanceInjector) in order to trigger the self-healing agent and evaluate the generated healing plans (i.e. DisturbanceAnalyser: evaluate the correctness of the proposed recovered mappings vs actual correct mappings). The idea behind this agent was inspired from the work done for autonomic benchmarking [18].
- User Agent: This BDI agent is responsible for all the GUI user interactions.

We developed a prototype of DEAM to conduct some preliminary experiments in order to evaluate the idea viability. In the next section, we discuss the experiment example and the obtained results.

### B. Prototype Evaluation

This initial version of the prototype implements only the component Self-Healing Agent. In the next releases, we are going to implement the remaining agents: Benchmarking Agent and User Agent. Thus, the experiments conducted so far focuses only on the Self-Healing Agent component (which is the central piece of DEAM's architecture).

To describe the experiment we conducted on DEAM prototype, let's consider the following data exchange scenario. We need to transfer a shipping order between two systems.

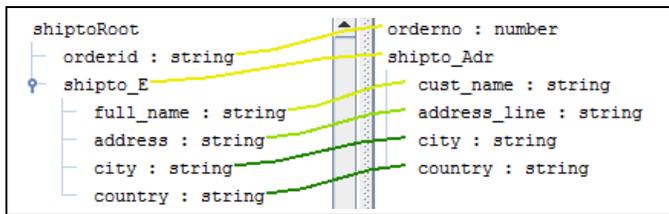


Figure 2. Initial mapping

The figure above describes the initial mapping between the source and target before the schema evolution for this experiment. After manually introducing a schema evolution (manual modification of the source XSD file to replace the full\_name XSD element with two new XSD elements: first\_name and last\_name), DEAM's monitoring task detected the change and triggered the self-healing process. Below, a summary of self-healing process steps.

- Step 1: Monitor a change (Monitor Task)
  - Detect the change (XSD changed after schema evolution)
- Step 2: Analyze the change (Analyze Task)
  - Detect the broken mapping (identify, based on the deleted elements the broken mappings)
  - choose a diagnosis **Diagnose\_Mapping\_Degradation**
- Step 3: Generate a change plan (Planning Task)
  - Identify the change plan based on Diagnosis **Plan\_Mapping\_Adaptation**
  - Fetch the plan **Plan\_Mapping\_Adaptation\_steps**
- Step 4: Execute the change plan (Execute task)
  - generate transformation based on the new mapping file

This preliminary version of DEAM's prototype demonstrates, in practical way, the usefulness of the idea of self-healing for data exchange process. As a matter of fact, and as we can clearly see, introducing a change in the source schema resulted in faulty state of the data exchange process which was recovered completely and autonomously (without user intervention) by the autonomic manager (Self-Healing Agent). This experiment results can be considered as encouraging, however we are still working on some interesting challenges such as:

- Developing a new approach/algorithm for Schema Matching in order to replace the Schema Matching tool (COMA3.0) used currently in DEAM prototype.
- Dealing with Complex Mapping problem (Complex mappings map a set of attributes in the source to a set of attributes in the target [19]).

In the next section, we briefly introduce the implementation of our novel Agent-based Modelling and Simulation approach for the Schema Matching problem, called "Schema Matching Agent-based Simulation" (SMAS). We are planning to use our new ABMS approach for Schema Matching to replace the

Schema Matching tool (COMA3.0) in the next release of DEAM.

#### IV. ABMS APPROACH FOR SCHEMA MATCHING

Our solution (SMAS) aims at generating high quality schema matchings with minimum uncertainty. As far as we know, there is no previous literature describing a solution approaching the Schema Matching and Mapping problem under the angle of Agent-Based Modelling and Simulation.

In a nutshell, our idea is to model Schema Matching and Mapping process as interactions between a set of agents (schema attribute agents) within a self-organized environment. Each schema attribute agent belongs to a group (source or target schema attribute) and represents a single schema attribute for which we need to perform the schema matching.

The simulation starts with two groups of unmatched Attribute Agents (source and target groups), which have as main goal the creation of a relation, based on similarity measures, with one of the agents belonging to the other group (for instance, if the agent belongs to the source group it should find the best match in the target group and vice versa).

During each tick of the simulation run, each agent executes behaviors based on randomness (stochastic):

1. Similarity Calculation based on a similarity measures selected randomly from a similarity measures list.
2. Similarity Scores aggregation based on aggregation functions selected randomly from an aggregation function list (MAX, AVERAGE, WEIGHTED).
3. Similarity score validation based on generated random threshold value (within interval)

The simulation ends when each agent has reached a consensus, about its candidate matching, with another agent (both agents are referring to each-other as candidate matching).

SMAS was implemented in Java using the open source ABMS framework "Repast Symphony (2.1)" [20] and the open source framework for Text Similarity "DKPro Similarity (2.1.0)" [21].

We performed preliminary tests of our agent-based model for schema matching for which the results<sup>1</sup> obtained were satisfactory and confirmed our initial intuition about the fact that ABMS can be an efficient and a well suited paradigm for resolving the schema matching problem.

As opposed to deterministic solutions for schema matching, such as COMA, the nondeterministic and stochastic nature of our agent-based simulation increases the confidence in the quality of the matching results. Despite the fact, that the agent's behaviors are based on randomness (e.g. during the similarity calculation), our model can produce, most of the time, the right matchings at the end of each simulation run.

<sup>1</sup>Link for uploaded schema matching simulation animations as well as output files:  
[http://www.researchgate.net/publication/262181441\\_Schema\\_Matching\\_Agent-based\\_Simulation\\_\(SMAS\)\\_Test\\_animation](http://www.researchgate.net/publication/262181441_Schema_Matching_Agent-based_Simulation_(SMAS)_Test_animation)

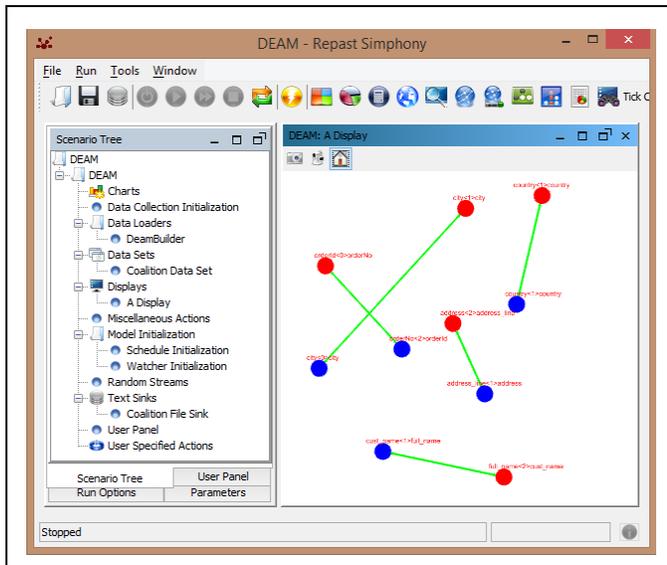


Figure 3. SMAS runtime

In the next releases of SMAS, we are planning to add statistical analysis in order to produce the final matching results based on multiple simulation runs data (batch mode).

#### CONCLUSION AND FUTURE WORK

In this work, we described our approach to make data exchange processes resilient to faults resulting from evolving schemas. This preliminary version of DEAM's prototype is the first step to demonstrate that the idea of self-healing for data exchange process is a promising approach. Also, we briefly introduced the implementation of our novel Agent-based Modelling and Simulation approach for the Schema Matching problem called "Schema Matching Agent-based Simulation" (SMAS). In the next release of DEAM, we are planning to use our new ABMS approach for Schema Matching as replacement for the Schema Matching tool (COMA3.0).

#### REFERENCES

[1] H. Assoudi et H. Lounis, « Self-Healing Data Exchange Process under Evolving Schemas: A New Mapping Adaptation Approach Based on Self-Optimization », in High-Assurance Systems Engineering (HASE), 2011 IEEE 13th International Symposium on, 2011, p. 188–190.

[2] M. Hartung, J. Terwilliger, et E. Rahm, « Recent advances in schema and ontology evolution », in Schema matching and mapping, Springer, 2011, p. 149–190.

[3] M. Arenas et L. Libkin, « XML data exchange: consistency and query answering », p. 13-24.

[4] P. Bohannon, E. Elnahrawy, W. Fan, et M. Flaster, « Putting context into schema matching », p. 307-318.

[5] A. Doan, P. Domingos, et A. Halevy, « Reconciling schemas of disparate data sources: A machine-learning approach », p. 509-520.

[6] R. McCann, B. AlShebli, Q. Le, H. Nguyen, L. Vu, et A. Doan, Mapping maintenance for data integration systems. VLDB Endowment, 2005.

[7] A. Gal, « Evaluating Matching Algorithms: the Monotonicity Principle », 2003.

[8] E. Rahm, « Towards large-scale schema and ontology matching », in Schema matching and mapping, Springer, 2011, p. 3–27.

[9] Y. Velegrakis, R. J. Miller, et L. Popa, « Adapting mappings in frequently changing environments », 2003.

[10] Y. Velegrakis, R. J. Miller, et L. Popa, « Preserving mapping consistency under schema changes », VLDB J., vol. 13, no 3, p. 274-293, 2004.

[11] H. Paul, « Autonomic computing: IBM's perspective on the state of information technology », also known as IBM's Autonomic Computing Manifesto. IBM, 2001.

[12] J. O. Kephart et D. M. Chess, « The vision of autonomic computing », Computer, vol. 36, no 1, p. 41-50, 2003.

[13] D. Ghosh, R. Sharman, H. Raghav Rao, et S. Upadhyaya, « Self-healing systems—survey and synthesis », Decis. Support Syst., vol. 42, no 4, p. 2164–2185, 2007.

[14] T. De Wolf et T. Holvoet, « Towards autonomic computing: agent-based modelling, dynamical systems analysis, and decentralised control », in Industrial Informatics, 2003. INDIN 2003. Proceedings. IEEE International Conference on, 2003, p. 470–479.

[15] C. M. Macal et M. J. North, « Tutorial on agent-based modelling and simulation », J. Simul., vol. 4, no 3, p. 151–162, 2010.

[16] A. Pokahr, L. Braubach, et K. Jander, « The Jadex Project: Programming Model », in Multiagent Systems and Applications, Springer, 2013, p. 21–53.

[17] D. Aumueller, H.-H. Do, S. Massmann, et E. Rahm, « Schema and ontology matching with COMA++ », in Proceedings of the 2005 ACM SIGMOD international conference on Management of data, 2005, p. 906–908.

[18] J. Coleman, T. Lau, B. Lokhande, P. Shum, R. Wisniewski, et M. P. Yost, « The autonomic computing benchmark », Dependability Benchmarking Comput. Syst., vol. 72, p. 1, 2008.

[19] Z. Bellahsene, A. Bonifati, et E. Rahm, Schema matching and mapping, vol. 20. Springer, 2011.

[20] M. J. North, E. Tatara, N. T. Collier, et J. Ozik, « Visual agent-based model development with repast simphony », Tech. rep., Argonne National Laboratory, 2007.

[21] D. Bär, T. Zesch, et I. Gurevych, « DKPro Similarity: An Open Source Framework for Text Similarity », in Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics: System Demonstrations, 2013, p. 121–126.

# Analysis, Design and Implementation of an Agent Based System for Simulating Connected Vehicles

Elahe Paikari, Behrouz H. Far

Department of Electrical and Computer Engineering, University of Calgary, AB, Canada, {Paikarie, Far}@ucalgary.ca

**Abstract-** PARAMICS traffic microsimulator is a popular simulator among universities and government agencies since it is capable of representing many parts of the world's street maps and designed to handle scenarios ranging from a single intersection to a congested freeway, or the modeling of a complete traffic system. However, it lacks the ability of simulating Connected Vehicle (CV) system and its applications of the Intelligent Transportation System (ITS) through designated traffic simulation network. In this study, we utilized the Multi Agent System Engineering (MaSE) methodology, step by step, to model CV as a Multiagent System (MAS). We implemented the MaSE artifacts as extensions for the PARAMICS using two APIs (Application Programming Interface) to add the ability to simulate CV systems. In this paper we provide detailed explanation of the MAS design and at the end introduce two experiments, made based on this research, as the case studies to evaluate the proposed CV system for estimating and improving traffic safety and mobility parameters in the network.

**Keywords –** Traffic simulator; PARAMICS; Connected Vehicles; Intelligent Transportation System; Application Programming Interface; MaSE methodology

## I. INTRODUCTION

Chain collisions can be potentially avoided, or their severity lessened, by reducing the delay between the time of an emergency event and the time at which the vehicles behind are informed about it [1]. One way to provide more time to drivers to react in emergency situations is to develop ITS applications to create connected vehicle (CV) systems using emerging wireless communication technology. The primary benefit of such communication will be to allow the emergency information to be propagated among vehicles much quicker than a traditional chain of drivers reacting to the brake lights of vehicles immediately ahead.

CV research has the potential to improve safety, reduce congestion, benefit the environment and enhance traveler services by enabling vehicles to wirelessly communicate with roadside infrastructure, nearby vehicles, cell phones, and other mobile devices [2, 3]. The basic connected vehicle concept is the establishment of a networked environment between vehicles and roadway infrastructure (V2I) and among vehicles (V2V) through wireless communications. With connected vehicles, V2V, V2I and other services are integrated to work together [4].

V2V is a main component of vehicular communication systems and allows detailed information to be exchanged among the individual vehicles. V2V systems may lead to preventing road collisions and alert motorists and decreasing travel time and crash risk in traffic networks in case of a reasonable penetration rate, i.e. percentage of equipped vehicles [5]. In Vehicle Infrastructure systems (V2I), vehicles

exchange information with roadside beacons, which are fixed. These beacons act as an interface between the vehicle network and external networks [5]. V2I takes an effective role in improving road safety and mobility through multiple in-vehicle and roadside units' technologies. The information collected by these roadside components will be shared with the transportation infrastructure operators who will in turn adjust the operation of various control devices (e.g. VMS) to maximize the efficiency of the transportation system and improve safety in response to traffic demand and road condition. VMS (Variable Message Sign) is an electronic message board located close to a roadway. It represents a cost-effective mechanism for disseminating information to drivers unequipped to receive guidance about advisory speed or incidents ahead. Hence, unequipped drivers can be directly influenced through VMS messages.

In this research, extensions of PARAMICS for CV have been designed using MaSE methodology and implemented by two APIs. API #1 creates predefined [5] or random incidents [6] in the network by taking user inputs on the probabilities of collisions and weather-related incidents. API #2 simulates the broadcasting of V2V and V2I communication in the network.

## II. BACKGROUND

Microscopic traffic simulation models are becoming increasingly important tools in modelling complex transport networks and evaluating various traffic management alternatives in order to determine the optimum solution for traffic problems that cannot be studied by other analytical methods. In the transportation simulation field there is a general agreement that micro-simulation, i.e., a computational resolution down to the level of individual travelers, may be the only answer to a wide variety of problems. Most existing commercial programs, such as PARAMICS [7] and VISSIM [8], provide ways of simulating traffic models which usually require quite computer related knowledge and coding efforts. Since they are closed-source, they provide API functions through which underlying simulation logic can be changed. Thus, researchers can simulate traffic models other than built-in models through these API functions.

Numbers of researchers investigated on providing comparative and functional evaluations on various traffic simulators [9-13]. From all of the studied simulators, the models AIMSUN [14], PARAMICS, and VISSIM are found to be suitable for congested arterials and freeways, and integrated networks of freeways and surface streets. Also, these models are potentially useful for ITS applications. While these packages have many similarities, each has its own specific characteristics that make it more or less suitable for certain modelling purposes.

### III. ANALYSIS

In this research, PARAMICS was selected for its outstanding illustration capabilities for a potential demonstration purpose. Due to its scalability, capability, its use in previous works examining variable speed limits and real-time crash risk [15], proven background on freeways, urban roads and VMS concept, which is used in this study, PARAMICS is able to simulate ITS applications required for implementing and evaluating CV systems properly and allows users to extend and test their own traffic control strategies. Given that, with the use of API, PARAMICS satisfies the transmission/generation requirements of warning messages.

Although, PARAMICS is beneficial over other types of microsimulation packages, it lacks the ability of simulating CV systems and their applications in the designated traffic simulation network. Moreover, simulated vehicles in PARAMICS strictly operate under car-following and lane changing models, which lead to an ideal error-free driving world, and therefore zero incidents happen. To allow for testing CV applications, it was necessary to manipulate the model in a way that a predefined incident or various stochastic incidents such as running red light, rear end collision, and weather caused incidents can be reproduced in the simulation world.

The MaSE methodology is a detailed and top-down approach which will cover information needed in models. Also it is a full-lifecycle methodology for analyzing, designing, and developing heterogeneous MAS. MaSE views MAS as a further abstraction of the object-oriented paradigm where agents are specialized objects and it builds upon well-founded object-oriented techniques and applies them to the specification and design of MAS.

The MaSE Analysis phase consists of three steps: Capturing Goals, Applying Use Cases, and Refining Roles. The Design phase has four steps: Creating Agent Classes, Constructing Conversations, Assembling Agent Classes, and System Design. A major strength of MaSE is the ability to track changes throughout the process [16]. Every object created during the analysis and design phases can be traced forward or backward through the different steps to other related objects. For instance, a goal derived in the Capturing Goals step can be traced to a specific role, task, and agent class. Likewise, an agent class can be traced back through tasks and roles to the system level goal it was designed to satisfy.

In the MAS, agents coordinate their actions via conversations to accomplish individual and community goals instead of objects whose methods are invoked directly by other objects. Since our proposed system consists of several modules, such as CV and infrastructures, which send messages to each other in order to connect and transmit warning messages, we selected MAS as the approach to the system. MAS composed of multiple interacting intelligent agents used to solve issues which are hard for an individual agent or module to solve. Our proposed CV model multi-system includes 4 independent agents, which can offer different functions and accomplish CV simulation. In this modeling process, there will be V2V, Non V2V, Database and PARAMICS agents.

The overall approach in the Analysis phase is defining the system goals from a set of functional requirements and then defining the roles necessary to meet those goals. While a direct mapping from goals to roles is possible, MaSE suggests the use of Use Cases to help validate the system goals and derive an initial set of roles.

#### A. Capturing goals

The proposed system consists of four components: CV, non-CV, RSU (Road Side Units), and VMS. CV is equipped with a wireless system to create V2V and V2I connection. RSU, the infrastructure placed within the network, is capable of collecting information about traffic conditions and communicate with the CV located within a Dedicated Short Range Communication (DSRC) range, and sending this information to the VMS. VMS located in network to show warning messages and traffic information collected by RSU to communicate with non-CV. Non-CV do not have the equipment to connect with other vehicles or RSUs.

In the first step, we created a set of functional requirements:

- 1- Two types of vehicles are created in the network (CV/Non-CV).
- 2- CVs are connected to each other and RSUs.
- 3- Non-CVs have a connection with RSUs.
- 4- RSUs and VMS are linked.
- 5- If a vehicle is in accident, its flag is set to 1.
- 6- All the CVs and RSUs are informed about the accident.
- 7- Non-CVs are notified about the accident via VMS.
- 8- Consequently, some vehicle might change their route or speed.

From the requirements these goals are extracted:

- 1- Generate vehicles (CV/Non-CV) and infrastructures (RSU/VMS).
- 2- Create connections.
- 3- Transmit messages.
- 4- Change route/speed.

Next, the goals are analyzed and put into a hierarchical form. A Goal Hierarchy Diagram is a directed graph where the nodes represent goals and the arcs define a sub-goal relationship. The overall system goal is placed at the top of it, which is simulating CV as an independent agent added to PARAMICS module. Once a basic goal hierarchy is in place, goals may be decomposed into new sub-goals and each sub-goal must support its parent goal. System goals are depicted in a goal hierarchy diagram, as shown in Fig. 1.

#### B. Applying Use Cases

The objective of the Applying Use Cases step is to capture a set of use cases from the initial system context and create a set of Sequence Diagrams to help the system analyst identify an initial set of roles and communications paths within the system. Use cases of our system are drawn from the system requirements provided before and describe sequences of events that define desired system behavior; they are examples of how the system should behave. By creating use cases we intended to identify paths of communication. Fig. 2 shows the sequence diagram and how each goal can be accomplished.

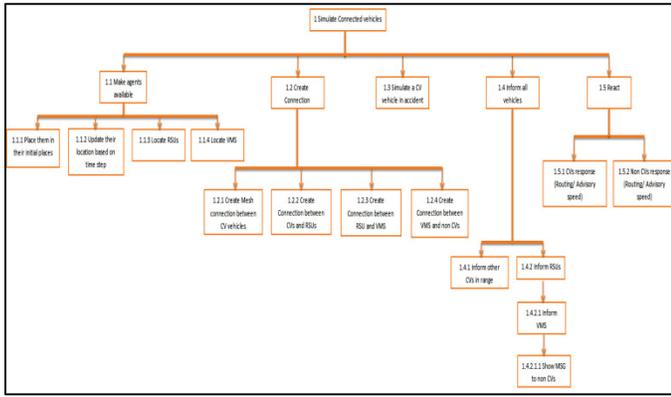


Figure 1. Goal Hierarchy Diagram

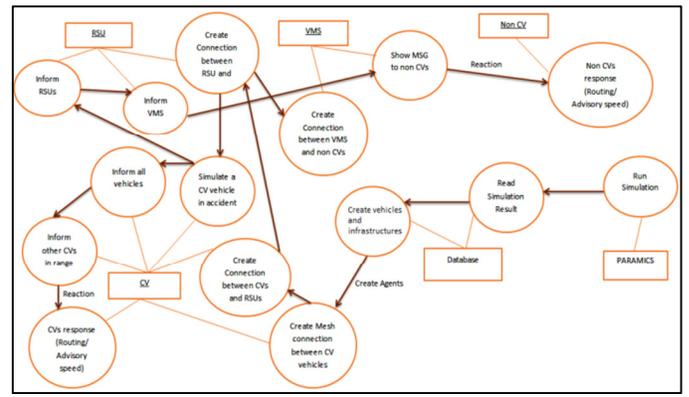


Figure 3. System Role Model

### C. Refining Roles

The purpose of the Refining Roles step is to transform the Goal Hierarchy Diagram and Sequence Diagrams into roles and their associated tasks, which are more suitable for designing MAS. Role definitions are captured in a MaSE Role Model, which includes information on interactions between role tasks and is more complex than traditional role models [17]. Fig. 3 illustrates system role model diagram that shows CV systems' roles and behaviours in PARAMICS.

The concurrent task model provides a built in timer activity. We build an initial Concurrent Task Model from the scenarios of creating CVs and creating accidents by taking the sequence of messages sent or received by the roles of CV/non-CV and use them to create a sequence of corresponding states and messages. Concurrent tasks in our system are defined in Concurrent Task Models Fig. 4 and are specified as finite state automata, which consist of states and transitions.

## IV. DESIGN

There are four steps to the designing a system with MaSE. In the first step we assigned roles to the 4 agent types in the system to create Agent Classes. In the second step, Constructing Conversations, the actual conversations between agent classes are defined while in the third step, assembling Agents Classes, the internal architecture and reasoning processes of the agent classes are designed. Finally the actual number and location of agents in the deployed system are defined. Each of these steps is discussed below.

### A. Creating Agent Classes

In the Creating Agent Classes step of the Design phase, agent classes are created from the roles defined in the Analysis

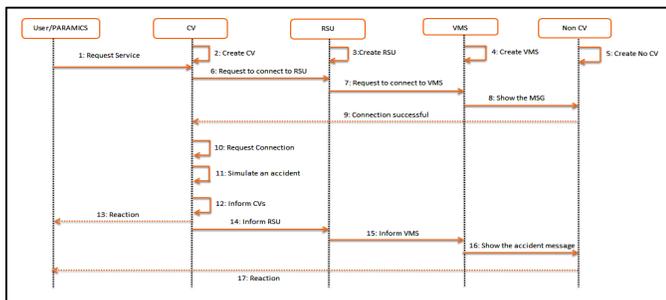


Figure 2. Sequence Diagram

phase. The result of this phase is an Agent Class Diagram, which illustrates the overall agent system organization consisting of agent classes and the conversations between them. At this point, the roles and tasks, which an agent class must play, are determined. The conversations, that an agent class must participate in, are derived from the external communications of the agent roles. The Agent Class Diagram is the first design object in MaSE that shows the entire MAS in a way it can be implemented. The CV model MAS mainly defines 4 agent classes as show in Fig. 5.

### B. Constructing Conversations

A MaSE conversation defines a coordination protocol between two agents. The Communication Class Diagram, as shown in Fig. 6, is similar to a Concurrent Task Model and defines the conversation states of V2V and non-V2V as the two participant agent classes. This conversation is about constructing connection between CVs and RSUs. CV, the initiator, begins the conversation by sending “create connection” as the first message. When the agent receives the message, it checks if CV is in range of corresponding RSUs. Only the vehicles which their distance from RSUs fall into DSRC range can be linked with them. Otherwise, the request will be declined and RSU cannot get the message from the CV and show it on VMS.

### C. Assembling Agents

During this step, the internals of agent classes are created. This is accomplished via two sub-steps: defining the agent architecture and defining the components that make up the architecture. Components consist of a set of attributes, methods, and, if complex, may have sub-architecture. Internal component behaviour may be represented by formal operation definitions as well as state-diagrams that represent events passed between components. Basically, each task from each role, played by an agent, defines a component in the agent class. Fig. 7 shows agents' architecture of CV model multi agent system.

### D. System Design

The final step of the MaSE methodology takes the agent classes defined previously and instantiates actual agents using Deployment Diagram to show the numbers, types, and locations of agents within a system. The concept of instantiating agents from agent classes is similar to

instantiating objects from object classes in object-oriented programming. Deployment Diagrams describe a system based on agent classes defined in the previous steps of MaSE. Strength of MaSE is that a designer can make these modifications after designing the system organization, thus generating a variety of system configurations. System Deployment Diagram includes 5 agents, there into, V2V Agent and DB1 Agent run in the same physical node, Non V2V Agent and DB2 Agent in one node and PARAMICS Agent run in the other physical node, as shown in Fig. 8. PARAMICS agent acts as the PARAMICS software itself where the other agent classes should be implemented and communicate with it. The Connected vehicles and No Connected Vehicles platforms should be added to the simulation platform to apply CV system as the extension to the software. This is done by programming APIs.

### I. IMPLEMENTATION

By using MaSE methodology, the essential agents, relationships and conversations between them, and the particular parameters of proposed CV system were identified. Moreover, MaSE specified how every agent should act in possible scenarios (Figure 2) to eventually reach the goals and sub-goals defined in Analysis phase (Figure 1). This specified design is implemented by adding two APIs to the PARAMICS programmer module. In API #1, the connection between PARAMICS agent to V2V and Non-V2V agents was created through the use of Database. However, the main duty of API#1 is to generate predefined or randomly [6] incidents for running test case scenarios.

Most of the proposed design by MaSE methodology is used in API #2, where both V2V and No-V2V agent classes and their relevant agents (i.e. CV, non-CV, RSU, VMS) were created (Figure 5). Additionally, the roles and tasks for each agent class and the conversations between these agent classes were programmed, such as the example conversation showed in Figure 6. It specifies the procedure of creating connection and message transmission between CVs and RSUs when they are in predefined DSRC range, and then showing the message context on the VMS. API #2 consists of 2 packages: V2V package in which V2V agent class was implemented and V2I package where the Non-V2V agent class were created. API #2 attempts to mitigate or to minimize the effect of the generated incidents by creating V2V and V2I communications.

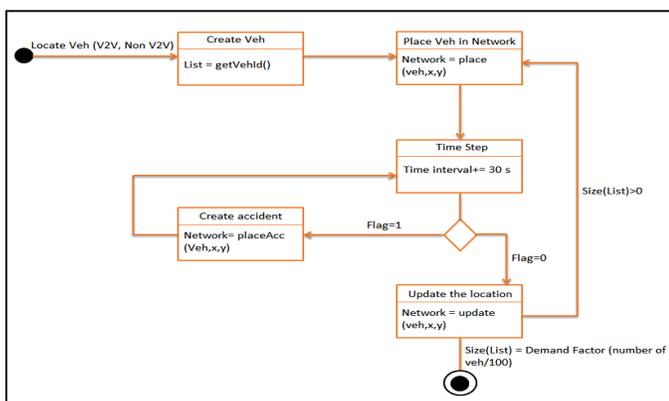


Figure 4. Concurrent Task Diagram

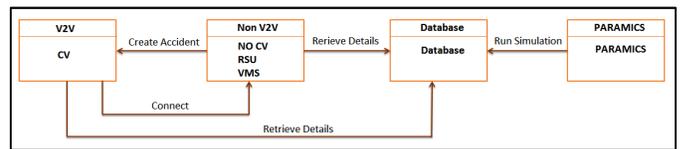


Figure 5. Agent Class Diagram

Once incidents were created (randomly or predefined) applying API #1, the second API, API #2, accordingly simulates the broadcasting of V2V and V2I communication in the network and models its effects on drivers' behavior. This API was developed to allow vehicles communicate with each other and infrastructures to apply the connection and conversations between V2V and Non-V2V agent classes. When a CV is involved in an incident, API #2 turns the vehicle's color to red to indicate this condition. This allows users to visually pick out which vehicle on the studied network was involved in the incident. API #2 then calculates the distance of surrounding CVs, notifies them of the incident ahead and provides an advisory speed. At this point, all notified CVs will have an improved awareness and a decreased aggressiveness (vehicle parameters that programmed intentionally). These two modules will be set randomly for different CV. In other words, their awareness ranges between 6 and 9, whilst their aggression rate will be between 1 and 4 (the threshold of awareness module is 1-9).

API#2 allows CVs to communicate with the RSUs. When CVs encounter an incident, they send a message to the nearest RSU along the road and transmit the location and the type of the incident to the unit. The corresponding RSU will forward aforementioned information to the control center. The control center defined as a decision maker chooses which VMS should be enabled to reflect the message to the upstream and downstream vehicles and what will be the context of message. Usually the message contains warning information for drivers to let them know that there is a collision ahead alongside an advisory speed. The advisory speed is lowered in increments as the distance from collision decreases and it is increased for downstream of traffic to let the congestion around the accident be cleared easily.

API #1 also retrieves and stores the simulated vehicles' information from PARAMICS. Vehicle information storage is completed through a vehicle map. Each vehicle's information is stored in the Vinfo struct. Vinfo struct includes information related to vehicles: Float x, Float y, Float z, Int ID, Float

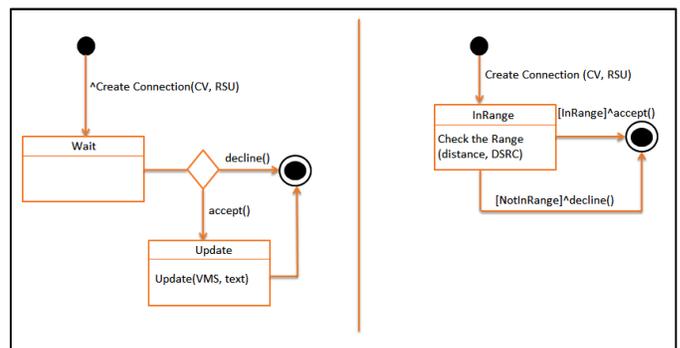


Figure 6. Create Connection conversation initiator and responder

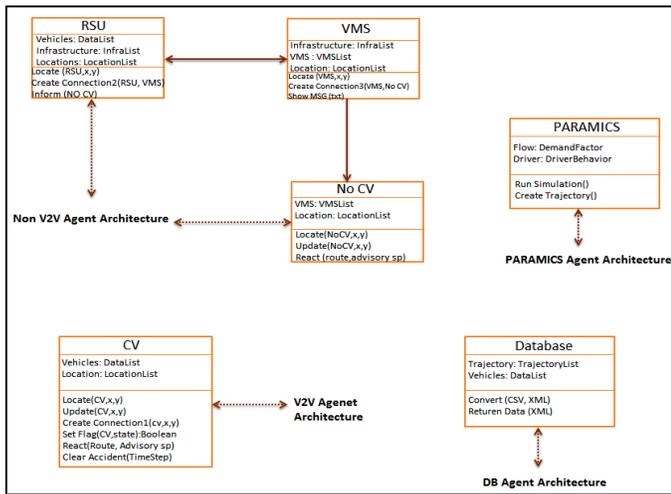


Figure 7. System architecture for agent class

bearing, Float gradient, Float length, Float width, Int age, Bool incident, Int usertag, Bool stopped, Bool collide\_stop, Float stop\_time, Float distance, VEHICLE \*vpoint, LINK \*link, Int lane.

API #2 retrieves CV information from the Vinfo struct. The retrieved information includes coordinate information for calculation of the DSRC range. The class of functions involved in creating incidents, information storage, information retrieval and sending V2V and V2I messages will be described in the following sections. The developed APIs along CV agents is shown in Fig. 9.

#### A. API #1

API #1 creates incidents and requires a set of functions to extract vehicle information from the network, to check for incidents that have occurred, to identify and register any incidents, and finally to create a readable database of vehicles. A majority of its classes of functions are described below.

**Qpx\_NET\_postOpen** is a PARAMICS extension (prefix qpx) built-in function. When a model is first opened in PARAMICS, the postOpen function is called. The total number of links and lanes in the network is recorded by its functions and vehicles and infrastructures are generated in the traffic network.

**Qpx\_NET\_second**, its functions are executed every second when the simulation is running. It is essentially the core of the program. The subroutine includes all of the functions in API#1.

**Qpx\_VHC\_arrive**, its functions remove the vehicle from the vehicle map when a vehicle reaches its destination zone.

**InformationExtraction** sets up the vehicle map. The vehicle map is where vehicle information is stored. API #2 retrieves vehicle information from the vehicle map.

**Accident generation** determines when an incident should occur. It can be a predefined incident or it can be several numbers of incidents based on the probability that user provided when the model is first loaded.

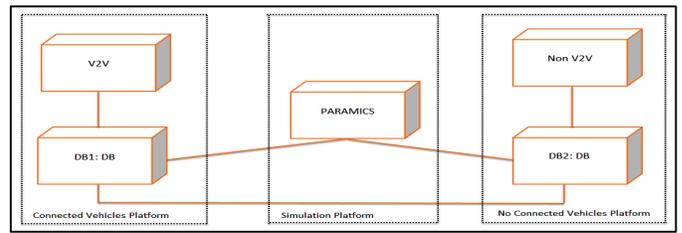


Figure 8. System Deployment Diagram

#### B. API #2

API #2 reads vehicles' information in the vehicle map and stores all V2V enabled vehicles in a V2V map. It then searches for occurred incidents in the network and sends warning messages to surrounding vehicles and the nearest RSUs fall in DSRC protocol. These messages increase driver awareness and enable them to avoid a consequent collision. When the incident has been totally cleared, the situation resumes to the initial state.

**Qpx\_NET\_timeStepPostLink** checks every link and adds CVs to the V2V map and infrastructures to the network. The V2V map consists only of CVs. It updates vehicle positions in the V2V map every time step and checks the incident flag for each vehicle. If there has been an accident, it calls the Send Message class of functions. If accident has been cleared, it gets back to the initial state.

**SendMessage**, when a vehicle is involved in an accident, its Vinfo struct is sent to the sendMessage. It iterates through the V2V map and calls calculateDistance. If any CV is within the DSRC range (1000m) of the vehicle involved in an accident, that vehicle is signaled of the danger ahead. With the increased driver awareness, the vehicle does not partake in the same incident. Based on the DSRC, CV in incident sends message to RSU, RSU sends it to control center and control center decides about the advisory speed and which beacons should show that. It should be noted that DSRC can be changed accordingly for experimenting its effects on CV system.

**CalculateDistance** receives vehicles' Vinfo struct classes, calculates and returns the distance between the two V2V enabled vehicles. It can also calculate and returns the distance between vehicle and RSU.

## II. CASE STUDY

To test the ability of the CV model during road hazard, two studies have been assessed [5, 18]. Evaluation scenarios (Fig.2) were developed for the morning AM peak traffic from 7:00 AM to 8:10 AM. The proposed model in PARAMICS was used to study the impact of deploying CV on an 8-km southbound section of Deerfoot Trail, Calgary, Alberta.

In these case studies, V2V and V2I communications were evaluated under various load conditions. Moreover, advisory speed recommendation and re-routing guidance were implemented to recommend the optimum treatments, reduce rear-end and lane change crash risks and decrease travel time.

In the first case study, several incidents happen in the network based on the probability provided by user in the start

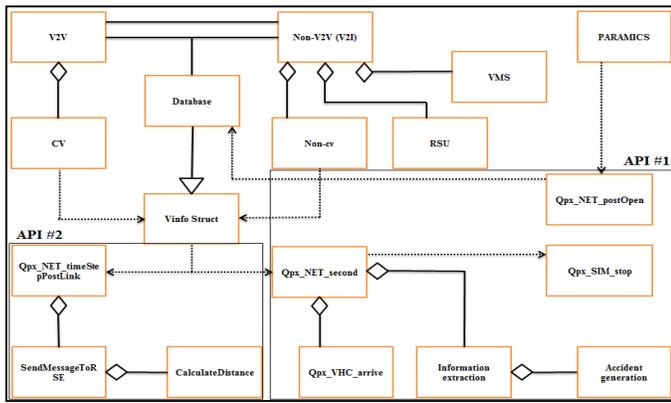


Figure 9. Implemented APIs using CV agents

of the simulation to evaluate CV system in hazard condition. The advisory speed recommendation was only for upstream traffic of the location of accident where speed differences between upstream and downstream vehicles were high. Also, safety and mobility indices were measured where the percentage of CVs changes but the other elements including demand factor remain constant.

Although the first case study found the improvement in travel time and crash likelihood, we did not develop the DSRC range as a factor for distributing messages in V2I module and we only focused on upstream traffic. In the second case study, an incident that blocks one lane was generated at predefined location to precisely study the impact of CV applications. Also, we demonstrate effect of considering DSRC, re-routing guidance and advisory speed for upstream and downstream traffic, where we evaluated different demand loading, DSRC range and penetration rate of CVs. The results show that CV technology can enhance traffic safety and mobility in freeways, if the percentage of CVs is significant (e.g. 30-40%) and the CV technology is accompanied by advisory speed reflected on VMS on both upstream and downstream of the incident location using DSRC range.

### III. CONCLUSION

In this study we developed, implemented and demonstrated a DSRC based V2Vassisted V2I traffic information system for estimating and disseminating traffic safety and mobility parameters. One of the main research priorities of the ITS is to facilitate wireless communication between vehicles and infrastructure so that traffic safety information data can be exchanged. We used PARAMICS microsimulator as the traffic network simulator to implement CV systems. Since it does not have the ability to simulate V2V and V2I connections, we designed our proposed CV scenarios using MaSE methodology to view the CV modules as the MAS. By using API, we implemented our system as the extension to the PARAMICS. Two studies have been made to evaluate this system and the overall results showed that applying CV in Calgary's freeways will improve the safety and mobility applications.

The future work would be enhancing the connected vehicles study by providing a simulation environment that combines the capabilities of a traffic network simulator

together with wireless communication functionalities in order to optimize the communication (frequency of information update), provide a cost effective alternative through traffic microscopic and wireless communication simulators to detect failure or latency of communication through testing of V2V and V2I communications systems based on different communication and DSRC ranges.

### REFERENCES

- [1] Q. Xu, R. Segupta, D. Jiang, and D. Chrysler, "Design and analysis of highway safety communication protocol in 5.9 GHz dedicated short range communication spectrum," in *Vehicle Technology Conference, 2003. VTC 2003-Spring. The 57th IEEE Semiannual*, 2003, pp. 2451-2455.
- [2] P. Farradyne, "VII architecture and functional requirements," *Report prepared for the ITS Joint Programme Office, Federal Highway Administration, US Department of Transportation*, 2005.
- [3] B. A. Hamilton, "Vehicle infrastructure integration (VII) concept of operations," ed. Federal Highway Administration, U.S. Department of Transportation, Washington, DC., 2006.
- [4] G. Abu-Lebdeh, "Exploring the potential benefits of intellidrive-enabled dynamic speed control in signalized networks," in *Transportation Research Board 89th Annual Meeting*, 2010.
- [5] E. Paikari, L. Kattan, S. Tahmasseby, and B. H. Far, "Modeling and simulation of advisory speed and re-routing strategies in connected vehicles systems for crash risk and travel time reduction," in *Electrical and Computer Engineering (CCECE), 2013 26th Annual IEEE Canadian Conference on*, 2013, pp. 1-4.
- [6] L. Kattan, M. Moussavi, B. Far, C. Harschnitz, A. Radmanesh, and S. Saidi, "Evaluating the Potential Benefits of Vehicle to Vehicle Communication (V2V) under Incident Conditions in the PARAMICS Model," in *Proceedings of the 13th International IEEE Conference on Intelligent Transportation, Madeira, Portugal*, 2010.
- [7] Q. Paramics, "The paramics manuals, version 6.6. 1," *Quastone Paramics LTD, Edinburgh, Scotland, UK*, 2009.
- [8] P. P. T. V. AG, *VISSIM 5.40 User Manual*: epubli GmbH, 2012.
- [9] F. Choa, R. T. Milam, and D. Stanek, "CORSIM, PARAMICS, and VISSIM: What the manuals never told you," in *Ninth TRB Conference on the Application of Transportation Planning Methods*, 2004.
- [10] S. L. Jones, A. J. Sullivan, N. Cheekoti, M. D. Anderson, and D. Malave, *Traffic simulation software comparison study* vol. 2217: University Transportation Center for Alabama, 2004.
- [11] S. Panwai and H. Dia, "Comparative evaluation of microscopic car-following behavior," *Intelligent Transportation Systems, IEEE Transactions on*, vol. 6, pp. 314-325, 2005.
- [12] P. Hidas, "A functional evaluation of the AIMSUN, PARAMICS and VISSIM microsimulation models," *Road and Transport Research*, vol. 14, 2005.
- [13] J. W. Shaw and D. H. Nam, "Micro-Simulation: Freeway System Operational Assessment and Project Selection in Southeastern Wisconsin: Expanding the Vision," in *Transportation Research Board 81st Annual Meeting, Washington, DC*, 2002.
- [14] AIMSUN, NG. "Manual 6.0." Transport Simulation Systems (2009).
- [15] Y. Gardes, A. D. May, J. Dahlgren, and A. Skabardonis, "Freeway calibration and application of the PARAMICS model," in *81st Annual Meeting of the Transportation Research Board, Washington, DC*, 2002.
- [16] A. Scott and M. Wood, "Developing multiagent systems with agenttool," in *Intelligent Agents VII Agent Theories Architectures and Languages*, ed: Springer, 2001, pp. 46-60.
- [17] E. A. Kendall, "Agent roles and role models: New abstractions for intelligent agent system analysis and design," in *International Workshop on Intelligent Agents in Information and Process Management*, 1998.
- [18] E. Paikari, S. Tahmasseby, and B. H. Far, "A Simulation-based Benefit Analysis of Deploying Connected Vehicles Using Dedicated Short Range Communication," accepted in *Intelligent Vehicles Symposium, 2014 IEEE Intelligent Transportation Systems Society*, 2014.

# Towards a Taxonomy of Services for Developing Service-Oriented Robotic Systems

Lucas Bueno R. Oliveira<sup>\*†</sup>, Fernando S. Osório<sup>\*</sup>, Flavio Oquendo<sup>†</sup>, and Elisa Yumi Nakagawa<sup>\*</sup>

<sup>\*</sup>Dept. of Computer Systems, University of São Paulo - USP, São Carlos, SP, Brazil

<sup>†</sup>IRISA Research Institute, University of South Brittany, Vannes, France

Email: {buenolro, fosorio}@icmc.usp.br, flavio.oquendo@irisa.fr, and elisa@icmc.usp.br

**Abstract**—Robotic systems have been increasingly adopted in several sectors of the society. To cope with this demand and diversity, researchers have investigated the Service-Oriented Architecture (SOA) to develop such systems. SOA promotes interoperability between software modules and heterogeneous hardware devices, and a better reusability and flexibility for robotic systems. However, due to the lack of a common understanding on how services for robotic systems should be designed, described and also classified, these services are sometimes difficult to be used in other projects, reducing the potential of reuse provided by SOA. The main contribution of this paper is to propose a taxonomy of services for robotic systems that was based on results of a systematic review, reference architectures, and knowledge of specialists. Results have pointed out that our taxonomy is an important element to organize different types of services, what can promote reuse and productivity in the development of robotic systems.

**Keywords:** *Taxonomy, Service-Oriented Architecture, Robotics.*

## I. INTRODUCTION

Robotics has currently become one of the most active fields for researchers and enterprise, significantly impacting the society and our daily lives [13]. Different types of robots have been developed and used in a wide variety of application areas, such as house cleaning, surveillance, and entertainment. In this perspective, the complexity of the robotic systems has been increasing, creating a considerable challenge for their development. To deal with this complexity and improve the productivity of the development, several researchers have adopted the Service-Oriented Architecture (SOA) to design robotic systems [10].

SOA is as a successful architectural style that uses services as the basic constructs to support the development of software systems [11]. Services are independent, self-contained, and well-defined software modules that can be dynamically composed to form more complex software solutions [11]. This architectural style intends to contribute with loosely-coupled systems, promoting reuse and productivity in the software development [11]. In robotics, SOA has been used to develop more flexible, scalable, and reconfigurable robotic systems built as a set of independent software modules. It has also been adopted to facilitate integration of heterogeneous hardware devices and reuse of complex software modules. In this perspective, companies and research institutes have created means to support the development of Service-Oriented

Robotic System (SORS), i.e., robotic systems designed according to SOA. Two well-known examples of environments to develop these systems are Robot Operating System (ROS) [14] and Microsoft Robotic Developed Studio (MRDS) [8].

Due to the relevance of SOA for the robotics, several SORS have been developed in recent years [10]. These works not only have contributed to the consolidation of SOA in the domain, but also have produced a considerable amount of services for SORS. However, most of these SORS were developed using their own structures, without a common understanding about how and which software modules should be provided as services. This lack of consensus has reduced the reusability of the services and hampered the discovery of these services by developers of SORS. In this sense, identification and classification of the types of services that can be used in the development of SORS would contribute to facilitate the discovery and reuse of services.

The main contribution of this paper is to present a classification of services for SORS. For this, we have established a taxonomy, which is a form of classification widely accepted in different domains, such as software architecture [4] and robotics [5]. This taxonomy is based on the SORS available in the literature and identified in a systematic review [10], as well as a set of reference architectures of the domain [6] and knowledge of experts in robotics. Results obtained from a survey with domain experts have shown that the proposed taxonomy is representative and can be used to successfully classify the robotics services, as well as provides a good support to discovery such services to their future reuse.

The remainder of this paper is organized as follows. Section II describes the establishment of the taxonomy. Section III presents the survey used to evaluate the taxonomy. Section IV discusses on the results obtained from the survey and the perspectives of application of the taxonomy. Finally, Section V presents our conclusion and future work.

## II. DESIGNING THE TAXONOMY OF SERVICES FOR SORS

To establish the taxonomy of services, we followed a well-defined, systematic set of steps, as illustrated in Figure 1. In short, in Step 1, we first elicited services from different sources of information. In Step 2, we grouped the

types (or sets) of services that have similar characteristics and purposes of use. In Step 3, these groups were described and organized in different abstraction levels, resulting in the types of services and the Robotics Services Dependency Stack (RSDS). Finally, in Step 4, the types of services and RSDS were evaluated in a survey applied to experts of robotics area. Following, each step used to establish our service taxonomy is presented in details.

### A. Step 1: Service Elicitation

In this step, we selected different information sources to elicit the types of services used to develop SORS. These sources encompassed both theoretical and practical views of the robotic system development. The three sources were: (i) a systematic review addressing 39 studies that report on the development of SORS [10]; (ii) a set of reference architectures that encompass knowledge of how to structure robotic systems [6]; and (iii) expertise and knowledge of experts on how to develop robotic systems. By investigating these sources, we obtained a broad set of services.

### B. Step 2: Service Analysis and Categorization

Given the services identified, we performed brainstorm meetings for analysis and classification. For this, we considered the expertise of specialists in software architecture, SOA, and robotics. These meetings were guided by reference architectures for robotics, since they contain modules and functionalities that should be considered in robotic systems. In this step, we have mitigated the following issues: (i) similar services with different names; (ii) services with the same name but providing different functionalities; (iii) services lacking cohesion; and (iv) modules that could be provided as services, but were not identified in the information sources. We also identified abstraction levels for the services. As a result, we obtained five groups of services: Device Driver, Knowledge, Task, Robotic Agent, and Application.

### C. Step 3: Taxonomy Establishment

Based on the identified groups of services, we proposed a taxonomy of services for SORS. This taxonomy is composed by two main parts: (i) RSDS that links groups of services and (ii) description of all types of services of these groups. To define the dependency stack, we adopted the overall structure

of the layered S3 reference architecture [1]. S3 is a widely accepted description of the relationships among services in SOA. Figure 2 presents RSDS and its layers, which represent the groups of services. In RSDS, less abstract services, for instance, services contained in Device Driver layer, provide functionalities for the higher ones, for instance, contained in Task layer. Besides that, services in lower layers are more fine grained and provide, for instance, software abstractions of hardware devices that can be used in different application domains. Services in higher level layers coordinate services in lower level layers to perform complete activities and are, therefore, more dependent of the application domain.

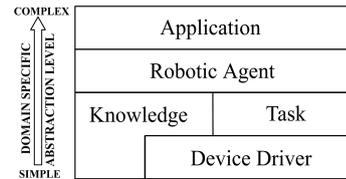


Fig. 2: Robotics Services Dependency Stack

Device Driver layer encompasses services that control hardware devices, providing their functionalities to the higher layers. The Knowledge layer represents services that manage information used by the robotic system to make its decisions. Supported by Device Driver and Knowledge services, the Task layer groups services that provide tasks of robotics (e.g., mapping or localization) according to different behaviors. The Robotic Agent layer represent services that encapsulate the system that controls a robot as a service (i.e., Robot as a Service - RaS), which performs tasks – using Task services – based on information gathered from Knowledge services. Finally, the Application layer encompasses services that coordinate one or more Robotic Agents to perform more complex activities, such as surveillance and entertainment.

Notice that none of these five layers are mandatory and it is possible to develop robotic systems without using the lower level services, by designing a monolithic robotics service. However, we discourage this practice, since it may reduce flexibility and potential of reuse. A detailed description of each group of services is presented as follows.

1) *Device Driver Services*: This group includes services that encapsulate hardware drivers. Device driver services

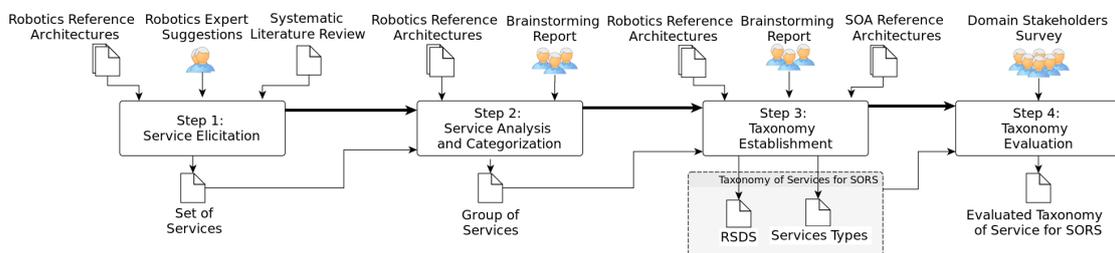


Fig. 1: Steps followed to establish the taxonomy of services

are used as a design solution to provide better integration among heterogeneous off-the-shelf resources. There are two types of service in this group: *Actuators* and *Sensors*. Sensor services encapsulate drivers that coordinate hardware devices used to ‘feel’ the environment where the robot is. Actuator services are responsible for controlling the hardware used to interact with the environment. The services of this group are illustrated in Figure 3 and described as follows. Notice that some devices available nowadays can combine more than one type of service, such as for Position and Orientation. Interfaces of these services are therefore a composition of interfaces of provided services.

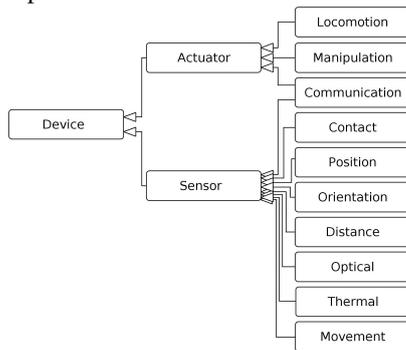


Fig. 3: Device Driver service group

**Locomotion:** is responsible for providing mobility to the robot. There are different examples of locomotion driver services, such as for wheels, joints, helix, and so forth.

**Manipulation:** allows the robot to manipulate or hold objects of the environment. Drivers for controlling arms and grippers are considered as manipulation device services.

**Communication:** plays the role of both actuator and sensor. This is due to the fact that a communication is often bidirectional, i.e., a robotic system can use this service to feel the environment or interact with it. Different types of services can be considered as communication services, such as drivers for multimedia, network, and radio frequency.

**Position:** is used to obtain information of where the robot is in the environment. A well-known example of position service is the GPS device driver service.

**Orientation:** provides information about the orientation of the robot in the environment. Inclinometer driver and compass driver are both examples of orientation services.

**Movement:** measures the distance traveled by the robot. Encoder driver can be considered as a movement service.

**Contact:** controls sensors that detect whether the robot is in contact with objects in the environment. Drivers for barrier and bumper are examples of contact services.

**Distance:** is used to measure distance between the robot and the objects inside the environment. Drivers for lasers and sonars are examples of distance measuring services.

**Optical:** converts images of the environment into digital information that can be processed. Drivers for stereo cameras can be considered as examples of optical sensor services.

**Thermal:** is used to measure temperature of objects inside the environment. Drivers for thermal cameras are well-known examples of thermal sensor services.

2) **Knowledge Services:** This group comprises services that are responsible for gathering, interpreting, storing, and sharing information that is necessary to perform tasks and control the robot as a whole. This information allows the robotic system to learn about characteristics of its environment and objects inside it. Knowledge services can use not only data from sensors but also semantic information from a wide range of sources, such as ontologies or machine learning datasets. Services that belong this group are presented in Figure 4 and detailed as follows.

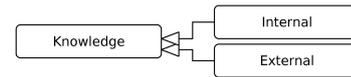


Fig. 4: Knowledge service group

**Internal:** is concerned with gathering, storing, and sharing information from inside the environment. This information can be obtained from both sensors and databases hosted in a back-end server. Information obtained from sensors is generally produced by Task services and then stored in internal knowledge service to be shared among robots. Two examples of internal knowledge service are: (i) a robot learning how to deal with a given object by accessing a service hosted in a back-end server; and (ii) a robot sharing with another one its optimal path for a given position.

**External:** is responsible for obtaining and interpreting general purpose information from sources that are outside of the environment. By using this type of service, a web-enabled robot can search, access, and obtain information from machine-readable content on the Internet or even ordinary web sites, portals, and wikis. These external sources allow the robot to learn procedures, semantic concepts, and relationships that were not considered during the design of the robotic system. An example of this type of service is an autonomous robot learning how to make pancakes using information from the web, as shown in [15].

3) **Task Services:** This group encompasses services that provide functionalities considered as fundamental tasks of robotics. Task services allow the robot to perform basic activities, such as move from a position to another. These services can be implemented according to different behaviors, i.e., a robot can move to another position either by following a wall or keeping the distance between walls. This group can be divided in seven main types, as presented in Figure 5. Details of such services are discussed as follows.

**Mapping:** encapsulates algorithms used to estimate and build representations of the environment where the robot is. There are two types of mapping services, those that use Metric Maps and those that use Non-metric Maps. Metric maps are 2D/3D representations that use coordinates – *Geometric* or in a *Grid* – to describe the real location of

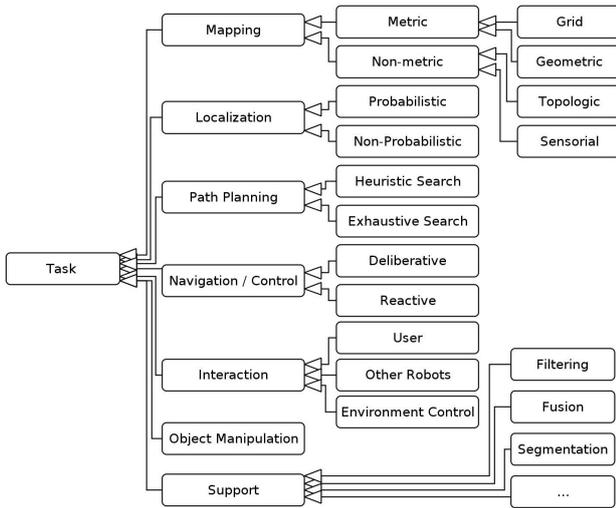


Fig. 5: Task service group

objects inside an environment. Non-metric maps are logical representations that can be *Topological* (e.g., an adjacency graph) or *Sensorial*, such as a sequence of images.

**Localization:** is used to estimate the position of the robot. For this, localization services use sensor data, previous knowledge about the environment, or both. There are two types of localization services: *Probabilistic* and *Deterministic*. Kalman filter [9] is a well-known algorithm that can be provided as a Probabilistic localization service to be executed and accessed by multiple robotic agents, as presented in [12]. Besides that, a Deterministic localization service can be any localization algorithm based solely on data from a GPS device or other type of position service.

**Path Planning:** is used to defined good (or optimal) paths between two or more positions in the environment. These services are often supported by mapping services and are developed based on two main strategies: (i) *Heuristic Search* (e.g., A-Star (A\*) [7]) and (ii) *Exhaustive Search* (any kind of services that provide a path examining all possible paths).

**Navigation/Control:** is used by the robot to navigate in the environment. Navigation can be done based on a predefined path or not. There are two types of navigation/control services: *Deliberative* or *Reactive*. A service based on Deliberative navigation performs a path according to a predefined plan. Reactive navigation service controls the robot based on their current sensory data, e.g., using a Potential Fields [3]. Notice that a robotic system can combine these two types of service to provide a more robust hybrid behavior, combining reactive and deliberative control.

**Interaction:** allows the robot to work together with the *Environment*, *Other Robots*, or to accomplish an interaction with a *User*. Interaction services are responsible for supporting data exchange and procedures invocation between the robotic system and these elements. These services can

be used, for instance, for requesting to an environment controller to open a door in a room.

**Object Manipulation:** provides algorithms to support physical interactions with objects inside the environment. Object manipulation services provide control algorithms that coordinate actuator devices, such as arms and grippers.

**Support:** provides general purpose functionalities that support the development of robotic systems. These functionalities generally involve data filtering, data fusion, math calculations, point cloud processing, and segmentation of images from cameras. As support services are not only from robotics domain, it is not possible to determine a finite number of subtypes of services that belong to this group.

4) **Robotic Agent Services:** This group encompasses services responsible for coordinating other services located in less abstract layers, such as Task and Device Driver. Providing a robotic agent as a service allows the robot to be remotely controlled and eases the coordination of multi-robotic systems. As presented in Figure 6, there are two types of Robotic Agent services: *Non-mobile* and *Mobile*.

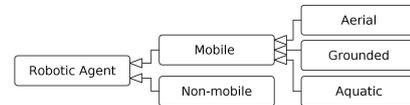


Fig. 6: Robotic Agent service group

**Non-mobile:** provides functionalities of a robot without mobility capability. An example of non-mobile service is a robotic system that controls an industrial robot designed to manipulate objects.

**Mobile:** provides functionalities related to both locomotion and object manipulation. Due to the different types of mobile robots – as well as their distinct representations of position and orientation – mobile robotic services can be divided into three categories: *Aerial*, *Grounded*, and *Aquatic*.

5) **Application Services:** Services in this group are responsible for managing robots to perform more complex activities. Application services are orchestrators that acquire knowledge through the robotic agent services, process it, and then request a set of tasks that satisfy a given activity. Figure 7 shows the three different types of Application services: *Single Robot Application*, *Multi-robot Application*, and *Swarm Application*.

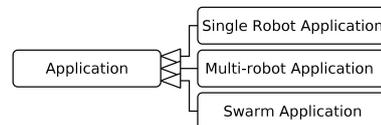


Fig. 7: Application service group

**Single Robot Application:** is used to describe, coordinate, and monitor high level robotic activities. Applications such as for robotic vacuum cleaning and intrusion detection are examples of this type of service.

**Multi-robot Application:** is used to describe, coordinate,

and monitor multiple robotic agents (i.e., multiple robots) for a given application. This type of service is also responsible to allocate tasks for robots according to their specific features and availability. Coordination of multiple robotic arms in a factory line is an example of this type of service.

**Swarm Application:** coordinates and monitors a great amount of simple robots to perform cooperative applications. In a swarm application, all robots have the same service interface and provide the same functionalities. Measurement of temperature of an environment using a robotic swarm is an example of this type of service.

### III. TAXONOMY EVALUATION

In Step 4, we evaluated our taxonomy by performing a survey with experts of the robotics domain. A group of ten experts was asked to read the taxonomy documentation and then answer questions in an on-line form. This group was formed by software architects, software engineers, software developers, and research team leaders from six different institutions from five countries, from both academy and industry. Results obtained from the survey are presented as follows and discussions are made in Section IV.

#### A. Evaluating Acceptance of the Taxonomy

In order to evaluate acceptance of our taxonomy, three aspects were evaluated during the survey: (i) RSDS, (ii) groups of services, and (iii) overall acceptance of the proposed taxonomy. For each aspect, we proposed statements that the experts had to answer using the following options: Strongly Agree (SA), Agree (Ag), Tend to Agree (TA), Neutral (Ne), Tend to Disagree (TD), Disagree (Di), and Strongly Disagree (SD). A Neutral answer means that the interviewee does not feel confident to accept or deny a given statement.

Regarding the acceptance of RSDS, three statements were made: ST1 - “The layers of RSDS are sufficient to describe the main parts and organization of a robotic system”; ST2 - “The dependencies between layers of RSDS are coherent”; ST3 - “Layers of RSDS are disjoint, i.e., there is a clear separation among layers”. Table I reports the answers for each of these statements. Notice that RSDS is considered (or tends to be considered) complete and coherent by more than 70% of the experts (i.e., more than 70% of answers about ST1 and ST 2 were SA, Ag, or TA). Besides that, 60% of the experts were in favor (strongly agreed, agreed, or tended to agree) and 30% were neutral about the statement that there is a clear separation between layers of RSDS.

TABLE I. Acceptance of the RSDS

Statement	SA	Ag	TA	Ne	TD	Di	SD
ST1	30%	40%	10%	10%	10%	0%	0%
ST2	30%	30%	10%	20%	10%	0%	0%
ST3	20%	20%	20%	30%	0%	10%	0%

To evaluate the types of services identified in the taxonomy, we made three statements for each of the five groups of services: ST1 - “The group of services is complete.”; ST2

- “The group of services is correct.”; and ST3 - “The group of services has an adequate level of abstraction.”. Table II presents the results. It is possible to note a high degree of acceptance in the groups Device Driver, Task, Robotic Agent, and Application, where an average of 87.5% of answers were positive (i.e., SA, Ag, and TA). In group Knowledge, we observed positive answers from most experts, albeit few disagreements were also observed. It is due to the fact we adopted a classification using only two different types, since a deeper classification probably would lead to an incomplete set of subcategories, i.e., it is not possible to identify all subtypes of internal and external services. Notice that a similar classification was also adopted elsewhere [2].

TABLE II. Acceptance of the groups of services

Type	ST	SA	Ag	TA	Ne	TD	Di	SD
Device Driver	ST1	30%	30%	20%	0%	20%	0%	0%
	ST2	30%	40%	10%	20%	0%	0%	0%
	ST3	30%	30%	40%	0%	0%	0%	0%
Knowledge	ST1	10%	30%	20%	20%	20%	0%	0%
	ST2	10%	30%	20%	40%	0%	0%	0%
	ST3	0%	40%	10%	20%	10%	0%	20%
Task	ST1	10%	60%	20%	0%	10%	0%	0%
	ST2	10%	50%	30%	10%	0%	0%	0%
	ST3	20%	40%	30%	0%	0%	0%	10%
Robotic Agent	ST1	40%	30%	20%	10%	0%	0%	0%
	ST2	30%	30%	20%	20%	0%	0%	0%
	ST3	30%	30%	20%	10%	0%	10%	0%
Application	ST1	20%	40%	30%	10%	0%	0%	0%
	ST2	20%	40%	20%	20%	0%	0%	0%
	ST3	30%	30%	40%	0%	0%	0%	0%

To measure the overall acceptance of the proposed taxonomy, we made four statements: ST1 - “I believe the taxonomy is clear and well-described”; ST2 - “I believe the taxonomy was defined adequately”; ST3 - “I believe the taxonomy is useful for describing services for diverse types of robotic applications”; and ST4 - “I believe the taxonomy is useful for describing services of different types of robotic systems”. Table III presents the results. It can be observed that an average of 80% have strongly agreed, agreed or tended to agree with all statements. Moreover, none of them have disagreed with the statements aforementioned. These facts indicate that the proposed taxonomy could be used as a first step to describe and classify services currently available for the development of SORS.

TABLE III. Overall acceptance of the taxonomy

Statement	SA	Ag	TA	Ne	TD	Di	SD
ST1	30%	30%	20%	20%	0%	0%	0%
ST2	40%	30%	0%	30%	0%	0%	0%
ST3	50%	10%	30%	10%	0%	0%	0%
ST4	20%	40%	20%	20%	0%	0%	0%

#### B. Assessing the Comprehension about the Taxonomy

We also assessed the understanding of the experts about our taxonomy. They were asked to classify services using the taxonomy to answer different questions (e.g., in which group a collision avoidance service should be classified). Table IV summarizes the topics that were evaluated, the number of multi-choice questions for each topic (Questions (#)), the

number of choices that has each of these questions (Choices (#)), and the average of correct answers (Avg. score).

TABLE IV. Questions and scores obtained in the assessment

Topic	Questions (#)	Choices (#)	Avg score
RSDS	8	5	78.75%
Device Driver	6	5	90.00%
Knowledge	4	2	80.25%
Task	6	4	88.33%
Application	4	2	80.00%

Notice that 78.75% of the questions made about the RSDS were correctly answered. Besides that, from 80% to 90% of the 20 questions made about the types of service in the five groups were correctly answered. These results evidence a good comprehension of our taxonomy and its use to classify services of SORS.

#### IV. A BRIEF DISCUSSION

Defining a widely accepted classification of elements of a given domain is a difficult task. Aiming at establishing a representative taxonomy of services for SORS, we considered a broad amount of information sources, including a systematic review and specialists in robotics. We also submitted the taxonomy to evaluation by the robotics community, intending to observe both its acceptance and understanding. The received feedback indicates that the taxonomy can be used for classifying services available to develop SORS. Moreover, there is a good comprehension about the taxonomy and its service groups. Minor issues were pointed out, mainly related to the Knowledge group and the completeness of the Device Driver group. Regarding the Knowledge services, we identified that sometimes there is a misunderstanding about the role played by services in this group. Knowledge services are specially dedicated to manage and share information useful for robotic systems, which sometimes is produced by services from Task group. Regarding the completeness of the Device Driver group, it is important to highlight that this taxonomy does not encompass all types of drivers for sensors and actuators available for embedded systems, but the most important and used in the development of SORS. It is also worth mentioning that this taxonomy is not final. Thus, as this research area evolves, new types of services may arise, requiring to update the taxonomy.

The proposed taxonomy moves the first step to a better communication among developers of SORS. As a consequence, its use can also contribute to improve the reusability and productivity in the SORS development, since services described using a well-defined terminology can be more easily found by other developers. Nowadays, service repositories for SORS do not support the discovery of services through categories and the identification of services that could be reused is sometimes a hard and time-consuming. Thus, based on our taxonomy, we have proposed a service management system<sup>1</sup> that supports publication and discovery

<sup>1</sup>www.labes.icmc.usp.br:8595/RegistroServicoWeb/

of services in a more productive way. In this system, services can be published and described according to one or more service types defined in the taxonomy and, then, discovered by developers using a desktop interface integrated to ROS.

#### V. CONCLUSION AND FUTURE WORK

SOA has been increasingly adopted for the development of SORS, getting considerable advantages of SOA and resulting in more flexible robotic systems. In this scenario, the main contribution of this paper was to present a taxonomy that intends to adequately classify all services of the robotics domain and improve the understanding about them. Results of our work have presented good perspective of this taxonomy to become an important element in SORS development processes. As future work, we plan to conclude the development of our service management system, which will allow services to be published, classified, and discovered based on the taxonomy. By disseminating our taxonomy we intend to contribute to the reuse of services and, as a consequence, increase the productivity of SORS development.

**Acknowledgments:** This work is supported by CNPq, Capes, and grants 2011/06022-0 and 2011/23316-8, São Paulo Research Foundation (FAPESP).

#### REFERENCES

- [1] A. Arsanjani, L.-J. Zhang, M. Ellis, A. Allam, and K. Channabasaviah. S3: A service-oriented reference architecture. *IT Professional*, 9(3):10–17, 2007.
- [2] M. Blake, S. Remy, Y. Wei, and A. Howard. Robots on the web. *IEEE Robotics Automation Magazine*, 18(2):33–43, 2011.
- [3] J. Borenstein and Y. Koren. Real-time obstacle avoidance for fast mobile robots. *IEEE Transactions on Systems, Man and Cybernetics*, 19(5):1179–1187, 1989.
- [4] S. Ducasse and D. Pollet. Software architecture reconstruction: A process-oriented taxonomy. *IEEE Transactions on Software Engineering*, 35(4):573–591, 2009.
- [5] A. Farinelli, L. Iocchi, and D. Nardi. Multirobot systems: a classification focused on coordination. *IEEE Transactions on Systems, Man, and Cybernetics*, 34(5):2015–2028, 2004.
- [6] D. Feitosa and E. Y. Nakagawa. An investigation into reference architectures for mobile robotic systems. In *ICSEA'12*, pages 465–471, Lisbon, Portugal, 2012.
- [7] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on System Science and Cybernetics Computing*, 2(4):100–107, 1968.
- [8] J. Jackson. Microsoft Robotics Studio: A technical introduction. *IEEE Robotics & Automation Magazine*, 14(4):82–87, 2007.
- [9] J. J. Leonard and H. F. Durrant-Whyte. Mobile robot localization by tracking geometric beacons. *IEEE Transactions on Robotics and Automation*, 7(3):376–382, 1991.
- [10] L. B. R. Oliveira, F. S. Osório, and E. Y. Nakagawa. An investigation into the development of service-oriented robotic systems. In *SAC'13*, pages 223–226, Coimbra, Portugal, 2013.
- [11] M. P. Papazoglou and W.-J. Heuvel. Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal*, 16(3):389–415, 2007.
- [12] S. Roullet and G. Bekey. Distributed multirobot localization. *IEEE Transactions on Robotics and Automation*, 18:781–795, 2002.
- [13] E. Ruffaldi, E. Sani, and M. Bergamasco. Visualizing perspectives and trends in robotics based on patent mining. In *ICRA'10*, pages 4340–4347, Anchorage, Alaska, 2010.
- [14] T. Straszheim, B. Gerkey, and S. Cousins. The ROS build system. *IEEE Robotics & Automation Magazine*, 18(2):18–19, 2011.
- [15] M. Tenorth, U. Klank, D. Pangercic, and M. Beetz. Web-enabled robots. *Robotics Automation Magazine, IEEE*, 18(2):58–68, 2011.

# Knowledge from Document Annotations as By-Product in Distributed Software Engineering

Anna Averbakh\*, Kai Niklas<sup>†</sup>, Kurt Schneider\*

\* Software Engineering Group, Leibniz Universität Hannover, Germany

{anna.averbakh, kurt.schneider}@inf.uni-hannover.de

<sup>†</sup> Talanx Systeme AG, Germany

kai.niklas@talanx.com

**Abstract**—Knowledge management can play a major role in the success of a distributed software engineering project promising huge increases in efficiency and effectivity. However, it often suffers from a lack of participation. Major problems are that sharing knowledge is time consuming and bears additional effort for the knowledge worker.

In the course of development projects, software engineers create, read and annotate (particularly during reviews) a lot of documents. These annotations can contain valuable knowledge which should be made persistent and sharable with project partners. To lower the sharing effort for project participants, we present a light-weight approach to collect annotations as by-product from project (-related) documents. The annotations are extracted from documents and interlinked with other experience and knowledge artifacts in a shared Wiki-based experience infrastructure for global software engineering. As immediate benefit for knowledge workers, making annotations and context searchable saves information retrieval time. In the long term, annotations combined with other experiences are engineered into reusable recommendations.

In a preliminary evaluation in the software engineering research field and industry confirmed that our annotation sharing concept was perceived as helpful and time saving.

**Index Terms**—annotation management; knowledge management, distributed software engineering

## I. INTRODUCTION

Team members in a software development project do not have to be distributed to show the symptoms of a distributed project. Already a distance of 30 meters is enough to diminish communication and awareness like in a globally distributed project [1].

Many distributed collaborative software development projects follow structured and formal processes like Waterfall, V-Model or RUP. In such a process the team's work mostly revolves around (distributed) documents. Project participants often make annotations in these documents for different purposes, e.g. review or quick retrieval. Distributed projects have less direct communication and thus a lower awareness about project-related knowledge at a partner's location [1]. Participants from the partner company may not know that these annotations exist or in which documents to look. Document annotations are a valuable knowledge source. Making these annotations available and efficiently searchable can save money and time for the subcontractor, which would cost him, if he had to ask around or look through each document in the

shared project repository. Besides, annotated documents are often stored in the personal or project-wide workspace and do not become available for further projects.

As a motivating use case consider a situation, where a subcontractor team in a distributed collaboration has to create a design document. The subcontractor team needs to look up recommendations and examples on how the partner company creates UML diagrams. Such knowledge can be found e.g. in annotations inside reviewed documents from older projects containing UML diagrams. Beside communication and awareness problems, the perceived effort is still one of the most severe impediments to information and knowledge sharing (e.g. [2]). If the effort to share knowledge exceeds the reward or benefit, people will not do it [3].

Our *contribution* in this paper approaches these problems by presenting a light-weight concept to lower the effort and time to share project document annotations. The pieces of information and knowledge contained in annotations are made accessible and searchable for other teams, projects and collaboration partners. We present a methodology including a tool and environment, where extracted annotations can enrich the project memory and together with other experience and knowledge artifacts engineered into reusable guidelines.

In the next chapter we present the concepts of annotations as by-product and how to link them with other experiences. Then we present three preliminary evaluations and discuss them followed by a conclusion and topics for future research.

## II. RELATED WORK

In this section we present related research about annotation sharing and supporting the user by extracting knowledge as by-product.

### A. Annotation Extraction and Sharing

There are a lot of annotation systems and tools for collaborative work on web documents (e.g. [4], [5]). Alani et al. automatically extract knowledge from web documents and populates a knowledge base [6]. Bischofberger et al. use annotations as tags to connect software artifacts for cooperative tasks [7]. These annotations are rather formal. In our case, we extract *free text* annotations from distributed *offline* software engineering project(-related) documents.

Tools like Mendeley<sup>1</sup> or Adobe Reader<sup>2</sup> allow sharing annotations. However, Mendeley only allows to view them in its infrastructure and Adobe Reader only offers a non-searchable annotation list without visual context. Martine et al. gather and link discussions from e-mail attachments, shared folders and meetings to documents they relate to in a Wiki [8]. These discussions can have a similar content to annotations. However, information about a document extracted from annotations has a more tight connection to parts of the document than discussion points from e-mails or meetings. Liang et al. provide Excel and Word plug-ins that allow highlighting text passages, add meta data annotations and relate them to others [9]. Our approach is more *light-weight* and is *less intrusive* in the every-day work process. It does (almost) not require additional effort from the software engineer and does not limit in the way how to annotate.

### B. Knowledge as By-Product

Meyer et al. followed a by-product approach using (Microsoft Word) meeting protocols [10] to create added value for project participants. They automatically extracted decisions, tasks and definitions and displayed them in a corporate Wiki, thus making them searchable. The tasks were automatically uploaded into an issue tracker (Jira). Our approach also strives to use documents that are created anyway without added effort for the project members. In our case, we use annotated documents. Schneider describes a light-weight method to store software engineering decision rationale as by-product [11]. Schneider also presents a strategy how to record knowledge emerging as by-product from prototype development [12]. We apply this approach to annotations.

## III. KNOWLEDGE FROM ANNOTATIONS AS BY-PRODUCT

In this section we present the concepts of this contribution. With a use case we illustrate why annotations are a valuable knowledge source and can save time as well as money in distributed software development. We present how they can be made persistent and shared for organizational learning.

A document creation process, e.g. writing a specification, usually contains review loops. In a co-located project setting, the feedback will likely be in oral form: an architect can personally tell the analyst what to revise in his specification draft. In a distributed development setting, however, direct communication is more difficult and asynchronous [13]. Thus, feedback will likely be shared in solid form like e-mail or as annotations inside the document. Figure 1<sup>3</sup> illustrates information and knowledge flow in a part of a (simplified) distributed development process. The process starts with an analyst from project A, who has the task to write a specification. Thereby, he consults other documents like guidelines, rules and norms (1). The output of the writing process is a specification draft. This draft is then reviewed by the customer and architect. The former reads the specification and makes annotations on functional issues like missing or wrong requirements, improvement suggestions or opaque passages (3). The architect reads the draft in order to check if all requirements can be implemented and if there are mistakes in diagrams (4). To strengthen his argument, he may reference to a rule or guideline. He also annotates passages that are critical or costly and should be renegotiated with the customer. After receiving the annotated drafts from the reviewers, the analyst from project A writes a final version of the specification. There can be, of course, more than one review cycle. The tester uses the specification to create test cases. He marks passages in the specification that are hard to or cannot be tested. He also may annotate requirements with the test case number to quickly retrieve the relation (5). To other analysts from project B or project A at a partner location, the annotated documents become inaccessible after project A ends (6). Besides, permissions and documents are too unstructured to be valuable for people from project B.

Our approach is to store (potential) knowledge from annotations (Fig. 2) in a knowledge base, where they can be immediately searched and commented and provide an infrastructure to engineer them into recommendations in the long term. An analyst from a different project B (or from a sub-project

<sup>3</sup>In Figs. 1 and 2 we use the FLOW notation to illustrate information and experience flows between people and documents. For the full syntax description refer to Stapel et al. [14].

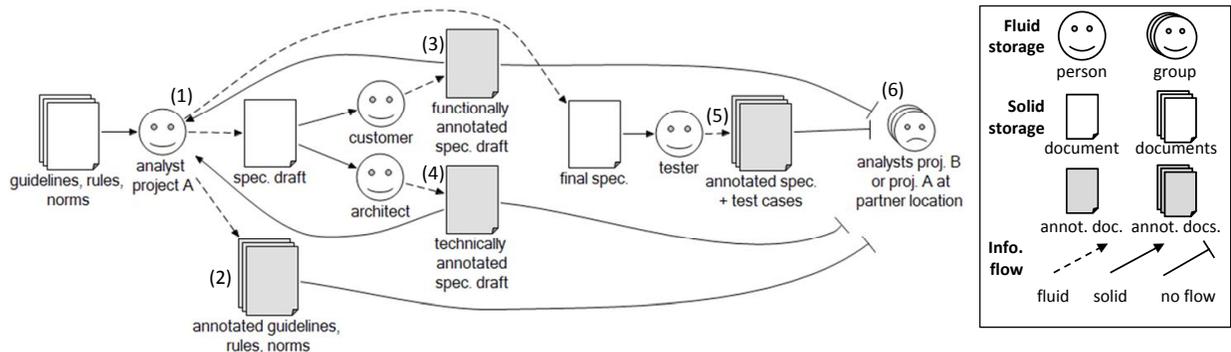


Fig. 1. Part of a (simplified) development process in a distributed setting without document annotation sharing.

of project A) can learn and benefit from the annotations. He can save time and money by reusing recommendations engineered from annotations in the previous project. He can avoid phrasing pitfalls or learn e.g. about company specific characteristics like UML diagram specifics. He can more early renegotiate and clarify untestable or very costly requirements. New annotations created during project B are a further input to the annotation database. Besides review annotations and annotated guidelines, other annotated cross-project documents like publications can also be collected. For a process manager annotations could reveal possible flaws and hidden places of unclarity in the specification template. We assume that repeated annotations denoting unclarity or problems about a similar topic or at the same place in the document could indicate that either the template is not sufficiently descriptive or the employees lack expertise and need training.

Considering the participation and motivation problems in knowledge management initiatives (e.g. [3]), we chose a lightweight approach similar to Schneider [11] to elicit the annotations. To be as little intrusive as possible for the annotation author, we extract the annotations automatically into the shared knowledge base. The author only needs one click to start the annotation extraction process. The command can be triggered from the context menu of the document.

#### IV. INTERLINKING AND ENGINEERING ANNOTATIONS

To create more benefit for the project participants, we related the extracted annotations to other knowledge and experience artifacts (s. Fig. 3). We integrated the annotations into a shared experience base for distributed software engineering (GloSE Base) [15]. This experience base is a Wiki containing experiences of different maturity. It has a rights management structure to handle sensible experiences, which is also applied to annotations. Subjective (raw) experiences are engineered into reusable recommendations. The extracted annotations in Fig. 3 can be viewed and searched in the shared knowledge base. A copy of the annotated document is attached to the page and can be downloaded. Each annotation is contextualized in the annotation context: a screenshot of

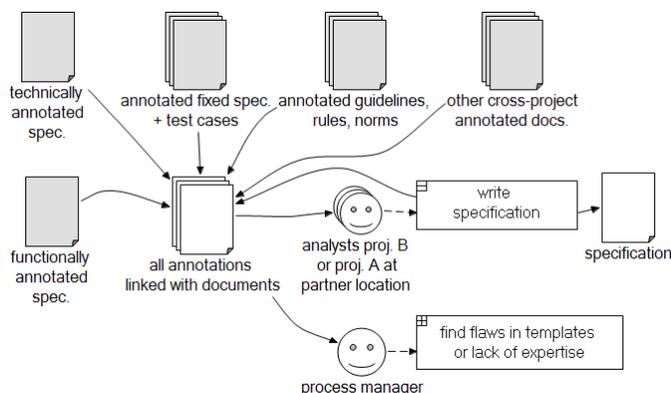


Fig. 2. Extracting and preserving project document annotations for multi-company-wide learning.

the annotation surroundings in the document. Annotations and other experiences can be engineered into a recommendation. In turn, a recommendation can link to the annotations that support it. Each annotation can be commented or linked to other documents that are related to the annotation content, e.g. a source to prove a statement. These comments and documents can also be engineered into best practices. Additionally, project participants can still access the document in its original context of the original shared folder structure. We are implementing this interlinking concept with a huge publication knowledge base structured according to SPICE (ISO/IEC 15504) on global software engineering [16].

#### V. PRELIMINARY EVALUATION

We evaluated the usefulness of the annotation extraction and sharing concepts in the software engineering domain. We identified the following research questions:

- RQ1: Is persisting and sharing annotations useful for software engineering tasks?
- RQ2: What kind of annotations are used in the software engineering field?
- RQ3: Is our annotation extraction and interlinking approach helpful? In particular, does our approach save time reading a document to find relevant information to solve a problem?
- RQ4: Can annotations be used to improve software engineering processes? Specifically, can review annotations denoting frequent mistakes of a kind give evidence to flaws and unclarity in document templates?

##### A. Methodology

We conducted three evaluations. To answer RQ1 and RQ2, we distributed a survey to software engineers in three middle sized and big software companies. The software engineers were given questions about their annotation behavior and whether our approach could be helpful for their work.

To answer RQ3, we conducted a cross validation. Our participants were software engineering research associates. The evaluation was conducted at their workplace. We implemented an annotation extractor for PDF<sup>4</sup> documents that automatically uploads annotations into our Wiki-based experience base as described in section IV. The participants were given two publications and had to answer several questions once with and once without the help of extracted and shared annotations. The participants from the control group had the same procedure but with switched publications. The publications were chosen to be tedious to read without the possibility to quickly scan the content. They were long (23 and 31 pages), relatively unstructured with long textual blocks and few figures. Their topic was chosen to be related to publications the participants would normally read but not from their direct research field. The questions were not directly searchable in the documents.

To answer RQ4, we analyzed 32 annotated requirement specifications from several reviews in a student software

<sup>4</sup>We have observed that researchers more often read and annotate PDF than MS Office documents.

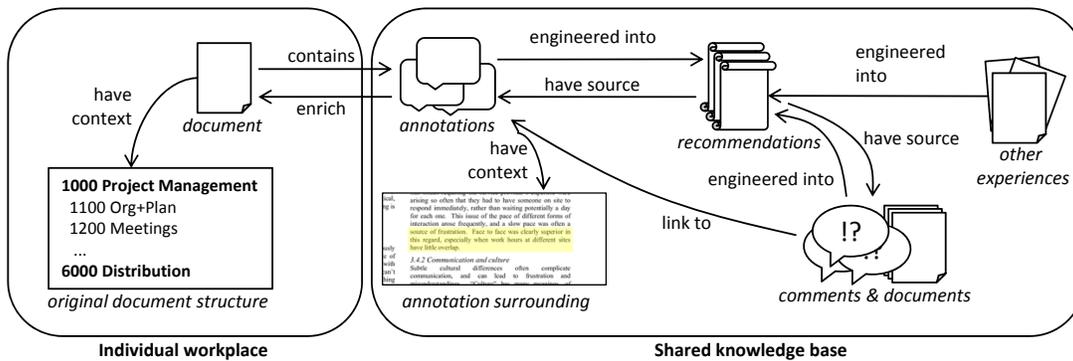


Fig. 3. Interlinking annotations, documents, comments, recommendations and the original project structure.

project at the Leibniz Universität Hannover. To create requirements specifications, students were provided templates containing a predefined structure with helpful questions and descriptions. These requirement documents were reviewed by the customer and a coach assigned to each student group. Altogether, we extracted 697 review annotations. To extract annotations denoting unclarity, we used a light-weight heuristic. We assumed that a search for annotations containing a “?” would yield questions, i.e. messages of unclarity. With the help of the text analysis framework Lucene<sup>5</sup> we identified 283 annotations containing a question. We assume that this way we can identify the most unclear chapters or places in the requirements template.

## B. Results

Altogether, we received 6 survey responses from the industry. The participating software engineers consisted of 2 developers, 1 analyst, 2 architects and 1 project manager. We can answer RQ1 positively: 4 out of 6 participants consider annotating project documents very helpful and 1 rather helpful.

To answer RQ2 we found out that software engineers mostly annotate for review purposes (with and without being involved in writing the reviewed document) and as a reminder to retrieve information quickly (5 answers). Annotating for the purpose of summarization or abbreviation of content as well as marking text to draw attention to it is less frequently used (2 answers). The most highly rated annotations to share were the reminders (3 answers). Sharing review annotations was favoured by 2 participants. All participants annotate specifications, but rarely diagrams, presentations, publications and other project-related (e.g. mercantile) documents. The most often annotated file types were Microsoft Word and Excel (3 answers), but also a lot of printed documents. PDFs, PowerPoint documents and screenshots are less often annotated.

It is not very surprising that RQ3 can be answered positively, indicating that annotations reduce the cognitive load when reading a long document. Having 9 participants, all of them needed distinctly less time to answer the questions with annotated papers than without annotations: the regular group

needed 5.7 min. without and 2.5 min. with the help of annotations as median. The control group’s median was 8.0 min. and 3.0 min. After the evaluation we received feedback that most of the participants were frustrated and even upset because they took very long to read the papers. One participant even didn’t answer the questions without the help of annotations, but wrote: “Do you really expect me to read a paper with 23 pages?” He answered the questions with the annotated paper, though. This indicates that annotations lower the barrier to read long documents if you lack the motivation. All in all, annotations and their presentation were perceived as helpful or rather helpful by 7 out of 9 academic participants. From the survey in the industry, 3 out of 6 participants stated that our annotation extraction concept could successfully support their work. Two participants did not find it useful (1 person abstained). One participant noted that searching annotations in the Wiki is helpful in a situation, where he does not remember in which document to look for.

To answer RQ4, we found out that most annotations (74) denoting unclarity were on pages where the students had to write use cases and draw use case diagrams. This finding is supported by the fact that the most frequent term in all annotations of unclarity was “use case”. This analysis indicates that these template chapters lack a good example or explanations or that the students lack expertise about how to create good use cases. After project end, the students had a post-mortem session where they could share their feedback and experiences during the project. Many groups complained about their problems with use cases and lack of a good use case example in the template. This feedback also supports the results of the annotation analysis.

## VI. DISCUSSION

Our study indicates that extracting, sharing and interlinking project (-related) document annotations is useful and helpful. Certain limitations should be considered when interpreting our results: Both the academic and industrial studies did not produce statistically significant results due to a low number of participants ( $n = 9$  and  $n = 6$ ). A statistic significance could be achieved with a larger group.

<sup>5</sup><http://lucene.apache.org/>

We limit our evaluation to “perfect” annotations in a fixed content (PDF publications). We consider an annotation “perfect” when it marks the answer in the text. We wanted to see whether such annotations can be helpful for the task of finding an answer in a research environment. Thus, our results seem unlikely to be completely representative in the software engineering field. An extensive study could give more reliable insights but our evaluation provides a tendency.

The rather low inclination (2 of 6 answers) for sharing review annotations probably reflects study participants’ opinion that these kind of annotations have a too specific context to be helpful in other projects. This may apply for direct reuse. However, our analysis results of review annotations indicates they can be helpful after engineering. Annotation analysis can be an indirect way to communicate development process flaws, which can be suitable especially in a distributed setting with impaired direct communication. Improving document templates, for example, could be helpful in the next project, as templates are usually reused. In a collaborative project, a good template is even more important if a team has to use a partner’s template [17].

As a limitation to our evaluation, we only analyzed a specific type of documents (requirement specifications) in a co-located undergraduate student software project and not in an industrial collaborative setting.

For the annotation analysis some of our assumptions, like the heuristic for finding annotations of unclarity, may be simplistic. However, it provided a problem indication that was later confirmed by students participating the software project. In addition, we plan a more sophisticated analysis to uncover more specific problems.

## VII. CONCLUSION AND OUTLOOK

In this contribution, we have presented a concept for extracting project document annotations as by-product, make them persistent and searchable as part of the organizational memory and interlinking them with other knowledge and experience artifacts in a shared experience base. Sharing document annotations can save information retrieval time. Engineering them into reusable recommendations can help to avoid mistakes that can become costly. Preliminary evaluations indicate that document annotations in the software engineering domain are a valuable knowledge asset and should be shared and engineered. We also showed, that our concept is perceived helpful by software engineers.

Our analysis of annotations has shown that they can be an indicator to flaws (in templates or in expertise) and give a notion on what can be improved in a software engineering process. This indirect way of uncovering problems can be helpful in a distributed setting when direct communication is impaired.

For further research we will add the possibility to monitor folders and automatically upload annotated documents to unburden document authors even more. We will also consider an automatic and manual support for annotation categorization and tagging. This will provide further annotation context like

the topic or problem it refers to. It will make the search for annotations easier, since a document can have annotations related to different problems and topics.

In the case of a changeable document, a synchronization and tracing of annotation context in the Wiki and the original document can be helpful. Although we made the observation that helpful annotations can lower the barrier to read academic publications, we will evaluate to which extent this insight can also be applied for the distributed software engineering domain.

Our evaluation also showed that software engineers create a lot of review annotations in transient content, but only few participants consider them worth sharing. The analysis of reviewed student software project requirements documents has shown that analyzing annotations of unclarity can give a hint about flaws in templates or team members’ expertise. We will also evaluate other ways for annotation analysis to support software engineering process improvement. In particular, we will investigate if review annotations of a specification can indicate communication flaws between customer and requirements engineer or provide input for a glossary.

## REFERENCES

- [1] J. D. Herbsleb, “Global Software Engineering : The Future of Socio-technical Coordination,” *FOSE’07*, 2007.
- [2] A. Riege, “Three-dozen knowledge-sharing barriers managers must consider,” *Journal of Knowledge Management*, 2005.
- [3] J. Grudin, “Groupware and social dynamics: eight challenges for developers,” *Communications of the ACM*, 1994.
- [4] J. J. Cadiz, A. Gupta, and J. Grudin, “Using Web annotations for asynchronous collaboration around documents,” *CSCW ’00*, 2000.
- [5] P. Nokelainen, J. Kurhila, M. Miettinen, P. Floréen, and H. Tirri, “Evaluating the Role of a Shared Document-based Annotation Tool in Learner-centered Collaborative Learning,” *ALT*, 2003.
- [6] H. Alani, D. Millard, M. Weal, W. Hall, P. Lewis, and N. Shadbolt, “Automatic ontology-based knowledge extraction from Web documents,” *IEEE Intelligent Systems*, 2003.
- [7] W. Bischofberger, T. Kofler, K.-U. Matzel, and B. Schaffer, “Computer supported cooperative software engineering with Beyond-Sniff,” *Software Engineering Environments*, 1995.
- [8] T. Martine, M. Zacklad, and A. Bénel, “Elements of methodology for designing Participative Document Spaces,” *CAIS*, 2008.
- [9] P. Liang, A. Jansen, and P. Avgeriou, “Collaborative Software Architecting Through Knowledge Sharing,” in *Collaborative Software Engineering*. Springer Berlin Heidelberg, 2010.
- [10] S. Meyer, A. Averbakh, T. Ronneberger, and K. Schneider, “Experiences from Establishing Knowledge Management in a Joint Research Project,” *Profes 2012*, 2012.
- [11] K. Schneider, “Rationale as a By-Product,” in *Rationale Management in Software Engineering*, A. H. M. Dutoit, I. Mistrík, and B. Paech, Eds. Springer, 2006.
- [12] Schneider, Kurt, “Prototypes as assets, not toys: why and how to extract knowledge from prototypes,” *SE ’96*, 1996.
- [13] B. J. Noll, S. Beecham, and I. Richardson, “Global software development and collaboration: barriers and solutions,” *ACM Inroads*, 2010.
- [14] K. Stapel and K. Schneider, “Managing Knowledge on Communication and Information Flow in Global Software Projects,” *Expert Systems - Special Issue on Knowledge Engineering in Global Software Development*, 2012.
- [15] A. Averbakh, E. Knauss, and O. Liskin, “An Experience Base with Rights Management for Global Software Engineering,” *i-KNOW ’11*, 2011.
- [16] S. Schneider, R. Torkar, and T. Gorschek, “Solutions in global software engineering: A systematic literature review,” *Int. Journal of Information Management*, 2013.
- [17] J. D. Herbsleb, D. J. Paulish, and M. Bass, “Global Software Development at Siemens: Experience from Nine Projects,” *ICSE ’05*, 2005.

# Dedicated Support for Experience Sharing in Distributed Software Projects

Anna Averbakh\*, Eric Knauss<sup>‡</sup>, Stephan Kiesling\*, Kurt Schneider\*

\* Software Engineering Group, Leibniz Universität Hannover, Germany  
{anna.averbakh, stephan.kiesling, kurt.schneider}@inf.uni-hannover.de

<sup>‡</sup> Department of Computer Science and Engineering, Chalmers | University of Gothenburg, Sweden  
eric.knauss@cse.gu.se

**Abstract**—Systematic management of experience and knowledge in distributed software development promises huge increases in effectivity and efficiency. Yet, specific problems need to be overcome: Communication between partners is difficult and awareness about the knowledge available at different locations is impaired. Even motivated developers are often reluctant to share experiences, because they do not know where and how to submit them as well as if they are allowed to share sensitive information or intellectual property. If they submit, their experiences are often presented in a way not useful for others and cannot be easily refined into best practices. In this paper we identify barriers of knowledge sharing in a literature review and discuss how dedicated tool support and automated heuristic critiques can mitigate such problems by offering the following features and qualities: guidance of project participants in creating experiences that will more likely be helpful, positive influence of their motivation to share, and easily accessible and integrated into a trustworthy experience engineering processes. Preliminary evaluation with a prototype shows that this concept can increase willingness to submit experience as well as their quality.

**Index Terms**—experience elicitation; heuristic feedback; global software engineering

## I. INTRODUCTION

Around the year 2000, many large companies recognized the importance of knowledge for software development. Several companies (e.g. Siemens [1], Ericsson [2], Daimler [3], etc.) invested heavily in knowledge management for software engineering problems. Those approaches were extended towards systematic exploitation of experiences. The intention was to avoid repeating mistakes, and rather learning from a company’s own specific experience. Distributed projects run by more than one company or business unit, however, have added new challenges to these goals [4], [5], [6]. Due to geographical, temporal and cultural differences, distributed projects suffer from an impaired communication flow and awareness [4], [7].

The success of an experience and knowledge management (EKM) initiative highly depends on the quality of the generated content. It is important that experiences are engineered into reusable guidelines and not just distributed as *raw*, subjective experiences [8], [9]. There are three major challenges for getting sufficient input into this engineering process. The most severe impediment for project participants to share information over the years has still remained the *effort to benefit* factor

[10]: If the effort to share information exceeds the reward or benefit, people will not do it [11]. However, they are more willing to share, if they believe that their contributions will be valuable to others [12]. There has been evidence that “quality of the reported experience highly depends on the *individual communication skills* of the contributor, e.g. the ability to structure the content, to formulate the experience with accuracy, and to describe it properly according to the needs of the target audience” [13]. In addition, bearers of experience often do not know *how and where to submit* their experience [14]. More research is necessary to bridge this gap.

Based on a literature study we derive barriers for submitting experience in distributed software development and propose the use of *heuristic critiques* (automated computer based feedback on experience writing) to help lowering these barriers. Such critiques automatically analyze textual input against heuristic rules and offer immediate feedback to the author. In a preliminary evaluation, we evaluated the effects of this approach and found that the heuristic critique’s immediate feedback on *how* to write experiences had a positive influence on (i) the perceived *effort to benefit ratio* and willingness to contribute experience as well as (ii) the quality of experiences.

The rest of this paper is structured as follows. In the next chapter, we present the challenges of experience management in globally distributed software engineering projects based on related work and present our research questions. Then we discuss our concept and the preliminary evaluation. This paper concludes with a summary and suggestions for future work.

## II. BACKGROUND: CHALLENGES TO EXPERIENCE SHARING IN GLOBAL SOFTWARE ENGINEERING

The first step in an experience management initiative based on an Experience Factory [8] is to collect (raw) experiences [9]. One simple way to spontaneously externalize an experience is to write an *observation sheet*, a (digital or paper) form to capture ad-hoc experiences [15]. An *experience* is a triple, consisting of 1) an observation, 2) an emotion (about the observed event) and 3) a conclusion or hypothesis [9]. Globally distributed software development adds a new dimension to experience management [6]. Systematic learning during the project becomes even more important, because different organizational and demographic cultures come together. The distributed organization of the project needs to learn how to

use these characteristics to the advantage of the project. At the same time, new challenges arise that impede successful knowledge sharing.

To uncover these challenges we conducted a literature review. The search keywords were (“knowledge OR “experience”) AND “sharing” AND (“barriers” OR “impediments” OR “issues”) AND “software engineering”. Searched databases were Emerald (55 results), IEEEExplore (0), SpringerLink (102), ScienceDirect (2), Wiley InterScience Journal Finder (2), ACM Digital Library (994), SAGE Journals (28) and Google Scholar (19700). Included in the search were journal articles, workshop and conference papers. We applied our search on the title of the publications written in English language and the keyword “software engineering” on the text. Many publications were also found through an explorative search. For the Google Scholar search, we only used the first twenty pages (200 results).

As inclusion criteria, publications were considered relevant if they report on knowledge or experience barriers. This includes general personal sharing barriers or barriers that concern groups of people (social and cultural issues). In a first review iteration and motivated by experience in a distributed collaborative project [16] it became clear that many barriers are not specific to global software development. For this reason, this review is not restricted to global settings. It poses new challenges to experience and knowledge sharing but other barriers have influence on globally distributed projects as well.

Altogether, 78 relevant publications and 63 barriers were identified<sup>1</sup>. From each publication relevant to this review, factors and barriers were collected into a table. Afterwards similar factors were consolidated and grouped according to their barrier category. From the publication causal relations between the factors were drawn.

Fig. 1 summarizes the review by displaying the higher-level barriers which are most symptomatic to global software development. The number of sub-causes (not discussed in this paper) are given in brackets. In this paper we are focusing

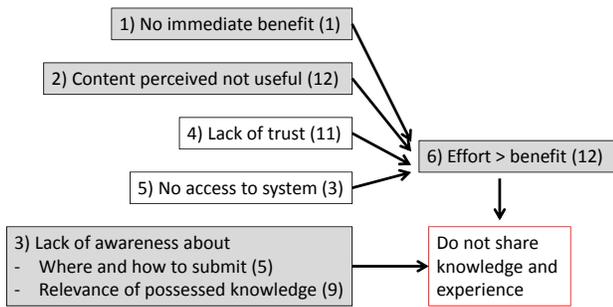


Fig. 1. EKM challenges: Barriers for sharing experience and knowledge in distributed software development.

on the elicitation of experiences and propose a tool as part of an experience environment to support experience writing

<sup>1</sup>A complete overview over the identified barriers and sources can be found at <http://www.se.uni-hannover.de/misc/aaverbakh/all-exp-barriers.pdf>

and lower the barriers from Fig. 1 in distributed software development. In the scope of this paper we will focus on the gray shaded barriers.

1) *No Immediate Benefit*: A knowledge and experience management initiative must offer the experience bearer a perceived benefit. Furthermore, the beneficiary must be the experience bearer who has contributed his experience [11]. As most people are no altruists, without personal benefit from the EKM initiative they will unlikely share again [17]. They may interpret the situation as “invest now, and someone else might harvest later” [18]. The benefit must be perceived shortly after contribution [17], [18].

To produce a quick benefit, many researchers focus on reward mechanisms to motivate knowledge bearers to share their experiences. For example, Bartol et al. propose organizational rewards [19] while Dencheva et al. discuss intrinsic reward mechanisms (levels, ranks and awards) [20].

The open question is if heuristic critiques can raise intrinsic motivation just by making the author feel guided or even proud of contributing a high(er) quality experience. There is evidence that developers are more motivated to share, if they believe that their contributions will be valuable to others [12]. To evaluate this assumption, we pose the following research question:

*RQ1: Do heuristic critiques help to write experiences with higher quality?*

2) *Content Perceived Not Useful*: An experience management initiative is not successful, if developers do not perceive its content as useful. The content may be perceived not useful, if it does not help the developer to solve his problem or specific task. Often, this is due to an inappropriate presentation or maturity level of disseminated experiences. It is important that experiences are engineered into reusable guidelines and not just distributed as *raw*, subjective experiences. Raw experiences are often too specific, unreliable, or contain confidential information. Thus, they have to be engineered to make them applicable to other projects [9].

Other research has tried to improve contributed content. Bettenburg et al. present a prototype that guides users through writing better bug reports [21]. Their study is related to our work when considering bugs as a special type of an experience. In a survey they reveal a mismatch between the information that developers need and the information that users supply. Shugerl et al. evaluate bug report quality with information retrieval techniques [22]. Support for general experience elicitation should provide the possibility to collect feedback not only to a product, but also to the development process. By this, we hope to support an active learning environment (in distributed software development) which is based on the exchange of relevant and well packaged experiences.

Similarly to the goal of heuristic critiques to improve the contributed quality, Wilson et al. present a system that aims to raise requirements quality with the help of natural language processing (NLP). It searches bad phrased requirements for improvement [23]. Bär et al. integrated NLP techniques into a Wiki to support users with the tasks of adding, organizing,

and finding content [24]. It remains to be shown that such concepts can improve the quality of experiences:

*RQ2: Do developers find experiences written with the help of critiques more helpful or richer in content?*

3) *Lack of Trust*: Lack of trust occurs when developers are not sure if sharing their experience with partner sites will be beneficial or detrimental (e.g. due to NDA policies) to them [29]. This barrier can generally be mitigated by a knowledge management infrastructure like e.g. an Experience Factory [8], which is a fundamental approach to systematically elicit, refine, package, and reuse experience. It is a basis for our concept of experience engineering. In our previous work, we presented an experience engineering process for collaborative projects that considers trust issues [30] and we will not further discuss this barrier in the scope of this paper.

4) *No Access to System*: No access to the experience base can be an issue when joint ventures contain partners that have a competitive relationship outside the collaborative project. Then, usually the strongest collaboration partner can restrict access to the stored knowledge for the competing partners. The developers of the partner organization(s) do not get enough access to system to benefit from it [16]. Similarly to *lack of trust*, our infrastructure from previous work [30] approaches this issue presenting a rights management structure and engineering process. It ensures that no sensible experiences can be viewed by competitive partners making access restrictions redundant. We will not further discuss this barrier in the scope of this paper.

5) *Lack of Awareness*: Lack of awareness is increased by physical distribution, leading to more difficult and less direct communication between project participants. Often, participants do not know where and how to submit experience [14]. In addition, participants often have difficulty to assess the relevance of their knowledge, as they are not aware about others' problems [20] — a general sharing problem.

A dedicated tool support mitigates the lack of awareness where and how to submit experiences, if designed to be unobtrusive and easy accessible in the workplace for each project participant. Schön's reflection-in-action approach [25] can raise awareness about one's knowledge through specific techniques that may tickle tacit knowledge out of the author he would not reflect on otherwise. Mitigating the second awareness problem, Fischer's domain-oriented design environments (DODEs) encourage practitioners to submit their experience providing an argument component with immediate feedback [26]. This feedback can initiate reflection.

As related research to raise awareness, Schneider [14] and Lübke et al. [27] propose to collect spontaneous feedback by providing specialized feedback facilities. Our approach is inspired by DODEs. The argument component providing immediate feedback (or critiques) is based on *heuristics* [28]. We apply his idea to help writing better experiences and to support reflection-in-action leading us to the following research question:

*RQ3: Does heuristic critique support influence the amount of experiences submitted by developers?*

6) *Effort > Benefit*: All barriers in Fig. 1 affect the effort to benefit ratio of EKM initiatives and thus ultimately their success: Not trusting other partners means additional effort to decide which information to share. Without access to knowledge, developers will not see any usefulness in the base's content and thus the whole system. Without enough perceived personal and immediate benefit, they will not make the effort to share [11].

As related approaches mitigating this issue, Schneider [31] and Meyer et al. [16] propose approaches aiming to extract experience and knowledge as by-product during usual work procedures without additional effort for the knowledge bearers.

Yet it remains an open question how heuristic critiques can motivate users to create more valuable experiences and make them feel more assured when writing experiences. At the same time, the concept should keep the experience writing effort as low as possible. It should not impose too many restrictions but, on the other hand, ensure the most important factors for a good experience. The above defined research question RQ3 can give indication if our approach mitigates this barrier.

In order to investigate our research questions, we need tool support that incorporates heuristic critiques. We created a prototype of such a tool as candidate solution based on the Heuristic Requirement Assistant (HeRA) [32]. Originally HeRA employed heuristic critiques to support writing of requirements and use cases. We reused this infrastructure and adapted HeRA's critique system and graphical interface to write experiences instead of requirements<sup>2</sup>.

### III. HEURISTIC SUPPORT FOR EXPERIENCE WRITING

In this section, we present the main concepts of this paper: characteristics of a *good* experience for an experience engineer and the heuristics to enhance experience quality.

We define *completeness*, *observability*, *readability* or *understandability*, *traceability* and *verboseness* as characteristics of a *good experience* in the software engineering domain. To avoid misconception, verboseness in this context is meant positively in the sense of being detailed. Our quality aspects are based on established quality characteristics of codified knowledge [33] (accuracy, readability or understandability, accessibility, currency, authority or credibility), requirements [23] (complete, consistent, correct, modifiable, ranked, testable, unambiguous, valid, verifiable) and bug reports [34], [22], [21] (focused, observable, readable, verbose). Requirements and bug reports are similar to an experience in the way that they describe events or make proposals, and are created with the goal to be easy to process in a software development (or maintenance) process. Besides, a bug report can be viewed as a special type of experience or feedback [27]. We did not consider some very bug report-specific artifacts like stack trace, patches, code samples, test cases, etc. [21]. Considering that a goal of our concept is to stay light-weight in terms of effort, we do not want to impose too many restrictions or

<sup>2</sup>For more details and a screenshot of our prototype, please refer to [http://www.se.uni-hannover.de/pages/en/heur\\_critiques\\_exp\\_elicit](http://www.se.uni-hannover.de/pages/en/heur_critiques_exp_elicit).

demands on the experience bearer but also want to ensure the most important aspects of a good experience. Too many demands may discourage experience bearers fearing too much effort to fulfil them. Thus, we discarded the quality aspects *focused*, *unambiguousness*, *consistency*, and *correctness*, because experiences are subjective and impressions of the same event can differ depending on the point of view. We also discarded *verifiability*, *validity*, *ranking*, and *modifiability*, as we consider ranking, rephrasing and comparing experiences with others experiences engineer’s tasks. Also *accessibility*, *currency* and *credibility* can be ensured during experience engineering and maintenance and do not need to be imposed on the experience bearer.

In Table I we list the concrete heuristic critiques for experience writing derived from or implementing the characteristics. For each critique the table displays characteristics it supports or enables as well as a criticality. *Error* is the most severe criticality, followed by a *warning*. *Hint* signifies an optional advice and info a general *information*. Feedback can either be general or specific to the experience part currently edited. All critiques, except 11, are context sensitive and are visible within a certain experience part. The last critique being a general information on how to write an experience is visible in all fields. Heuristics 1, 2, 3 and 8 are applied on observation, emotion and conclusion separately but are consolidated in this overview. We implemented the heuristic rules for German language, but they are generally applicable to other languages. To determine the threshold for Heuristic 2, we took a value slightly above the average sentence length of 7.08 words in German literary prose [35]. For the readability measure in Heuristic 3 we implemented the Flesch Reading Ease score [36]. We included this algorithm to raise the experience bearer’s awareness about writing style, even though we were expecting low performance because of the brevity of most observations and conclusions. This was successful, as we

could observe participants to actually rewrite texts into shorter and less nested sentences during evaluation. For Heuristics 2 and 3 we intentionally did not monitor the emotion field. In our experience, it is not important for an emotion to be very detailed. In Heuristic 4 we use emotion words from the WordNet Affect list<sup>3</sup>. To control for the quality of our critiques, we pose the following research question:

*RQ4: Do software engineers find our critiques helpful and justified?*

#### IV. PRELIMINARY EVALUATION

Our research goal is to lower the impact of individual communication skills, improve the effort-benefit ratio, and give experience bearers general support on how to submit their experiences. We evaluated our four research questions.

##### A. Setting

Ten graduate students with a Bachelor of Science degree in computer science and software development experience participated in our evaluation. We specifically chose participants with recent software engineering (SE) experiences, for example students, who had recently participated in SE activities like writing a specification or developing a mobile application in a team. 30% of our participants work or have worked in industrial software projects beside their study.

##### B. Methodology

We conducted a cross validation with two groups (G1 and G2) to evaluate the helpfulness and acceptance of our heuristic support. The participants in G1 were asked to write 2-3 SE-related experiences without and then with our critique system. The control group G2 had a reverse order of experience writing: first with and then without critiques. To answer RQ1

<sup>3</sup>We used the words set from <http://www.cse.unt.edu/~rada/affectivetext/>.

TABLE I  
THE HEURISTIC CRITIQUES FOR WRITING AN EXPERIENCE. THE COLUMN *Helpfulness* DISPLAYS THE PERCENTAGE OF TEST SUBJECTS THAT FOUND THE CRITIQUE VERY OR RATHER HELPFUL IN THE EVALUATION.

ID	Characteristic	Criticality	Feedback	Heuristic rule	Helpfulness
1	Complete	Error	The observation / emotion / conclusion does not contain text.	The observation / emotion / conclusion field is empty.	70%
2	Readable, Verbose	Warning	Your observation/ conclusion is very short.	Observation / conclusion text contains < 8 words.	90%
3	Readable	Warning	Your observation/ conclusion could be hard to read.	The readability index is < 30.	40%
4	Observable	Warning	Your emotion might not be clearly stated.	The text in emotion field does not contain an emotion word.	70%
5	Complete, Observable	Warning	Your conclusion might not convey a recommendation.	The text in conclusion field does not contain modal verbs.	60%
6	Observable, Traceable	Warning	Your conclusion possibly lacks actors or persons in charge.	The text in conclusion field is written in passive.	50%
7	Readable	Warning	Your observation / emotion / conclusion contains acronyms.	The text contains at least 2 successive capital letters.	30%
8	Traceable	Hint	The author is missing.	The author field is empty.	30%
9	Traceable, Verbose	Hint	Observation is...	-	60%
10	Traceable, Verbose	Hint	Emotion is...	-	60%
11	Traceable, Verbose	Info	Experience consists of an observation, emotion and conclusion.	-	70%

and assess the quality of the written experiences, we specified experience quality metrics. These metrics are very similar to the heuristics. Each experience was read by an experience engineer and evaluated if it complied with the metrics. The experience engineer however, did not know which critiques were fulfilled or not for this experience. Afterwards, in order to evaluate RQ2, RQ3 and RQ4, both groups answered a questionnaire<sup>4</sup> about the helpfulness of the critiques in general and for each critique in particular. They also had to rank four experiences according to their perceived quality. Two of the experiences were of poor and average quality according to our metrics. They lacked a conclusion, contained very specific acronyms, or were too short to convey enough information. The other two were the same ones but enhanced with our critique system. We used real experiences from a past distributed SE project.

### C. Results

The boxplot in Fig. 2 shows a visible experience quality improvement for RQ1. The median rates of the fulfilled quality metrics for experiences created without feedback (G1 and G2) are distinctly lower in comparison to those with critiques (G1c and G2c). The boxplot for G2 indicates a slight learning effect compared to G1. The median of G2 is slightly higher than of G1 and the quality improvement between G2 and G2c is smaller than between G1 and G1c.

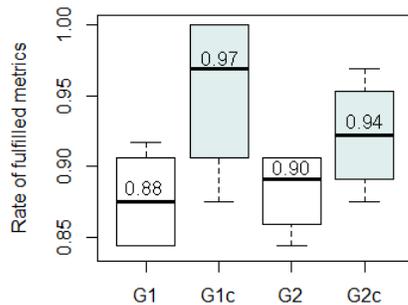


Fig. 2. Experience quality without (G1, G2) and with critiques (G1c, G2c). Note that G1 started without, while G2 started with critique support.

RQ2 can also be answered positively. In Fig. 3, 80% consider experiences enhanced with the help of critiques more helpful. Experiences enhanced with the help of critiques (E2 and E4) were ranked higher than the original ones (E1 and E3). The poor experience E1 (very short, acronyms, no conclusion) was unanimously ranked as of bad quality.

RQ3 was confirmed. 70% would write more experiences with a tool that has critiques support. For 50% critiques could lower the barrier and motivate to write more experiences. The latter is a rather inconclusive result. It indicates that users should be able to deactivate the heuristic critiques. RQ4 was also confirmed. The statistic for RQ4 shows that 100% of the students found the critiques (rather) helpful and 80% justified.

<sup>4</sup>The questionnaire (in German) can be downloaded from <http://www.se.uni-hannover.de/misc/aaverbakh/Questionnaire.pdf>

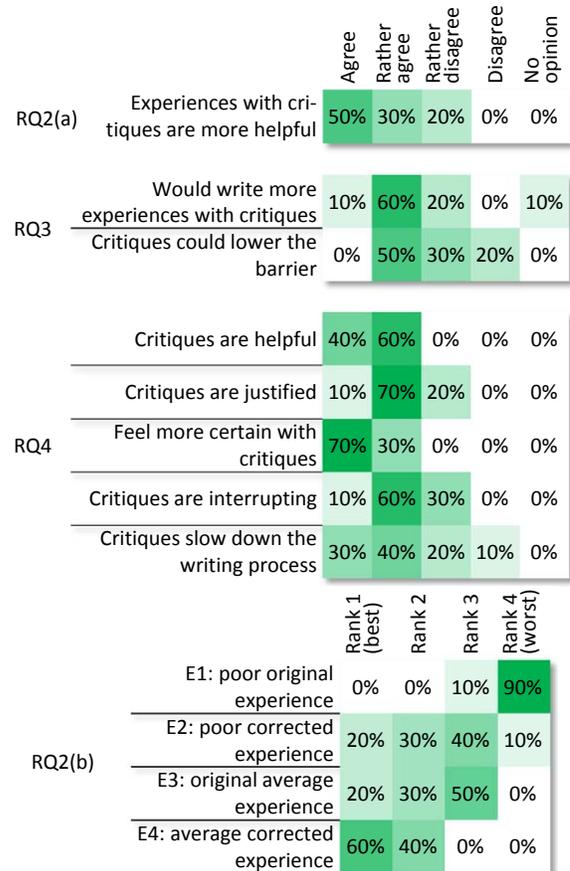


Fig. 3. Survey results.

The helpfulness of each particular heuristic is displayed in Table I. The critiques made all participants feel more certain how and what to write in an experience. On the other hand, the heuristic feedback was perceived as rather interrupting (70%). The participants often had to stop writing to read the feedback text. For 70% of the test participants, feedback slowed down their writing process. We think however, that this negative effect can vanish with repeated use and growing familiarity with the critiques.

### V. THREATS TO VALIDITY

For the preliminary evaluation of our concept we prioritized a controlled environment over transferrability of our results in industrial practice. Therefore, we employed a controlled setup with graduate students. Though our results are affirmative, they are not statistically significant ( $p = 0.19$  for G1 and  $p = 0.13$  for G2) and we had a comparably small number of participants ( $n = 10$ ). Nonetheless, our results provide a distinct indication.

Concerning the questionnaire, it is always possible that participants might misinterpret questions. We mitigated this risk by testing the questionnaire and by offering assistance during the experiment. There is a risk that students tried to confirm what they believed to be our research goals. We

mitigated this threat by formulating questions alternatively from different perspectives and by formulating our research goals in the training as neutral as possible. Rating the four experiences (fig. 3, RQ2(b)) after learning our quality criteria (expressed by the heuristics) may bias the ordering and let the participants better distinguish bad and good experiences. Future work should pose this question before the experiment.

Usability issues with HeRA might have influenced the results, e.g. some students complained that the Tab key did not work to navigate between text fields. To mitigate a negative impression, we warned the participants in advance about these issues and made clear that the editor was only a prototype.

Overly simplistic heuristic critiques may have negatively influenced the participants' opinion towards their helpfulness. For example, in case of Heur. 4, the warning often did not vanish, even if an emotion was apparent but was not on the emotion words list. Nevertheless, 70% of the participants rated this critique as (rather) helpful. It did not significantly diminish the overall perceived helpfulness of our concept (80% positive).

Based on a positive indication of this preliminary evaluation, future work should evaluate such concepts in an industrial setting in a longitudinal study as well.

For such future evaluation, we plan to use more sophisticated natural language processing techniques in our heuristics.

## VI. SUMMARY AND OUTLOOK

In this paper, we presented a concept to support writing ad-hoc experiences by providing feedback based on heuristic critiques. We defined quality aspects for a good experience (completeness, observability, readability, traceability, and verbosity) and derived 11 heuristic critiques from them. Our evaluation indicates that this approach can improve the overall experience quality and increase the willingness of experience bearers to share their experience. Both effects would be especially valuable in distributed software development. Future research needs to quantify these effects and we hope that others are able to build on our work. We are certain that practitioners would profit from integrating our concept in their experience and knowledge management initiatives and are willing to share our experience beyond the scope of this paper.

## REFERENCES

- [1] T. H. Davenport and G. J. Probst, *Knowledge Management Case Book: Siemens Best Practices*. John Wiley & Sons, Inc., 2002.
- [2] T. Dingsøy and R. Conradi, "A survey of case studies of the use of knowledge management in software engineering," *Int. Journal of SEKE*, 2002.
- [3] K. Schneider, J.-P. von Hunnius, and V. Basili, "Experience in implementing a learning software organization," *IEEE Software*, 2002.
- [4] A. Mockus and J. Herbsleb, "Challenges of global software development," in *METRICS*. IEEE, 2001.
- [5] B. J. Noll, S. Beecham, and I. Richardson, "Global software development and collaboration: barriers and solutions," *ACM Inroads*, 2010.
- [6] J. D. Herbsleb, "Global Software Engineering : The Future of Socio-technical Coordination," *FOSE*, 2007.
- [7] D. Šmite, C. Wohlin, T. Gorschek, and R. Feldt, "Empirical evidence in global software engineering: a systematic review," *Empirical Software Engineering*, 2009.
- [8] V. Basili, G. Caldiera, and H. D. Rombach, "The Experience Factory," *Proceedings of the 14th annual Software Engineering Workshop*, 1989.
- [9] K. Schneider, *Experience and Knowledge Management in Software Engineering*. Springer-Verlag, 2009.
- [10] L. Damodaran and W. Olphert, "Barriers and Facilitators to the Use of Knowledge Management Systems," *Behaviour & Information Technology*, 2000.
- [11] J. Grudin, "Groupware and social dynamics: eight challenges for developers," *Communications of the ACM*, 1994.
- [12] A. Cabrera and E. F. Cabrera, "Knowledge-Sharing Dilemmas," *Organization Studies*, 2002.
- [13] E. Ras, G. Avram, P. Waterson, and S. Weibelzahl, "Using Weblogs for Knowledge Sharing and Learning in Information Spaces," *Journal of Universal Computer Science*, 2005.
- [14] K. Schneider, S. Meyer, M. Peters, F. Schliephacke, J. Mörschbach, and L. Aguirre, "Feedback in Context: Supporting the Evolution of IT-Ecosystems," in *Product-Focused Software Process Improvement*, ser. Lecture Notes in Computer Science. Springer, 2010.
- [15] K. Schneider and T. Schwinn, "Maturing Experience Base Concepts at DaimlerChrysler," *Software Process Improvement and Practice*, 2001.
- [16] S. Meyer, A. Averbakh, T. Ronneberger, and K. Schneider, "Experiences from Establishing Knowledge Management in a Joint Research Project," *PROFES '12*, 2012.
- [17] K. Schneider, "LIDs: a light-weight approach to experience elicitation and reuse," *Product Focused Software Process Improvement*, 2000.
- [18] V. R. Basili, M. Lindvall, and F. Shull, "A Light-Weight Process for Capturing and Evolving Defect Reduction Experience," *Proceedings. Eighth IEEE International Conference on Engineering of Complex Computer Systems*, 2002.
- [19] K. M. Bartol and A. Srivastava, "Encouraging Knowledge Sharing: The Role of Organizational Reward Systems," *Journal of Leadership & Organizational Studies*, 2002.
- [20] S. Dencheva, C. R. Prause, and W. Prinz, "Dynamic Self-moderation in a Corporate Wiki to Improve Participation and Contribution Quality," *ECSCW*, 2011.
- [21] N. Bettenburg, S. Just, and C. Weiss, "What makes a good bug report?" *FSE*, 2008.
- [22] P. Schugerl, J. Rilling, and P. Charland, "Mining Bug Repositories—A Quality Assessment," *CIMCA*, 2008.
- [23] W. M. Wilson, L. Rosenberg, and L. E. Hyatt, "Automated Analysis of Requirement Specifications," *ICSE*, 1997.
- [24] D. Bär, N. Erbs, T. Zesch, and I. Gurevych, "Wikulu: An Extensible Architecture for Integrating Natural Language Processing Techniques with Wikis," *Proc. of the ACL-HLT 2011 System Demonstrations*, 2011.
- [25] D. A. Schön, *The reflective practitioner: How professionals think in action*. Basic books, 1983.
- [26] G. Fischer, "Domain-Oriented Design Environments," *ASE*, 1994.
- [27] D. Lübke and E. Knauss, "Dealing with User Requirements and Feedback in SOA Projects," *Software Engineering Methods for Service-Oriented Architecture*, 2007.
- [28] E. Knauss and K. Schneider, "Supporting Learning Organisations in Writing Better Requirements Documents Based on Heuristic Critiques," in *Proceedings of Requirements Engineering: Foundation for Software Quality (REFSQ '12)*, Essen, Germany, 2012, pp. 165–171.
- [29] A. Riege, "Three-dozen knowledge-sharing barriers managers must consider," *Journal of Knowledge Management*, 2005.
- [30] A. Averbakh, E. Knauss, and O. Liskin, "An Experience Base with Rights Management for Global Software Engineering," *i-KNOW*, 2011.
- [31] K. Schneider, "Rationale as a By-Product," in *Rationale Management in Software Engineering*. Berlin / Heidelberg: Springer, 2006.
- [32] E. Knauss, D. Lübke, and S. Meyer, "Feedback-driven requirements engineering: The Heuristic Requirements Assistant," in *ICSE*, Vancouver, Canada, 2009.
- [33] K. Dalkir, *Knowledge Management in Theory and Practice*. Routledge, 2011.
- [34] S. Tatham, "How to Report Bugs Effectively," <http://www.chiark.greenend.org.uk/~sgtatham/bugs.html>, last accessed on Feb. 11, 2014.
- [35] K.-H. Best, *Satzlängen im Deutschen: Verteilungen, Mittelwerte, Sprachwandel*. Göttinger Beiträge zur Sprachwissenschaft 7, 2002.
- [36] J. P. Kincaid, R. P. Fishburne, Jr, R. L. Rogers, and B. S. Chissom, "Derivation of New Readability Formulas (Automated Readability Index, Fog Count and Flesch Reading Ease Formula) for Navy Enlisted Personnel," *Tenn: Naval Air Station*, 1975.

# QoS-Based Web Service Composition by GA Using Consumer Decision-Making Function

Gang Wang<sup>1</sup>, Li Zhang<sup>1</sup>, Wei Jiang<sup>1</sup> and Han Zhaogang<sup>2</sup>

Department of Computer Science and Engineering, Beihang University<sup>1</sup>

Commercial Aircraft Corporation of China, Ltd.<sup>2</sup>

Beijing, P.R.China

e-mail: f\_lag@cse.buaa.edu.cn

**Abstract**—There are various approaches for QoS-based web service composition using GA (Genetic Algorithm). Most of them use the sum of the values of QoS properties times their weights as the fitness function of GA, to evaluate the QoS of service compositions. However, by using this function, the service composition calculated by GA has shortcomings. For example, response time of the produced service composition is low, good for QoS, and availability is also low, bad for QoS sometimes. It influences users' QoS experiences. The problem is the function just considers overall QoS while ignoring the equilibrium of QoS properties' contributions to overall QoS. To deal with this problem, referring to the theory of customer behavior, we use the Cobb-Douglas function indicating users' decision-making behavior as the fitness function in GA. In this way, the service composition not only has high overall QoS, but each QoS property' contribution to overall QoS is more balanced.

**Keywords**- web service composition, QoS, GA, Cobb-Douglas function

## I. INTRODUCTION

SOA (Service-Oriented Architecture) becomes a popular development pattern for information system, by coupling various web services loosely and fast to adapt their business processes to meet new requirements. Web services as a basic technology for SOA have been more and more used. With various web services provided, selection of web services for composition becomes an important research topic. With constraints of users' QoS, selection from the candidate set of web service of each subtask is constraint satisfaction problem. There are several QoS-based approaches of web service composition using GA (Genetic Algorithm) [1-6], which can solve global optimization problems rapidly. GA evaluates the QoS of service compositions by defining the fitness function. QoS requirements have two aspects: values and preferences. So the fitness function is usually defined as the sum of the normalized values of QoS properties times their weights as the fitness function of GA. We call it the product-addition function. However, by using this function, the service composition calculated by GA has shortcomings. For example, response time of the service composition produced by GA is low, is good for QoS. But availability produced is also low, bad for QoS sometimes. It means the service

composition responses very quickly, but the possibility of unavailability of the service composition is also very high. It severely influences users' QoS experiences. The problem is the function just considers overall QoS while ignoring the equilibrium of QoS properties' contributions to overall QoS. It means current researches are concerned about the maximum of the fitness function while ignoring the equilibrium of each QoS property' contribution to the fitness function, which cause GA may find the service composition of which one or partial QoS properties are very good. To deal with this problem, referring to the theory of customer behavior, we use the Cobb-Douglas function indicating users' decision-making behavior as the fitness function in GA. In this way, the service composition not only has high overall QoS, but each QoS property' contribution to overall QoS is more balanced.

The remainder of this paper is organized as follows. Section 2 details the QoS-based web service composition by GA using consumer decision-making function. Section 3 argues about the advantages of GA using the Cobb-Douglas function, in contrast to GA using the product-addition function. Finally, Section 4 concludes.

## II. SEARCH FOR OPTIMIZATION SOLUTIONS OF WEB SERVICE COMPOSITION BY GA USING COBB-DOUGLAS FUNCTION

With constraints of users' QoS and cost, selection from the candidate set of web service of each subtask is constraint satisfaction problem. GA is used to search for optimization solutions of web service composition, for GA can solve global optimization problems rapidly. We explain our approaches according to the steps of GA specialization for QoS-based web service composition.

The optimization objective is to find optimal solutions of web service composition which has the best trade-off, known as a Pareto set. It indicates best balance between high quality and low cost within the constraints users set.

The implementation of genetic algorithms specific for optimal composition solutions includes the following steps: (1) genome coding, (2) population initialization, (3) fitness evaluation, (4) genetic manipulation

### a) Genome coding.

First we need to design a genome in GA to represent the

problem.

Each gene  $g_i$  represents a subtask  $t_i$ . And the  $g_i$  takes the integral value from 1 to  $n_i$  the number of candidate web services for the subtask  $t_i$ . We label candidate web services for a subtask from 1 to  $n_i$ . If value of the gene is  $j$ ,  $0 < j \leq n_i$ , it means selection of web service  $j$  to complete the subtask  $t_i$ . Fig. 1 shows genome coding.

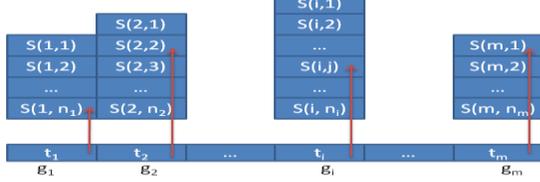


Figure 1. Genome coding

### b) Fitness evaluation.

To measure individual the degree near optimal solution, we need to define a fitness function. Referring to the theory of customer behavior, we choose the Cobb-Douglas function [7] as fitness function, which evaluates the composite solutions is the best solutions.

Formula 1 is the function prototype.

$$u(x, y) = x^a y^b, a > 0, b > 0 \quad (1)$$

Cobb-Douglas function is one of the most widely used utility function, with its special feature which reflects users' preference have the following characteristics: First, decision maker think that the more value of the property is, the better the solution is, some properties which is negative correlative with the utility of  $u$  such as cost we can use methods to make it positive correlative, secondly, for a specific value  $u$ , it is better for all properties contribution to  $u$  as equal as possible than one or several properties contribution to  $u$  far more than others. Because these features reflect characteristics of thinking when making decisions in our daily lives, this paper uses the Cobb-Douglas utility function as decision-making function.

Before using Cobb-Douglas function, data should be preprocessed. Different properties of QoS and cost of web services have different dimension. In our present system, we consider three QoS (response time, availability, reliability) and cost. To remove the effect of different dimensions, data should be normalized to make different properties into the same extent of 0 to 1.

For the positive correlative properties, this paper uses the following normalization function

$$np_{i,j} = (p_i - P_i^{\min}) / (P_i^{\max} - P_i^{\min}) \quad (1)$$

For the negative correlative properties, this paper uses the following normalization function

$$np_{i,j} = (P_i^{\max} - p_i) / (P_i^{\max} - P_i^{\min}) \quad (2)$$

where  $p$  represent a property of QoS and cost of web services,  $p_{i,j}$  is the value of a property of QoS of the candidate web service  $s(i, j)$  of the subtask  $t_i$ ,  $P_i^{\max}$  is the maximum of the property  $p$  in the candidate web service set of the subtask  $t_i$  and  $P_i^{\min}$  is the minimum. Then  $P_i^{\max}$  is  $p_i^{\max}$  plus several percentage points of  $(p_i^{\max} - p_i^{\min})$ ,  $P_i^{\min}$  is

$p_i^{\min}$  minus several percentage points of  $(p_i^{\max} - p_i^{\min})$  so that normalized properties are **not zero** and can be used in Cobb-Douglas function. In this paper, we take 10 percentage points. It may be adjusted according to actual situation.

Then Cobb-Douglas function used as fitness function is:

$$u = nrt^{\omega_1} na^{\omega_2} nr^{\omega_3} nc^{\omega_4}, \quad \sum_{i=1}^4 \omega_i = 1 \text{ and } 0 \leq \omega_i \leq 1 \quad (3)$$

where  $nrt$ ,  $na$ ,  $nr$  and  $nc$  represent normalized response time, normalized availability, normalized reliability and normalized cost of composite web services.  $\omega_1$ ,  $\omega_2$ ,  $\omega_3$ , and  $\omega_4$  indicates preference weights of response time, availability, reliability and cost, which are specified by the system or users.

Response time, availability, reliability and cost of composite web services in (3) are calculated based on QoS of component web services as follows:

TABLE I. QoS CALCULATION OF COMPOSITE WEB SERVICES

	sequence	parallel	switch	loop
response time	$\sum_{i=1}^m rt_i$	$\text{Max}_{i=1}^p (rt_i)$	$\sum_{i=1}^n (p_i * rt_i)$	$k * rt$
availability	$\prod_{i=1}^m a_i$	$\prod_{i=1}^p a_i$	$\sum_{i=1}^n (p_i * a_i)$	$a^k$
reliability	$\prod_{i=1}^m r_i$	$\prod_{i=1}^p r_i$	$\sum_{i=1}^n (p_i * r_i)$	$r^k$
cost	$\sum_{i=1}^m c_i$	$\sum_{i=1}^p c_i$	$\sum_{i=1}^n (p_i * c_i)$	$k * c$

In Table 1,  $m$  represents the number of tasks in a sequence construct,  $p$  represents the number of parallel tasks in a parallel construct,  $p_i$  represents the probability of the case  $i$  in a switch construct where  $\sum_{i=1}^n p_i = 1$ ,  $k$  represents the estimated number of iterations in a loop construct. The calculation method of QoS and cost of composite web services is adopted from [9, 10].

### c) Genetic manipulation

The selection function is the most used roulette selection. And the crossover function is the standard two-points crossover [11] while the mutation function randomly selects a subtask  $t_i$  (i.e., a gene in the genome) and then randomly select another web service to replace from  $t_i$ .

The approach has been used in our previous work [12, 13]. We argue about the advantages of this approach in contrast to GA using the traditional product-addition function in the following section.

## III. EXPERIMENT-VALIDITY OF GA USING COBB-DOUGLAS FUNCTION

We use the process combining four kinds of executive logic in Fig. 2 to check the effect of GA using Cobb-Douglas function to search for optimization solutions of web service composition. In Fig.2, ( $t_6, t_8$ ) and  $t_7$  are parallel tasks,  $t_3$  and  $t_4$  are switch tasks.

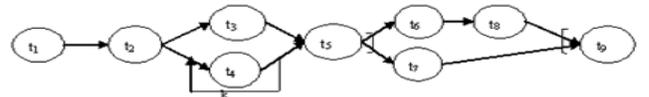


Figure 2. An example of combination to fulfill a complex task

Experimental tool is MATLAB 7.11.0 (R2010b) Genetic Algorithm Toolbox. We design 5 web services with different characteristics of QoS and cost used for candidate web services for each task. The amount of service composition of this complex task is about  $1.95 \times 10^6$ . We compare optimization solutions produced by GA using Cobb-Douglas function with that using traditional product-addition function,

to check the equilibrium of QoS properties' contributions to the overall QoS of the web service composition.

First, We compare four optimization solutions of web service composition produced by GA using  $u_1 = \prod_{i=1}^4 nq_i^{\omega_i}$  with those produced by GA using  $u_2 = \sum_{i=1}^4 nq_i \times \omega_i$  in the case whether there are preferences in QoS properties.

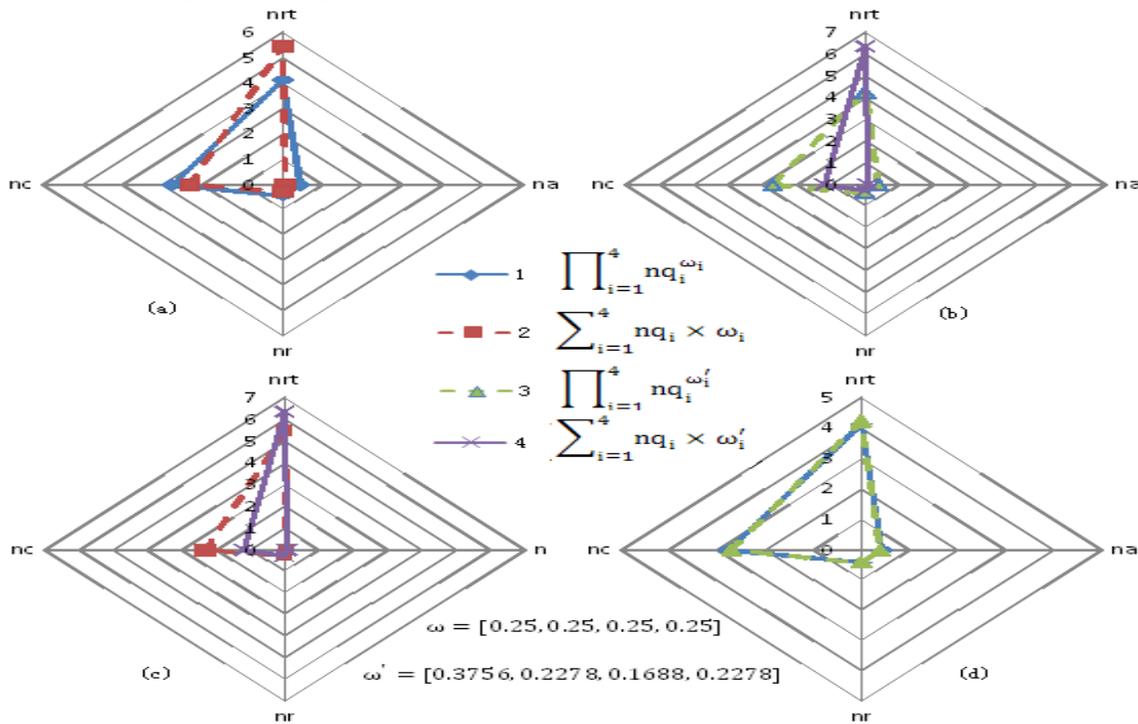


Figure 3. Comparison of optimization solutions by GA using the two fitness functions  $u_1$  and  $u_2$

Fig.3(a) shows the optimization solutions of web service composition produced by GA using the two fitness functions with no preferences for QoS properties, which means there are equal weights on QoS properties. In Fig.3(a), nrt in series 2 is higher than nrt in series 1, and the value of  $u_2$  in series 2 is higher than the value of  $u_2$  calculated with optimization values of QoS properties in series 1. But the difference between the two values in  $u_2$  is a little (0.0847, 4.2% compared to series 2). The contributions of QoS properties to the fitness function in series 1 are more balanced than those in series 2. The conclusion can be intuitively perceived by geometric features (centroid and area).

Fig.3(b) shows the optimization solutions of web service composition produced by GA using the two fitness functions with preferences for QoS properties, which means there are unequal weights on QoS properties. In Fig.3(b), nrt in series 2 is higher than nrt in series 1, and the difference is more in Fig. 3(b) than that in Fig.3(a). It is caused by more weights on nrt. The value of  $u_2$  in series 2 is still higher than the value of  $u_2$  calculated with optimization values of QoS properties in series 1, and the difference between the two

values in  $u_2$  is still a little. The contributions of QoS properties to the fitness function in series 1 are more balanced than those in series 2.

The series 2 means users get a solution of service composition which has very low response time, but may be too expensive to be acceptable to users. The series 4 means users get a solution of service composition which has low response time, but is not expensive contrasted with the series 2. The solutions between series 4 and series 2 are similar in overall QoS which are evaluated by the fitness function. So it is better for users to choose the solution of service composition in series 2, which agrees with the logic of making decisions in our daily lives, reflecting customers' behavior.

In Fig.3(c) and Fig.3(d), we compare the optimization solutions of web service composition produced by GA using the two fitness functions individually from the perspective of different weights on QoS properties.

Fig.3(d) compares the optimization solutions of web service composition produced by GA using the fitness functions  $u_1 = \prod_{i=1}^4 nq_i^{\omega_i}$  in the two weights on QoS properties. In the contrast of the results in Fig.3(c), the

contributions of QoS properties to the fitness function are still balanced. Optimization solutions in series 1 and 3 reflect the characteristics of Cobb-Douglas function aforementioned.

There is a variant of the general product-addition function  $u_3 = (na \times \omega_2 + nr \times \omega_3) / (nrt' \times \omega_1 + nc' \times \omega_4)$ . Positive QoS properties such as availability and reliability are in the numerator. The higher they are, the higher  $u_3$  is, and the more beneficial for users. On the other hand,

negative QoS properties such as response time and cost are in the denominator. The lower they are, the higher  $u_3$  is, and the more beneficial for users. All the QoS properties are normalized by (2) for this function.

In the same way, we compare four optimization solutions of web service composition produced by GA using  $u_1 = \prod_{i=1}^4 nq_i^{\omega_i}$  with those produced by GA using  $u_3$  in the case whether there are preferences in QoS properties.

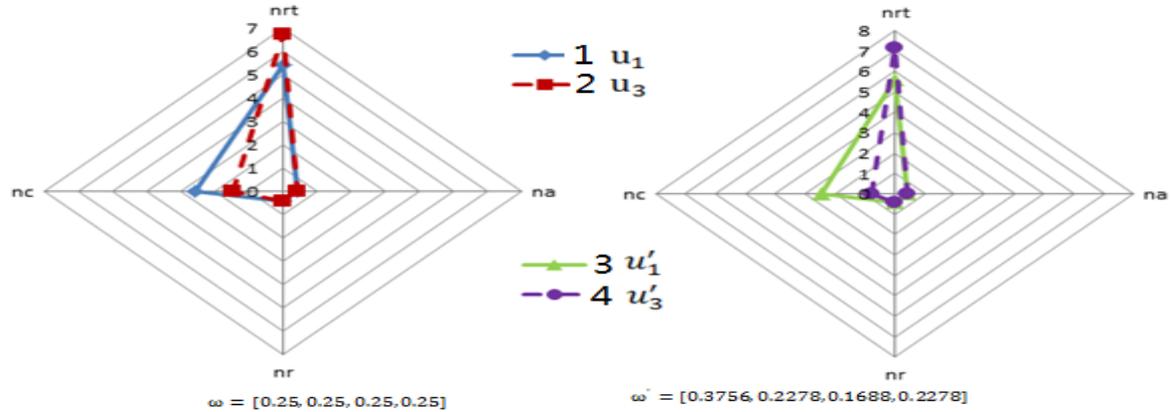


Figure 4. Comparison of optimization solutions by GA using the two fitness functions  $u_1$  and  $u_3$

The results in Fig.4 are the similar with the results in former figures. Optimization solutions in series 1 and 3 reflect the characteristics of Cobb-Douglas function aforementioned. With preferences, the imbalance in optimization QoS solutions produced by GA using  $u_3$  is more obvious.

#### IV. CONCLUSION

This paper proposes an approach of QoS-based web service composition by GA using Cobb-Douglas function. Referring to the theory of customer behavior, the produced service composition not only is high in overall QoS, but also is balanced of QoS properties' contributions to QoS, which reflects the characteristics of thinking when making decisions in our daily lives. This paper makes an improvement in the fitness function of GA. Other approaches of improved GA used in web service composition, which use the traditional product-addition function, can be replaced by this function to get a better result.

**Acknowledgments** This paper is sponsored by the National Natural Science Foundation of China under Grant No.61170087 and the Fundamental Research Funds for the Central Universities under Grant No. YWF-13-D2-XX-08.

#### REFERENCES

[1] Anja Strunk: QoS-Aware Service Composition: A Survey. In Proceeding of 8th European Conference on Web Services (ECOWS), pp. 67-74. Ayia Napa, Cyprus(2010)  
 [2] Canfora G, Penta M D, Esposito: An approach for QoS-aware service composition based on genetic algorithms. In Proceeding of the 2005

Conference on Genetic and Evolutionary Computation, pp. 1069-1075. ACM Press, New York(2005)  
 [3] Liang-Jie Zhang, Bing Li: Requirements driven dynamic services composition for Web services and grid solutions, vol. 2, pp. 121-140. Journal of Grid Computing(2004)  
 [4] Canfora G, Penta M D, Esposito: A Lightweight Approach for QoS-Aware Service composition. In: 2nd International Conference on Service Oriented Computing, pp. 36-47. IBM Technical Report, New York(2004)  
 [5] Adrian Klein, Fuyuki Ishikawa, Shinichi Honiden: Efficient Heuristic Approach with Improved Time Complexity for QoS-Aware Service Composition. International Conference on Web Services (ICWS), pp. 436-443. Washington, DC(2011)  
 [6] Zhongjun Liang, Hua Zou, Fangchun Yang, et al.: A Hybrid Approach for the Multi-constraint Web Service Selection Problem in Web Service Composition, vol. 29(13), pp. 3771-3781. Journal of Information & Computational Science(2012)  
 [7] Hal Ronald Varian: Intermediate Microeconomics: a modern approach. 7th. ShangHai: Truth&Wisdom Press(2009)  
 [8] QoS for Web Services: Requirements and Possible Approaches, <http://www.w3c.or.kr/kr-office/TR/2003/ws-qos/>  
 [9] J. Cardoso: Quality of Service and Semantic Composition of Workflows. PhD thesis, Univ. of Georgia(2002)  
 [10] Liang-zhao Zeng, Boualem Benatallah, Anne H.H. Ngu, Marlon Dumas, J. Kalaganam, and H. Chang: QoS-aware middleware for web services composition, vol. 30, pp. 311-327. IEEE Transactions on Software Engineering(2004)  
 [11] D. E. Goldberg: Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley Pub Co(1989)  
 [12] G. Wang, L. Zhang, and K. Nie: Multi-strategic Approach of Fast Composition of Web Services. In: 14th Asia-Pacific Web Conference, pp.504-512, Kunming, China (2012)  
 [13] Gang Wang, Li Zhang, Jing Jiang and Wei Jiang: QoS-Based Service Composition under Various QoS Requirements. In Proceeding of 20th IEEE Asia-Pacific Software Engineering Conference, 2013, to be published

# Using Web Mining to Support Low Cost Historical Vehicle Traffic Analytics

Charanjeet Kaur, Diwakar Krishnamurthy, Behrouz H. Far

Department of Electrical and Computer Engineering, University of Calgary, AB, Canada  
{ckaur, dkrishna, far}@ucalgary.ca

**Abstract**— Analyzing historical vehicle traffic data has many applications including urban planning and intelligent in-vehicle route prediction. A common practice to acquire this data is through roadside sensors. This approach is expensive because of infrastructure and planning costs and cannot be easily applied to new routes. In this paper, a low-cost Web mining approach is proposed to address these limitations. Our system gathers information about vehicle commute times, accidents, and weather reports from heterogeneous Web sources. Information from these sources can be combined to support road traffic analytics. We illustrate the utility of our system through a clustering analysis that investigates the traffic patterns of the busiest highway in Calgary along with factors having the most impact on commute time. The analysis shows that most of the accidents are localized around a small section of the highway near the city center and that the commute time in this segment is significantly more than that in other segments. Bad weather increases the typical evening rush hour commute time by 60% for days with moderate accidents and by a factor of 100% for days with large number of accidents. Overall, commute times can vary by a factor of 4 depending on accidents and weather.

**Keywords**—road traffic; clustering; data analysis; Web mining; traffic management

## I. INTRODUCTION

In recent years, there has been an increased interest in road traffic analysis and predictions. Traffic data can be collected in various ways. A common practice to acquire this data is through roadside sensors. This approach is expensive because of infrastructure and planning costs and cannot be easily applied to new routes.

In this paper, we propose Web mining as an alternative approach. This approach leverages the capabilities of existing Web sources and does not require any expensive infrastructure to capture current traffic data. Commonly available APIs are used to capture this already available information from the Web which can then be stored for a longer period of time. This approach is flexible and can be tailored to any route of interest and it can also incorporate new data sources and factors which affect the commute time.

Typically, the data to be collected for vehicle traffic analysis include the following:

- Number of cars, speed, flow, occupancy
- Time of the day, weekday, weekend, day of the month, season, year
- Weather, temperature, humidity

- Road type
- Events scheduled, e.g., hockey games
- Events unscheduled, e.g., fire
- Accidents, detours, lane closures
- Police archives of incidents
- Parking: location, occupancy
- Zones: schools, elderly houses, event locations, historically accident prone areas
- Glare (direction of sun)
- Tweets: incidents, locations
- Points of interest (POIs)

Many of the above information can be collected from multiple Web sources such as Google Maps [1], Twitter [2] and the weather websites. Data mining techniques can then be applied on such information to infer how the traffic pattern on a given road is related to factors such as time of the day, day of the week, accidents, and weather events. In contrast to the traditional approach of relying on specialized roadside instrumentation, this approach is more flexible in that it can be adapted with little effort and cost to analyze any road for which such Web data is available.

This paper describes our efforts to develop such a Web mining driven traffic analytics system. As a proof of concept, our system continuously collects and maintains historical commute time estimates for 18 heavy traffic roads in Calgary, Alberta, Canada. The system also extracts reports of accidents on these roads by mining Twitter posts, i.e., Tweets, related to these accidents and overlays this information with the commute time data. Finally, detailed information about weather conditions such as temperature, snowfall, and snow accumulation on the ground are mined from the Web and associated with the other pieces of data.

We illustrate the utility of our system through a study that investigates the traffic patterns of the busiest highway in Calgary along with factors having the most impact on commute time. Specifically, we use clustering [3] to discern commute time trends over a period of 153 days spanning September 2013 to February 2014. Our study shows that there was at least one accident in all of the workdays in the dataset with most accidents happening between 5 PM to 6 PM. It also shows that most of the accidents are localized around a small section of the highway near the city center and that the commute time in this segment is significantly more than the commute times in other segments. On a day when the weather is good, i.e., no snow falling or on the ground, typical evening rush hour

commute times increase by a factor of 35% for days with large number of accidents. Whereas bad weather increases the typical evening rush hour commute time by 60% for days with moderate accidents and by a factor of 100% for days with large number of accidents. Overall, commute times can vary by a factor of 4 depending on accidents and weather.

This paper makes several new contributions. Firstly, we are not aware of other studies which have exploited Web mining for traffic analysis. Our methodology can serve as an example for others attempting similar studies. Secondly, we provide new insights on commute time patterns of Calgary's busiest highway and the factors that influence them. This could for example be useful for those that manage the highway, researchers who want to validate traffic simulators for the highway, and those that are interested in predicting commute times based on historical trends. Finally, we plan to share datasets generated by our system with others for use in their studies.

The paper is organized as follows. Section 2 discusses related work. Section 3 describes our data collection, storage and clustering analysis methodologies. Results of our commute time analysis are presented in Section 4. In Section 5, limitations of this work are discussed and Section 6 concludes the paper.

## II. RELATED WORK

There has been an abundance of research work to understand traffic characteristics. Several researchers have exploited data mining techniques such as clustering and regression to characterize as well as predict traffic conditions [4][5][6][7][8][9][10]. Most of the existing studies rely on specialized roadside instrumentation such as sensors and cameras to collect traffic information. For example, Jain *et al.* exploit image processing techniques to automatically infer traffic congestion based on live camera feeds [8]. Zhang *et al.* performed visualized spatial-temporal traffic data analysis to evaluate the traffic situation in the road network level [9]. Quek *et al.* developed a fuzzy neural network model for short term traffic flow prediction [10].

Our work is different from all these studies in two important ways. First, it does not rely on video or data feeds from sensors. It is open and low-cost as data is gathered from various Web sources and there is no need for special infrastructure to be in place to collect the data. Second, these studies mainly focus on real-time traffic problems while our system focuses on historical traffic trends. Last, our technique is general and flexible. It is not limited to predefined routes and can be used on any route anywhere for which the Web data is available. Thus it complements the sensor approach by offering a solution for municipalities and businesses to conduct a fast and cost-effective analysis of the routes of their interest.

## III. METHODOLOGY

In this section we present the data collection and analysis mechanisms used in this work. The first subsection describes the methodologies used to gather heterogeneous data from multiple Web sources and the overlaying of data from one source to the others. The second subsection describes the

methodology behind the clustering analysis on the collected traffic data.

### A. Data Collection

Our data collection system design is motivated by our previously stated objective of understanding some factors that influence commute time patterns in major urban roadways. In particular, we are currently collecting data on 18 heavily congested highways and arterial roads in Calgary. Due to space constraints, we focus our analysis in Section IV on the busiest of these roads called Deerfoot Trail. Deerfoot Trail is a multi-lane highway that spans about 50 KMs within Calgary and features 21 interchanges [11]. It has 3 to 4 lanes in each direction and has a speed limit of 100 KMs/hr. The roadway is the province of Alberta's busiest highway with traffic volumes ranging between 27,000 and 158,000 vehicles per day [11]. Although we focus our analysis on one specific roadway in Calgary, our data collection and analysis methodology can be replicated in a straightforward way for other roads.

Our source of commute time information is the Google Maps website. Given a source and a destination address, Google Maps provides a basic estimate of commute time based on route distance and posted speed limits of segments constituting the route. Furthermore, the service also reports a more advanced measure called the "in current traffic" (ICT) estimate, which represents a commute time estimate that takes into account current traffic conditions on the route. This estimate is obtained by continuously tracking in real-time GPS locations of participating mobile phone users travelling on the target route [12].

A number of challenges need to be addressed to exploit the commute time data provided by Google Maps. The Google Maps API [1] is an application programming interface which supports a programmatic approach to retrieve the information displayed by the Google Maps website. However, the free version of the API does not provide the advanced ICT commute time estimate. As a result, one needs to query the Web service through a browser, save the resulting Web page, and parse the saved page to extract this measure. Furthermore, the Google Maps website does not support queries that request historical ICT estimates for a given route. Such a feature is crucial for understanding how factors such as time of the day, day of the week, and month of the year impact commute times.

We developed custom C# scripts to continuously collect route information and ICT commute time estimates for specified routes. Both scripts take as input the source and destination GPS coordinates of a target route. To allow finer grained analysis, they also accept as inputs GPS coordinates of sub-segments in the route. Finally, the second script additionally takes as input the time instants at which the ICT commute time estimates need to be collected.

The first script queries the Google Maps API and is invoked once for every route of interest and the sub-segments within those routes. The API returns compact JavaScript Object Notation (JSON) files containing information for the route and its sub-segments. Each file contains several options for traversing from the source to the destination along with the basic commute time estimates for the options. The JSON

format makes it easier to programmatically query route information during the traffic analysis phase. However, as mentioned previously, these files do not contain the ICT commute time estimates for the routes.

The second script uses the HTTP protocol to capture the ICT commute time estimates for the route and its sub-segments for various times of the day. It queries the Google Maps website at each of the time of the day instants specified as input. Each query returns a set of Web pages containing the Google Maps response for the route as well as its sub-segments. The script uses a custom parser that we developed to extract the ICT commute time estimates pertaining to the route and its sub-segments from these files. The extracted values are then stored along with other information pertaining to the route and its sub-segments contained in the previously obtained JSON files to facilitate subsequent traffic analyses.

We chose our inputs carefully to limit the number of queries issued through the Google Maps API and the number of queries submitted to the Google Maps website. The Google Maps API imposes a limit of 2,500 queries per day. Moreover, we wanted to limit the number of queries sent to the Google Maps website for obtaining the ICT commute time estimates to avoid our scripts from adversely influencing the Quality of Experience (QoE) of human users of the website. As a result, we limited our time of the day queries to 19 different time instants. Queries were issued at 30-minute intervals during rush traffic periods and 2-hour intervals during other periods. We also limited the number of sub-segments monitored in each road. For example, for Deerfoot Trail we specified 6 nearly equidistant sub-segments with each sub-segment roughly encompassing 3 consecutive interchanges. The start and end coordinates of such sub-segments were obtained manually. We defer automating this task to future work.

We augment the commute time information gathered with traffic accidents information mined from the Twitter social network [2]. After an initial analysis of all traffic related Tweets in Calgary, we selected two sources namely, Canadian Traffic Network Calgary (@CTNCalgary) and 660 News Traffic (@660NewsTraffic). Key reasons for choosing these services were the comprehensiveness of updates and the consistent formatting of the updates, which permitted easy parsing of the location of accidents.

Tweets from these sources were programmatically gathered using the Twitter Search API [2]. Twitter Search API is an application programming interface which queries the indices of recent tweets and returns the collection of most recent tweets posted by a specified user. These tweets are retrieved in a JSON file. The API restricts the number of queries per day and limits the number of Tweets returned per query to 100. Consequently, queries are issued once every hour and each of these queries retrieved the 100 most recent Tweets. Tweets from both the sources (@CTNCalgary and @660NewsTraffic) are compared to avoid duplicity as same accidents are tweeted by both. For several representative days, we manually compared the Tweet stream from the API calls to the corresponding Twitter feeds displayed on a browser and made sure that no information was lost due to the API's restrictions.

Text parsing using regular expressions is performed on the collected Tweets to look for accident reports on the target routes and to eliminate duplicate information. Furthermore, accidents are also assigned to the sub-segments of all target routes being monitored. We manually validated that the Twitter script captured all accident reported for several representative days.

Finally, we augment the commute time and accident information with historical weather data collected automatically from the climate.weather.gc.ca website [13]. Informal insights from Calgary commuters indicated that significant snowfall on a particular day and significant snow and ice on the ground due to previous precipitation activity are better predictors of traffic woes than the temperature. For example, it is not uncommon to witness smooth flow of traffic even when the temperature is -25 Celsius provided there is no precipitation and the roads are clear. As a result, we focus on these two metrics in this paper. Results presented in Section IV vindicate the choice of these metrics since they correlate well with the number of accidents and commute times.

### B. Clustering Analysis

We now describe the methodology used to analyze the behavior of Deerfoot Trail. We consider 153 days of data spanning the period September 2013 to February 2014. Since it covers both fall and winter, the data captures diverse scenarios with respect to weather conditions. As shown in Figure 2, we first classified the data into "Good weather days", i.e., days with no falling snow or snow on the ground, and "Bad weather days", i.e., days with one or both of falling snow and snow on the ground. For each of these categories, we analyze days with accidents separately from days without accidents. For days with accidents, we apply k-means clustering [3] to ascertain a small number of unique commute time patterns.

For k-means clustering, each day in our dataset was represented by a 19-element feature vector. The elements of this vector are the commute times collected by our system at various times of the day for the entire 50 KM stretch of Deerfoot Trail. We used the Weka toolset [14] with the default k-means settings. The centroid of a cluster reported by Weka is a 19-element vector whose elements represent the average commute times at the various times of the day for all days included within that cluster.

We followed the approach presented by Menasce *et al.* to determine the appropriate number of clusters for a given dataset [15]. This approach uses  $\beta_{var}$  the ratio of the variance of mean intra-cluster distances to the variance of mean inter-cluster distances to decide on the appropriate k-value, i.e., number of clusters. For good quality clustering, intra-cluster distance, i.e., the distance of feature vectors within a cluster from their centroid, must be low while inter-cluster distance, i.e., the distance of the centroid of one cluster to the centroid of another cluster, must be high. Therefore, a lower  $\beta_{var}$  value indicates better clustering results. Clustering exercises with progressively higher k-values are carried out till the  $\beta_{var}$  value shows no appreciable decrease or starts to increase. The k-value which caused the least  $\beta_{var}$  is then chosen.

As shown in Figure 1, by applying this technique we identified 3 clusters for good weather days with accidents (C2, C3, and C4) and 2 clusters for bad weather days with accidents (C5 and C6). On closer analysis, bad weather days without

cluster. The x-axis represents the 19 time instants at which the ICT commute times are collected. The y-axis starts from a non-zero origin for the sake of clarity. We show the centroid of the cluster, the day that diverges most from the centroid, i.e.,

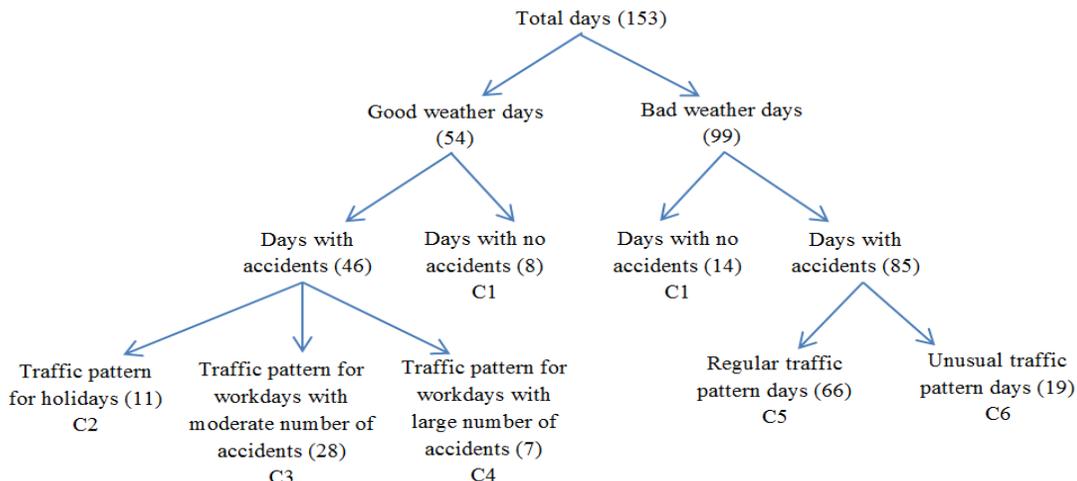


Figure 1: Distribution of days into categories based on weather and accidents for Deerfoot Trail (north to south)

accidents and good weather days without accidents contained only weekends and holidays and exhibited similar commute time patterns. Therefore, they are represented by a single cluster (C1) as the traffic pattern in these kinds of days is very similar.

#### IV. RESULTS & DISCUSSION

Due to space constraints, we only discuss the north-to-south traffic on Deerfoot Trail. We first describe some general trends from the data before discussing characteristics of the individual clusters.

From Figure 1, about 65% of the days in our observation period had bad weather. This is due to the very short fall and very long winter season in Calgary. Surprisingly, the likelihood of observing at least one accident on Deerfoot Trail is about 85% for both bad weather days and good weather days. However, closer inspection revealed that the average number of accidents per day is 1.8 times higher on bad weather days than on good weather days.

Analysis of the sub-segment information provided several interesting observations on the accidents. More than 34% of the total accidents reported in the Tweets were concentrated on a 4 KM stretch covering the 16th Avenue N, Memorial Drive, and 17th Avenue S interchanges. This sub-segment recorded higher number of accidents than any other sub-segment. Based on the speed limit of the highway, it should only take about 3 minutes to travel this sub-segment. However, the ICT commute time data indicates that commute times in this sub-segment went as high as 66 minutes at 3 PM on 23rd December 2013 when it was snowing and there was 18 cm of snow on the ground.

We next focus our attention on the clusters shown in Figure 1 and consider first the cluster C1 encompassing days with no accidents. Figure 2 shows the commute time trends for this

upper, and the day that diverges least from the centroid, i.e., lower.

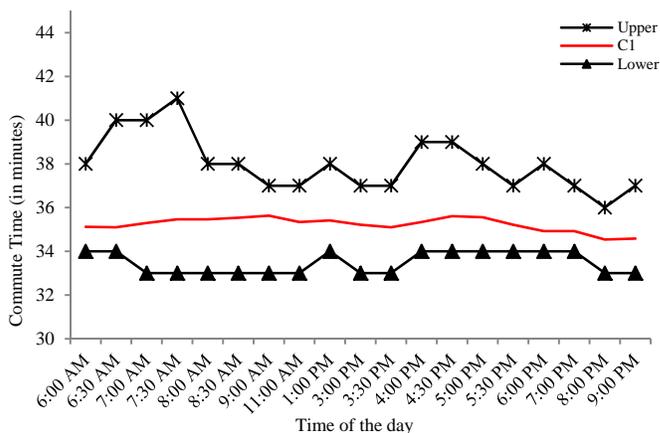


Figure 2: Days with no accidents

Figure 2 shows that weather does not seem to impact commute times significantly on weekends and holidays when there are no accidents. There is very little variability in commute times among different days as well as across different times of the day. The maximum commute times for days in this cluster varies from 42 minutes, observed on a day with bad weather, to 33 minutes, on a good weather day.

We next focus on good weather days with accidents. As shown in Figure 1, clustering identified three distinct clusters namely, weekends and holidays (C2), workdays with moderate number of accidents (C3), and workdays with large number of accidents (C4). Holidays only had an average of 1.8 accidents per day. Comparing the centroids of Figures 2 and 3, these accidents increase the commute time by up to 35% when compared to holidays without accidents. From Figure 3, one

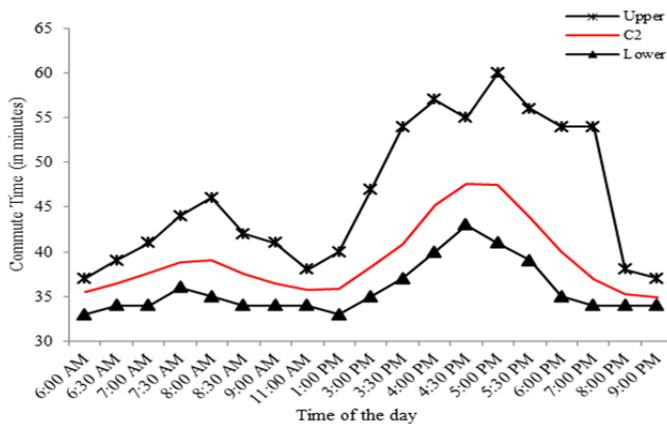


Figure 3: Good weather holidays with few accidents

can observe that commute times are longer in the evening pointing to more accidents in the evening.

Figure 4 shows traffic patterns for cluster C3, i.e., workdays with moderate number of accidents in good weather conditions. From Figure 1, this cluster represents the most likely pattern for good weather days and contains more than 50% of all good weather days. The number of accidents per day ranges from 3 to 6 with an average of 4.5. Figure 4 shows that the morning rush hours are from about 7:30 AM to 8:30 AM while the evening rush hours are from 3:30 PM to 5:30 PM. As with holidays, closer analysis showed that more accidents happen during evenings, which might explain why the evening rush hour period is longer. Figure 5 shows the traffic pattern for cluster C4, i.e., good weather days with large number of accidents. The number of accidents per day in this cluster range from 7 to 13 with an average of 8.2. Comparing the centroids of Figures 4 and 5, the magnitude of rush hour commute times and the durations of the rush hours increase significantly as the number of accidents increase. From Figure

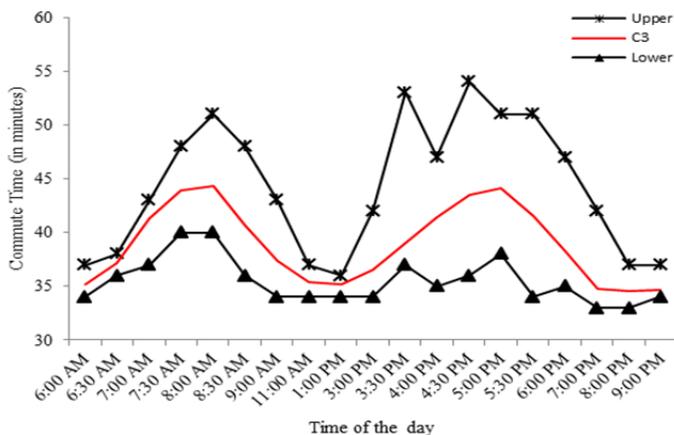


Figure 4: Good weather workdays with moderate number of accidents

5, the maximum commute time observed for this cluster was 1 hour and 46 minutes and evening rush hours are worse than morning rush hours.

Closer analysis of sub-segment commute times showed that for both C3 and C4 evening rush hour commute times in the stretch leaving the city center were much higher than the

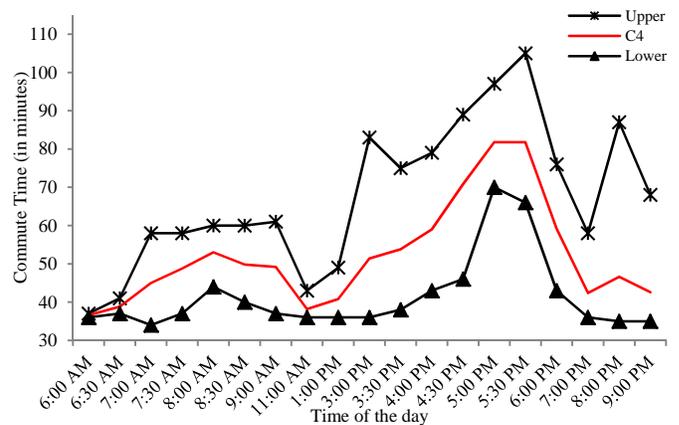


Figure 5: Good weather workdays with large number of accidents

morning rush hour commute times in the stretch leading into the city center. This suggests that the evening traffic volume out of the city center is higher than the morning traffic volume into the city center. This evening traffic represents residents living south of the city center getting back to their homes. These results confirm the longstanding intuition among Calgary residents that there is a lack of good alternatives to Deerfoot Trail for residents in the south.

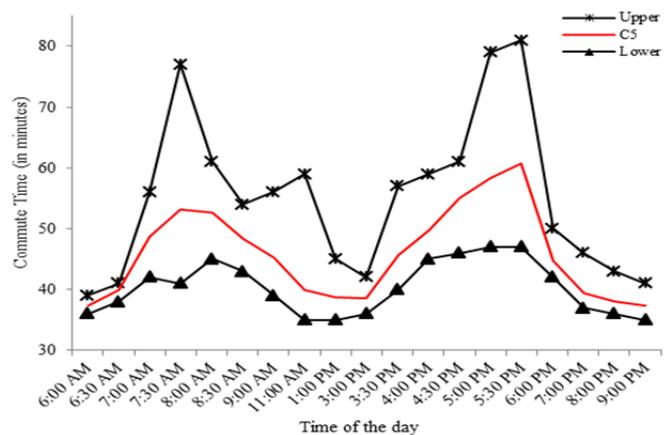


Figure 6: Bad weather days with moderate number of accidents and regular traffic patterns

We now focus our attention on bad weather days with accidents. About 66% of bad weather days were classified into a single cluster C5 as shown in Figure 1. Figure 6 shows the patterns for this cluster. The number of accidents per day range from 1 to 7 for these days with an average of 3.9. Comparing the centroid of Figure 6 with that of Figure 4 which represents the good weather day cluster with comparable average number of accidents, the maximum evening rush hour commute increases by 15 minutes due to bad weather.

Finally, Figure 7 shows the cluster C6 that contains bad weather, accident days with unusual patterns where commute time peaks happen outside of the usual peak hours. About 20% of the bad weather days fall into this cluster. The number of accidents per day varies from 8 to 16 with an average of 9.6. The upper curve of this cluster represents December 2, 2013 when Calgary was hit by an extreme snow blizzard. Commute

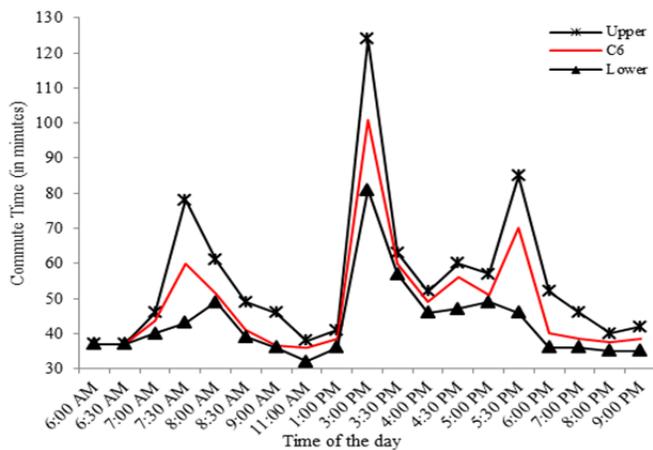


Figure 7: Bad weather days with large number of accidents and unusual traffic patterns

times were as high as 2 hours and 4 minutes with 16 accidents reported just on the north to south segment of Deerfoot Trail alone.

From Figure 1, clusters C3 and C5 contain about 62% of the days in our dataset. A noteworthy feature of these clusters is that the days in these clusters have well-defined patterns and are very close to their respective centroids. This suggests that clustering based commute time prediction models can be very effective. We intend to explore such models as future work.

## V. LIMITATIONS

Our methodology relies on the ICT commute time estimates from the Google Maps website. While our results seem to conform to real traffic trends in Calgary, a more rigorous evaluation is required to validate the accuracy of this metric. Our system also only focuses on accidents and weather as factors that influence commute time. As outlined in Section I, there are other factors such as sun position, special events, and lane closures that can impact commute times. While some of these factors, e.g., lane closures, can be identified by enhancing our Twitter scripts' text parsing capabilities, others require us to identify other Web sources, e.g., hockey game schedules from nhl.com. However, there may be factors such as traffic volumes that are currently not reported by any Web service.

As mentioned previously, the identification of sub-segments is currently done manually. Automating this would increase the flexibility of our system towards handling more routes. Our system's monitoring capabilities are also limited by query restrictions of Google Maps API and Twitter Search API.

Our analysis ignores fine-grained characterization of weather conditions based on factors such as temperature, visibility, and precipitation amounts. This limitation can be overcome easily since the climate.weather.gc.ca website [13] provides many of these metrics. We also did not focus on evaluating the effectiveness of other clustering techniques or on alternative configurations of k-means, e.g., sensitivity to various distance measures. Finally, our analysis could also benefit from a larger dataset.

## VI. CONCLUSION AND FUTURE WORK

This paper proposed a low-cost system for supporting historical analysis of vehicle commute times on roadways. The system does not require dedicated roadside sensors and associated infrastructure. Instead, it collects commute time estimates for any given route and its sub-segments from the Google Maps website over a long period of time. Furthermore, it overlays traffic accident information from the Twitter social network and weather information onto the collected commute times. We show that such a system is able to support analyses that characterize commute time patterns and their dependency on factors such as weather, accidents, and time of the day.

Future work will focus on first addressing the limitations outlined in Section V. Furthermore, the scope of the analysis could be expanded to cover multiple roads to study how traffic on a particular road affects traffic on other roads connected to it.

## ACKNOWLEDGMENT

This work has been partially supported by a grant from Natural Sciences and Engineering Research Council of Canada (NSERC).

## REFERENCES

- [1] Google Maps API, <https://developers.google.com/maps>
- [2] Twitter Search API, <https://dev.twitter.com/docs/using-search>
- [3] Tapas Kanungo, Nathan S. Netanyahu, and Angela Y. Wu, "An Efficient k-Means Clustering Algorithm: Analysis and Implementation", IEEE Transactions on Pattern Analysis and Machine Intelligence, July 2002.
- [4] Huang Yanguo and Xu Lunhui, "The Urban Road Traffic State Identification Method Based on FCM Clustering", 2011 International Conference on Transportation, Mechanical, and Electrical Engineering (TMEE), Changchun, China.
- [5] Gui-yan Jiang, Jiang-feng Wang, Xiao-dong Zhang, and Long-hui Gang, "The Study on the application of Fuzzy Clustering Analysis in the dynamic identification of road traffic state", 2003 IEEE 6th International Conference on ITS, Shanghai, China.
- [6] Maris Sekar, Mohammad Moshirpour, Julian Serfontein, and Behrouz Far, "Using Neuro-Fuzzy models to benchmark Road Safety Management Systems", 2013 IEEE International Conference on Systems, Man, and Cybernetics (SMC), pp. 4012-4017.
- [7] Christiane Stutz and Thomas A. Runkler, "Classification and Prediction of Road Traffic Using Application-Specific Fuzzy Clustering", IEEE Transactions on Fuzzy Systems, vol. 10, June 2002.
- [8] Vipin Jain, Ashlesh Sharma, and Lakshminarayanan Subramanian, "Road Traffic Congestion in the Developing World", New York University, 2012 ACM 978-1-4503-1262-2/12/03.
- [9] He-Sheng Zhang, Yi Zhang, Zhi-Heng Li, and Dong-Cheng Hu, "Spatial-Temporal Traffic Data Analysis Based on Global Data Management Using MAS", IEEE Transactions on Intelligent Transportation Systems, vol. 5, no. 4, December 2004.
- [10] Chai Quek, Michel Pasquier, and Bernard Boon Seng Lim, "POP-TRAFFIC: A Novel Fuzzy Neural Approach to Road Traffic Analysis and Prediction", IEEE Transactions on Intelligent Transportation Systems, vol. 7, no. 2, June 2006.
- [11] Deerfoot Trail, <http://www.transportation.alberta.ca/glengp.htm>
- [12] Rodric C. Fan, Xinnong Yang, and James D. Fay, "Using location data to determine traffic information", US Patent 6594576 B2, Jul 15, 2003.
- [13] Historical Climate Data, <http://climate.weather.gc.ca>
- [14] WEKA : Data Mining Tool, <http://www.cs.waikato.ac.nz/ml/weka>
- [15] D.A. Menasce, V. Almeida, R. Fonseca, and M.A. Mendes, "A Methodology for Workload Characterization of E-commerce Sites," Proc. 1999 ACM Conference on Electronic Commerce, Denver, CO, November, 1999.

# Cold-Start Web Service Recommendation Using Implicit Feedback

Gang Tian<sup>1,2</sup>, Jian Wang<sup>1\*</sup>, Keqing He<sup>1</sup>, Weidong Zhao<sup>2</sup>, Panpan Gao<sup>1</sup>

<sup>1</sup>State Key Laboratory of Software Engineering, School of Computer, Wuhan University, China

<sup>2</sup>College of Information and Science Engineering, Shandong University of Science and Technology, China  
{tiangang, \*jianwang, hekeqing}@whu.edu.cn, {zwdslj, g\_panpan}@163.com

**Abstract**—Service recommendation becomes an increasingly important issue when more and more Web services are published on the Internet. Existing Web service recommendation approaches based on collaborative filtering (CF) seldom consider recommending services based on users' ratings on services since such kind of explicit feedback is difficult to collect. In addition, the new user cold-start problem is also an important issue due to the lack of accuracy in service recommendations since the new users have not yet cast a significant numbers of votes. In this paper, a dataset consisting of much user-service interaction data is reported. The interaction data created according to users' behaviors can highly represent their preferences. Therefore, we construct pseudo ratings based on this kind of implicit feedback. We develop a novel service recommendation approach which can partially deal with cold-start problem using an online learning model. Experiments show the proposed approach can achieve satisfied results in prediction accuracy and time cost.

**Keywords**—cold-start Web service recommendation; implicit feedback; probability matrix factorization;

## I. INTRODUCTION

With the continuously increasing of Web services published on the Internet, recommending suitable Web services becomes a challenging issue [1, 2]. Existing works on recommendation systems of Web services are mainly based on collaborative filtering (CF), which computes the similarity of users or services to predict missing ratings based on the interdependencies among users and items.

Many CF-based Web service recommendation systems have been proposed in recent years [3, 4], most of which mainly focus on QoS prediction, while seldom consider recommending services based on users' explicit feedback since it is very hard to collect a large scale dataset that contains explicit feedback (e.g., explicit ratings of Web services assigned by users) or implicit feedback (e.g., service invocation history of real users). Although some artificial datasets are reported, they are very hard to simulate the diversity of real-world users' preferences.

Moreover, most existing Web service recommendation approaches based on CF pay little attention to the new user cold-start problem, which means that the systems may fail to provide any recommendation for new users since no historical information of these users are provided in the systems.

In this paper, we study the above mentioned issues of Web service recommendation, which particularly consider the situation where only implicit feedback is available.

Based on the online learning pattern, we propose a novel service recommendation approach. In particular, contributions of this paper include:

- We present a Web service recommendation approach that can partially deal with cold-start recommendation based on the implicit feedback recorded by users.
- We conduct experiments on a real-world Web services dataset, which consists of about 280,000 user-Web service interactions are collected from more than 65,000 users and more than 15,000 Web services and mashups, to verify our method. Experiments show that our recommendation system can achieve good results in prediction accuracy and time cost.

The rest of this paper is organized as follows. Section II discusses related work. Section III gives an overview of the dataset and basic concepts used in Web service recommendation. Section IV introduces our online Web service recommendation approach in detail. Section V reports the experiments and gives comparisons with existing approaches. Section VI concludes the paper.

## II. RELATED WORK

CF methods, content-based methods and hybrid methods are three kinds of methods that are widely used in Web service recommendation.

### A. CF Methods

The memory-based and model-based methods are two kinds of CF techniques that are widely used in recommendation systems. Well-known memory-based methods include user-based approaches [7] and item-based approaches [8]. Memory-based CF techniques have been recently adopted to provide QoS-aware recommendations [9, 10]. Shao et al. [9] propose a typical user-based CF method to predict QoS values which supposes that similar users tend to receive similar QoS from similar services, and they use Pearson Correlation Coefficient (PCC) to compute similarity between users. Zheng et al. develop a model which enhances the user-based CF by fusing item-based CF [10].

The model-based method allows the system to make intelligent predictions for the collaborative filtering tasks based on some learned models [5, 6]. Matrix factorization (MF) is one of the representative works. In [11], MF is used to construct a global model for predicting QoS data, which can achieve higher prediction accuracy. Yu et al. [13] propose a matrix completing approach using an effective

iterative algorithm. The method takes into account both the low-rank structure and the clustered representation of QoS data.

### B. Content Based Methods

The content based methods mainly focused on providing a mechanism to formalize users' preference, resource, and the description of Web services, and recommendations are generated based on the predefined semantic models.

Zhao et al. [15] provide a way to model services and their linkages by semantic algorithm. Based on the input keywords, users can get a set of recommendations with linkages to the query. Blake and Nowlan [24] compute a recommendation score by matching strings collected from the user's operational sessions and the description of the Web services. Based on this score, they judge whether a user is interested in the service. Mehta et al. [5] add quality and usage pattern to the service description to provide more information to discover a service that meets user requirements. Maamar et al. [6] propose a model for the context of Web service interactions and highlighted the resource on which the Web service performed.

### C. Hybrid Method

Since hybrid methods which often combine CF with other techniques can provide more accurate predictions, they are widely used. Numerous hybrid models have been presented that involve other related factors to improve service recommendation quality, such as users' locations [16, 17], social network information [18] and temporal effects [19]. Chen et al. [16] propose a CF algorithm which takes into account of users' physical locations and design a region model for large-scale Web service recommendation. Tang et al. [17] demonstrate a location aware CF model by incorporating locations of both users and services. Tang et al. [18] propose a trust-aware recommendation method with social network which integrates some social relation. Amin et al. [19] denote an approach that integrates ARIMA and GARCH models to capture the QoS attributes' volatility.

All the above mentioned approaches do not take into considerations the cold-start problem in service recommendation. There are some approaches focusing on overcoming the cold start problem in recommendation systems. For instance, Yu [12] integrates MF with decision tree learning to bootstrap service recommendation. MF is used to predict missing QoS data and then decision tree is used to handle new user cold-start issue. Bobabdilla et al. [25] design new similarity metrics using optimization based on neural learning which provides greater precision to mitigate new user cold start situations. However, works [12, 25] are both focus on providing more precise classification of new users, while we mainly deal with dynamic scenario. Ling et al [14] develop an online learning framework for collaborative filtering to handle dynamic scenario. We leverage their theory of online learning to handle cold-start problem in service recommendation.

## III. BACKGROUND

### A. Cold-start Problems in Web Service Recommendation

	Existing Users	New Users
Existing Web Services	(I)	(II)
New Web Services	(III)	(IV)

Figure 1. Partitions of Web service recommendation

Fig.1 is used to illustrate the cold-start problem in the service-oriented recommendation. Different partitions require different recommendation approaches. Available approaches for each partition are as follows:

Partition (I) (recommendation on existing Web services for existing users): this is the standard CF techniques. There is no cold-start problem in this case.

Partition (II) (recommendation on existing Web services for new users): for new users without historical ratings, the "most popular" strategy that recommends the highly-rated Web services to new users serves as a baseline. In this case, new users may rate one or more services recommended by the recommendation system. After Web services are rated by new users, the recommendation system needs to learn the new recommendation model. If the model is retrained by a batched pattern where all data is used, the time or memory cost is too high to accept. Therefore, it is reasonable that an online model training method is used to address this problem. In this paper, we mainly focus on this type of cold-start problems.

Partition (III) (recommendation on new Web services for existing users): content-based recommendation can recommend new Web services to existing users based on the users' historical ratings and features of Web services.

Partition (VI) (recommendation on new Web services for new users): in this scenario, the "random" strategy can be applied to deal with the recommendation problem.

### B. DataSet in Programmableweb

Programmableweb<sup>1</sup> (PW) is a well-known service registration center where Web APIs, member profiles and other data can be retrieved through its APIs. By using these APIs, we crawled a data set including 66 Web service categories, 17 service protocols, 7,155 mashups, 10,050 Web services, 69,384 users, and 280,611 user-service interactions in users' watch lists (by 2013-11-25). Please note that in PW, when a user is browsing services, if he is interested in a certain service, he can click on the button "track this API" and a piece of user-service interaction information will be recorded in the watch list in the form of (user, service, time). Therefore, services in users' watch lists can highly represent their preferences.

<sup>1</sup> <http://www.programmableweb.com/>

#### IV. ONLINE WEB SERVICE RECOMMENDATION

In this section, we firstly discuss how to quantify the implicit feedback of users, and then introduce our proposed approach.

##### A. Pseudo Rating From Implicit Feedback

According to our observation, each time when a service is updated, a record will be added to all the watch lists that contain this service automatically. Our statistics shows there are 133 services that are updated more than 5 times, which belong to 105 providers. Most providers have only one such updated frequently service, while the following providers are exceptions: Google has 18 frequently updated services, Yahoo has 5, Amazon has 2, and so on. Clearly a frequently updated service has a large probability of belonging to well-known companies, and thus may have a better quality.

We further investigate whether these frequently updated services are more preferred by users than these infrequently updated services. The data analysis shows that the answer is yes. For example, Google has 18 frequently updated services and 103 infrequently updated services in PW. For each frequently updated service provided by Google, it has an average user number of 72.8, that is, about 72.8 users selected this service into their watch lists, while the number is 22.75 for each infrequently updated service. The data collected from other companies also shows similar results.

Therefore, we can make an assumption that if a service is updated more frequently, it is maintained more frequently, and thus it will be more attractive to users. Therefore, it will have a larger probability to be rated higher by users.

The user-service interaction is represented as  $(u, s, \text{rating})$ , where  $u$  denotes the user,  $s$  denotes the service, and  $\text{rating}$  is the implicit feedback value calculated by Equation (1).

$$\text{rating}(u, s) = \frac{\text{freq}(s, \text{Watchlist}_u)}{|\text{Watchlist}_u|} \quad (1)$$

where  $\text{freq}(s, \text{Watchlist}_u)$  is the number that a given service  $s$  occurs in a user's watch list  $\text{Watchlist}_u$ .  $|\text{Watchlist}_u|$  is the number of services in user  $u$ 's watchlist.

##### B. MF and Probabilistic Matrix Factorization

Matrix Factorization is one of the most common approaches for recommender systems [21]. Suppose that there are  $m$  existing users and  $n$  services.  $M \in \mathbb{R}^{m \times n}$  denotes the user-service interaction, where  $M_{ij}$  represents the rating that user  $i$  assigns to service  $j$ . MF model finds a joint latent factor space of dimensionality  $k$ . MF computes two low-rank matrices  $U \in \mathbb{R}^{m \times k}$  and  $S \in \mathbb{R}^{n \times k}$  which can minimize the Frobenius norm  $\|M - US^T\|_F$  to approximate the original matrix  $M$ , as follows:

$$M_{u,s} \approx \hat{r}(u, s) = \sum_{k=1}^K U_{u,k} S_{k,s} \quad (2)$$

Probabilistic Matrix Factorization (PMF) adopts a probabilistic linear model with Gaussian observation noise [22]. It assumes Gaussian distribution on the residual noise of observed data and also places Gaussian priors on the

latent matrices  $U$  and  $S$ . Since PMF is the basis of our approach, we briefly introduce it first. The objective function of PMF for the frequency data is defined as follows:

$$p(R|U, S, \sigma^2) = \prod_{u=1}^m \prod_{s=1}^n [\mathcal{N}(R_{u,s}|U_u S_s^T, \sigma^2)]^{W_{u,s}} \quad (3)$$

where  $\mathcal{N}(x|\mu, \sigma^2)$  is the probability density function of the Gaussian distribution with mean  $\mu$  and variance  $\sigma^2$ ,  $R_{u,s}$  represents the rating that user  $u$  rates services  $s$ , and  $W_{u,s}$  is the indicator function that equals to 1 if user  $u$  rated service  $s$  and equals to 0 otherwise.

To calculate model parameters (e.g.,  $U_u$  and  $S_s^T$ ), optimization techniques should be used. Model parameters optimality is usually defined with a loss function  $l$  and the objective task is to minimize the sum of losses on the observed data. The loss function for PMF with quadratic regularization terms is defined as:

$$l = -\frac{1}{2} \sum_{u=1}^m \sum_{s=1}^n W_{u,s} (g(R_{u,s}) - g(U_u S_s^T))^2 + \frac{\lambda_u}{2} \|U\|_F^2 + \frac{\lambda_s}{2} \|S\|_F^2 \quad (4)$$

where  $g(x) = 1/(1 + e^{-x})$  is the logistic function used to map the value into the range of  $[0, 1]$ ,  $W_{u,s}$  is the indicator function, and  $\|\cdot\|_F^2$  denotes the Frobenius norm.

##### C. Algorithm

When we are using PMF, there is an assumption that all rating values are available before model learning, which makes it inappropriate for dynamic scenarios. However, the ratings in our system are extracted from users' watch lists. To capture a newly entered user's ratings, the model has to be retrained using all data. As is known to all, the time and memory consumption are very expensive to retrain the model. Therefore, online learning methods which incrementally modify the model according to the newly observed datum can highly deal with this situation.

In the following, we present our algorithms generalized from PMF named online web service recommendation (OWSR). As discussed above, we mainly deal with new user cold start problems. As shown in Fig.2, all existing ratings are organized into a sequential dataset  $\mathcal{Q}$  to train the model. Then the "most popular" strategy is applied to recommend the highly-rated Web services to new users. Finally we could adjust the recommendation model based on the new users' ratings using an online pattern. After the model is retrained, new recommendations can be offered to existing users.

In Fig.2, the "most popular" strategy provides the same recommendation to all new users based on the global popularity of services. Let  $C_s$  be the users who have rated service  $s$ , the global popularity of a Web service is defined as follows:

$$MP_s = \frac{\overline{\text{rating}}_s * n_s + \overline{\text{rating}} * \alpha}{n_s + \alpha} \quad (5)$$

where the average rating  $\overline{\text{rating}}_s$  is defined as  $\frac{1}{n_s} \sum_{u \in C_s} \text{rating}_{us}$ ,  $n_s$  is the number of users who have rated service  $s$ ,  $\overline{\text{rating}}$  denotes the average of all ratings

and  $\alpha$  denotes the shrinkage parameter which can regularize ill posed ratings. In this way, the highest rated services in the current time can be recommended to new users.

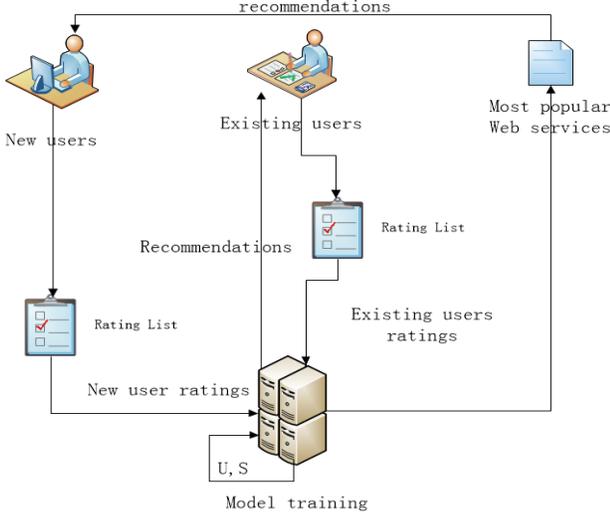


Figure 2. Cold-start Web service recommendation

The procedure of OWSR is described as follows.

**ALGORITHM1: OWSR**

**Input:** Interaction data of existing users:  $Q$

**Output:** Recommendations, Most popular Web services

1. Train the model using existing ratings.

Initialize  $U_u \leftarrow U_0$ ,  $S_s \leftarrow S_0$  randomly

$(U_u, S_s) = \text{ModelTraining}(U_u, S_s, Q)$

2. Based on  $U_u, S_s, Q$ , calculate  $MP_s$ , recommend items to new users, get interaction data of new users:  $Q_{\text{newuser}}$ , and get most popular Web service list  $MPList$ .

3. Retrain the model using new users' ratings  $Q_{\text{newuser}}$  and  $U_u, S_s$  obtained by step 1.

$(U_u, S_s) = \text{ModelTraining}(U_u, S_s, Q_{\text{newuser}})$

return  $U_u \times S_s$  and  $MPList$

Note that  $Q$  and  $Q_{\text{newuser}}$  are sequential datasets which record the user-service interactions. Each record in  $Q$  and  $Q_{\text{newuser}}$  has the form of  $(u, s, rating)$ .

Since ratings enter the system sequentially, as shown in Algorithm 1, we adjust the model using an online pattern. That is, we train the model using existing ratings first, and then adjust the model according to new users' ratings. The dual-average method for PMF which absorbs previous rating information in an approximate average gradient of the loss is one of the most widely used methods to solve optimization problems [14]. Given the  $t$ -th coming interaction data  $(u, s, rating)$ , we can update the average gradient  $\overline{Gu}_t$  and  $\overline{Gs}_t$  with respect to  $u_t$  and  $s_t$  respectively by the following rules:

$$\overline{Gu}_t = \frac{t_u - 1}{t_u} \overline{Gu}_{t-1} + \frac{1}{t_u} (g(R_{u,s}) - g(U_u S_s^T)) g'(U_u S_s^T) S_s \quad (6)$$

$$\overline{Gs}_t = \frac{t_s - 1}{t_s} \overline{Gs}_{t-1} + \frac{1}{t_s} (g(R_{u,s}) - g(U_u S_s^T)) g'(U_u S_s^T) U_u \quad (7)$$

where  $t_u$  denotes the number of services  $u$  has rated and  $t_s$  denotes the number of users who rate services  $s$ .

The model training algorithm is detailed as follows:

**ALGORITHM 2: Model Training**

**Input:**  $U, S, Q$

**Output:** new  $U, S$

1. Initialization:

$$\overline{Gs}_t \leftarrow 0, \overline{Gu}_t \leftarrow 0$$

for  $t = 1, 2, 3, \dots, |Q|$  do

2. Given the function  $l_t$ , compute the subgradient on  $U_t, S_t, \overline{Gu}_t, \overline{Gs}_t$ , For the coming instances  $(u_t, s_t, rating_t)$

$$l_t \leftarrow (g(R_{u,s}) - g(U_u S_s^T))^2$$

$$Gu_t \leftarrow (g(R_{u,s}) - g(U_u S_s^T)) g'(U_u S_s^T) S_s$$

$$Gs_t \leftarrow (g(R_{u,s}) - g(U_u S_s^T)) g'(U_u S_s^T) U_u$$

3. Update the average subgradient  $\overline{Gu}_t, \overline{Gs}_t$

$$\overline{Gu}_t = \frac{t_u - 1}{t_u} \overline{Gu}_{t-1} + \frac{1}{t_u} Gu_t$$

$$\overline{Gs}_t = \frac{t_s - 1}{t_s} \overline{Gs}_{t-1} + \frac{1}{t_s} Gs_t$$

4. Calculate the next iteration

$$U_{t+1} \leftarrow \arg \min_W \{ \overline{Gu}_t W + \lambda_U \|W\|_F^2 \}$$

$$S_{t+1} \leftarrow \arg \min_W \{ \overline{Gs}_t W + \lambda_S \|W\|_F^2 \}$$

endfor

return  $U, S$

Note that  $\lambda_U$  and  $\lambda_S$  are the regularization parameters, which are used to reduce over-fitting.

As for the time complexity, for each interaction record  $(u, s, rating) \in Q$ , only  $O(k)$  steps should be used to adjust the model, where  $k$  is the latent feature size and is much smaller than  $m$  and  $n$ . Since  $k$  is very small, it can be treated as constant time and thus the time cost scales linearly with the length of the sequential dataset  $Q$ .

V. EXPERIMENTS

A. Evaluation Metrics

The experiments are conducted on a dataset which is described in section III. The dataset contains 280,611 user-web service interactions. Since most users have very few rating information, we select 510 users who have more than 50 services in their watch list and totally 4,424 services are included. Thus we have a  $510 \times 4424$  matrix. The experiments are conducted in a PC with Core 1.8GHz processor and 4GB memory running in an operating system of Windows 8.

To measure the prediction quality of our method in comparison with other approaches, Root Mean Square Error (RMSE) metrics is embodied which is defined as:

$$RMSE = \sqrt{\frac{\sum_{u,s} (r_{u,s} - \hat{r}_{u,s})^2}{N}} \quad (8)$$

where  $r_{u,s}$  is the actual rating of Web service  $s$  assigned by user  $u$ ,  $\hat{r}_{u,s}$  is estimated value, and  $N$  indicates the number of predicted values.

B. Performance Evaluation

To illustrate the performance of our method, we compare the proposed OWSR with another five existing approaches. These approaches are illustrated as follows:

a) *UPCC*: UPCC is a typical CF algorithm using PCC as a measurement of similar users [7, 9]. We use Mahout0.8<sup>2</sup> and set the number of similar users as 50.

b) *IPCC*: IPCC is an item based CF method using PCC as a measurement of similar items [8].

c) *WSRec*: WSRec is a hybrid approach integrating both user and item based CF models [10]. During the evaluation, parameters ( $k, \lambda$ ) are set to (32, 0.2).

d) *SVT*: SVT is a singular value thresholding method that completes the matrix via minimum nuclear norm [23]. We use SVT Package<sup>3</sup>, and parameters are set as follows:  $N=510*4424$ ,  $R=20$ ,  $df=98280$ ,  $m=491400$ ,  $p=0.217796$ ,  $\tau=7510.39$ ,  $\sigma=5.50974$ ,  $iters=500$ ,  $\epsilon=0.00001$ , and  $EPS=0$ .

e) *PMF*: PMF is a matrix factorization model which scales linearly with the number of observations [22]. We use PMF Package<sup>4</sup> and parameters are set as follows:  $\lambda=\lambda_U=\lambda_V=0.01$ ,  $\eta=1000$ ,  $maxepoch=500$ , and  $num\_feat=20$ .

f) *OWSR*: parameters of our method are set as follows:  $\lambda=\lambda_U=\lambda_V=0.02$ ,  $maxepoch=500$ , and  $num\_feat=20$ .

User-service interactions are randomly divided into two categories in a certain proportion: the training set and the test set. The proportion that the training set accounts for the whole set varies from 0.1 to 0.9, which makes the sparsity of the matrix ranging from 0.9930 to 0.9992, as shown in Table I. The sparsity of the matrix is defined as  $sparsity = 1 - \frac{|M|}{|User| \times |Service|}$ , where  $| \cdot |$  is numbers of  $\cdot$ .

Table I presents the results of prediction accuracy performance of these six approaches. According to these experiments, we have the following observations.

First of all, compared with the existing four approaches (*IPCC*, *UPCC*, *WSRec*, and *SVT*), we observe that the OWSR algorithm achieves smaller RMSE values for all the cases, which shows the effectiveness of the proposed approach.

Second, the RMSE of PMF outperforms other approaches for all the cases. The average performance of PMF exceeds ours about 12%. The performance gap is probably due to the approximation when computing the gradient.

### C. OWSR (online pattern) versus PMF (batch pattern):

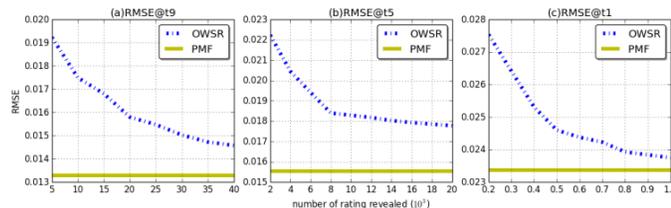


Figure 3. Comparison between OWSR and PMF

To further compare OWSR with PMF, we conduct other experiments. As shown in Fig.3, under RMSE@t9 (90% of all data are randomly chosen for training, and the remaining 10% are for evaluation), PMF exceeds our approach about 9%. Under these three settings, with the increase of the number of ratings revealed, the performance of OWSR converges to a certain value.

In Fig.3, we can see that under RMSE@t1 (10% for training and 90% for evaluation), our approach is closer to PMF compared with under RMSE@t9 and RMSE@t5. The most likely cause is that under the scenario of few training data, our model is less likely to fall into local optimum. As a whole, the performance of OWSR is approaching that of the PMF.

### D. Time Overhead

Since two algorithms (PMF and OWSR) we test are matrix factorization based, employing the same latent feature size is fair for comparison. The PMF spends more than 26 seconds in finishing 500 iterations when there are 19642 rating. OWSR spends only about 14 seconds in reaching a similar performance. As shown in Fig.4, when the number of ratings grows bigger, PMF takes much more time than OWSR. OWSR is much better than PMF to this point.

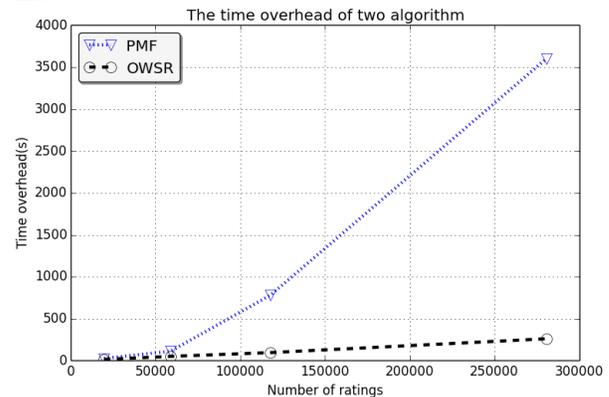


Figure 4. Time overhead of two algorithms

According to these experiments, we can conclude that the prediction accuracy of OWSR is close to PMF, while OWSR costs much shorter time.

In addition, as for the parameters used in OWSR,  $num\_feat$  is the number of latent feature indicates the degree of compression of the proposed model. We vary the latent feature number from 10 to 100 and an optimal RMSE is achieved when  $num\_feat = 20$ .

Parameter  $\lambda$  manages the trade-off between the regularization and the loss. We set  $\lambda_U=\lambda_V=\lambda$  for simplicity. An optimal result is achieved when  $\lambda = 0.02$ .

<sup>2</sup> <http://mahout.apache.org/>

<sup>3</sup> <http://statweb.stanford.edu/~candes/svt/code.html>

<sup>4</sup> <http://www.cs.toronto.edu/~rsalakhu/BPMF.html>

TABLE I. RMSE PERFORMANCE

Algorithm	Data Sparsity								
	0.9930	0.9938	0.9945	0.9953	0.9961	0.9969	0.9977	0.9984	0.9992
	Train Set Ratio								
	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1
IPCC	0.02350	0.03623	0.03749	0.04878	0.03841	0.03563	0.02257	0.03721	0.03773
UPCC	0.01929	0.03057	0.03491	0.04142	0.02230	0.03127	0.02691	0.02893	nan
WSRec	0.02956	0.02575	0.02192	0.02275	0.02160	0.02201	0.02275	0.02434	0.02652
SVT	0.02629	0.03157	0.02581	0.02639	0.03722	0.03028	0.03807	0.03130	0.03220
PMF	0.01330	0.01366	0.01520	0.01990	0.01555	0.02178	0.02172	0.02170	0.02338
OWSR	0.01458	0.01545	0.01610	0.02014	0.01797	0.02199	0.02235	0.02186	0.02375

## VI. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed an online CF algorithm based on implicit feedback data for Web service recommendation. We first introduce how to calculate ratings from the implicit feedback data. To deal with cold-start problems in Web service recommendation, we adopt the “most popular” strategy and propose a matrix factorization CF model using the online pattern. Experiments show the proposed model can achieve satisfied result in both prediction accuracy and time cost.

Our future work includes improving the performance of OWSR and exploring how to solve the cold-start problem discussed in partition III of Section III.

## ACKNOWLEDGEMENT

The work is supported by the National Basic Research Program of China under grant No. 2014CB340404, the National Science & Technology Pillar Program of China under grant No. 2012BAH07B01, the National Natural Science Foundation of China under grant No. 61202031 and 61373037.

## REFERENCES

- [1] L.-J. Zhang, J. Zhang, and H. Cai. Services computing. In Springer and Tsinghua University Press, 2007.
- [2] Q. Yu, X. Liu, A. Bouguettaya, and B. Medjahed, “Deploying and Managing Web Services: Issues, Solutions, and Directions”. The VLDB Journal, 17(3), pp. 537–572, 2008.
- [3] Y. Jiang, J. Liu, M. Tang, et al, “An effective Web service recommendation based on personalized collaborative filtering”, In ICWS, pp.213-218, 2011.
- [4] Z. Zheng, H. Ma, M.R.Lyu, et al, “QoS-aware Web service recommendation by collaborative filtering”. TSC, 4(2), pp. 140-152, 2011.
- [5] B. Mehta, C. Niederee, A. Stewart, C. Muscogiuri, and E.J.Neuhold, “An Architecture for Recommendation Based Service Mediation,” Semantics of a Networked World, vol. 3226, pp. 250-262,2004.
- [6] Z. Maamar, S.K. Mostefaoui, and Q.H. Mahmoud, “Context for Personalized Web Services,” roc. 38th Ann. Hawaii Int’l Conf. pp. 166b-166b, 2005.
- [7] J.S.Breese, D. Heckerman, C. Kadie Empirical Analysis of Predictive Algorithms for Collaborative Filtering, In UAI, pp. 43-52, 1998
- [8] M. Deshpande and G. Karypis, “Item-Based Top-N Recommendation”, ACM Trans. Information System, vol. 22, no. 1, pp. 143-177, 2004.
- [9] L. Shao, J. Zhang et al, “Personalized qos prediction for web services via collaborative filtering” , In ICWS, pp. 439-446, 2007.
- [10] Z. Zheng, H. Ma, M. R. Lyu, and I. King, “Wsrec: A collaborative filtering based web service recommender system”, In ICWS, pages 437–444, 2009.
- [11] Z. Zheng, H. Ma, M. R. Lyu, and I. King. Collaborative Web Service QoS Prediction via Neighborhood Integrated Matrix Factorization. IEEE Transactions on Service Computing (TSC), 2012.
- [12] Q. Yu, “Decision Tree Learning from Incomplete QoS to Bootstrap Service Recommendation”, In ICWS, pp. 194-201, 2012.
- [13] Q. Yu, Z. Zheng, H. Wang, “Trace norm regularized matrix factorization for service recommendation”, In ICWS, pp.34-41, 2013.
- [14] Ling G, Yang H, King I, et al. Online learning for collaborative filtering[C].Neural Networks (IJCNN), The 2012 International Joint Conference on. IEEE, 2012: 1-8.
- [15] C. Zhao, C. Ma, J. Zhang, J. Zhang, L. Yi, and X. Mao, “HyperService: Linking and Exploring Services on the Web,” Proc. Int’l Conf. Web Services, pp. 17-24, 2010.
- [16] X. Chen, X. Liu, Z. Huang, et al. “RegionKNN: a scalable hybrid collaborative filtering algorithm for personalized Web service recommendation”. In ICWS, pp. 9-16, 2010.
- [17] Tang M, Jiang Y, Liu J, et al. “Location-aware collaborative filtering for qos-based service recommendation”, In ICWS, pp. 202-209 2012.
- [18] Tang M, Xu Y, Liu J, et al. Trust-Aware Service Recommendation via Exploiting Social Networks[C] In SCC, 2013: 376-383.
- [19] Amin, Ayman, Alan Colman, and Lars Grunske. “An approach to forecasting QoS attributes of web services based on ARIMA and GARCH models”, In ICWS, pp.74-81,2012.
- [20] D.W. Oard and J. Kim, “Implicit Feedback for Recommender Systems”, Proc. 5th DELOS Workshop on Filtering and Collaborative Filtering, pp. 31–36, 1998.
- [21] Y. Koren, “Factorization Meets the Neighborhood: A Multifaceted Collaborative Filtering Model,” Proc. 14th ACM SIGKDD Int’l Conf. Knowledge Discovery and Data Mining, ACM Press, 2008, pp. 426-434
- [22] R. Salakhutdinov and A. Mnih, “Probabilistic Matrix Factorization,” Proc. Advances in Neural Information Processing Systems 20(NIPS 07), ACM Press, 2008, pp. 1257-1264.
- [23] J F. Cai, E J. Candes, and Z. Shen, “A singular value thresholding algorithm for matrix completion,” SIAM J. on Optimization, 20(4):1956–1982, 2010.
- [24] M.B.Blake and M.F. Nowlan, “A Web Service Recommender System Using Enhanced Syntactical Matching,” Proc. Int’l Conf. Web Services, pp. 575-582, 2007.
- [25] Bobadilla J, Ortega F, Hernando A, et al. A collaborative filtering approach to mitigate the new user cold start problem[J]. Knowledge-Based Systems, 2012, 26: 225-238.

# An Improved Structure-based Approach to Measure Similarity of Business Process Models

Jimin Ling, Li Zhang, Qi Feng

Software Engineering Institute, School of Computer Science and Engineering  
Beihang University (BUAA), Beijing, China  
lingjimin@cse.buaa.edu.cn, lily@buaa.edu.cn, miffy\_feng513@cse.buaa.edu.cn

**Abstract**—It is common for large enterprises or organizations to maintain repositories of process models. This paper focuses on how to measure the similarity and construct matching relations more effectively between process models. To resolve exponential time complexity to match node compositions, we proposed a graph-edit distance similarity metric based on SESE process fragments, and a greedy algorithm is employed to construct the optimal matching relations. Then a method to construct matching relations of fragments based on process structure tree is proposed. Finally, a comparative experiment based on real-world process models from BPM AI repository is conducted to evaluate the effectiveness and efficiency of our approach.

**Keywords** - Business process management; Process similarity; Process model matching; Process fragment

## I. INTRODUCTION

Various techniques have been used to manage the large collections of process models. The model search technique returns the models from the collection that are most similar to a specified input process model. The similarity measurement between two process models is the key basis of this model search technique. Besides, matching relations between the process activities need to be constructed when organizations want to resolve the differences between their own processes and industry-wide standards. In short, this paper focuses on how to measure the similarity and construct matching relations more effectively between two process models.

This topic has been studied by many researchers in recent years. They always build matching relations between nodes in process models first and then evaluate the global similarity of two models in a structural [1][2][3][4] or behavioral way [1][5][6]. The similarity evaluation is based on node matching between two models, either by unique activity name or lexical similarity between node labels. However, one business logic can be realized as different amount of activities or even different structures by various modelers. Therefore, it is not enough to only consider the node matches, matching relations between node compositions need to be taken into account when the similarity is measured [7].

It has exponential time complexity to construct appropriate matching relations between node compositions and some arbitrary compositions do not make a sense. Consider the complete and independent business logic of single-entry-single-exit (SESE) process fragment [8][9], we proposed a graph-edit distance similarity metric based on the fragments, and a greedy

algorithm is employed to compute the similarity value and construct the optimal matching relations. As a matching relation candidate set of fragments is required for similarity evaluation, we present a method to further construct matching relations of fragments by process structure tree. A comparative experiment is performed to evaluate the approach in this paper. The result shows that our approach have a positive impact on the accuracy of similarity evaluation comparing with current node matching based methods, and the efficiency of our technique is acceptable.

The rest of the paper is organized as follows. Section II introduces some basic concepts of business process model and process fragment. In section III we illustrate the key approach to measure the similarity of process models based on process fragments. Section IV presents how to construct matching relations between process fragments. Section V provides an experimental evaluation of our technique. Section VI discusses related work of our research. Section VII concludes the article and presents the future work.

## II. PRELIMINARIES

This section introduces the notion of process model graphs (PMG) and then we illustrate the key concept of process fragments which is used in this paper.

### A. Business Process Model

In this paper, we try to apply our method to various kinds of notations, so we will illustrate our method based on process model graph (PMG) rather than a specific modeling language. A PMG is simply a graph that captures nodes and edges whose properties such as names, types or resources used are treated as attributes of them. We can define the PMG and some other related concepts formally as follows.

**Definition 1 (Process Model Graph).** A process model graph (PMG) is a tuple  $(N, E, T, \Omega, \alpha)$ , in which:

- $N$  is a set of nodes;
- $E \subseteq N \times N$  is a set of directed edges;
- $T$  is a set of attribute names, e.g. TYPE, LABEL, RESOURCE, INPUT, OUTPUT, etc.;
- $\Omega$  is a set of text string values;
- $\alpha: (N \cup E) \rightarrow (T \rightarrow \Omega)$  is a function that maps nodes or edges to attributes, where an attribute is a mapping from an attribute name to a text string value.

To limit the scope, we assume that the process models are block-structured. A process model is block-structured if the sequences, branches, and loops are represented as blocks with well-defined start and end nodes. The block-structured process models are more understandable for users with less possibility of errors than the non-block-structured models. The proportion of block-structured models in a model repository is always very high and a non-block-structured model can be transformed to a block-structured one in most cases [10].

### B. Process Fragment

A block-structured process model can be decomposing hierarchically to a set of process fragments. We will introduce the single-entry-single-exit (SESE) process fragment [8][9] which is a basic concept of our approach.

**Definition 2 (SESE process fragment).** A node  $x$  is said to *dominate* node  $y$  in a directed graph if every path from start to  $y$  includes  $x$ . A node  $x$  is said to *postdominate* a node  $y$  if every path from  $y$  to end includes  $x$ . A SESE process fragment (process fragment for short) in a process graph  $G$  is an ordered edge pair  $(a, b)$  of distinct control flow edges  $a$  and  $b$  where:

- $a$  dominates  $b$ ,
- $b$  postdominates  $a$ , and
- every cycle containing  $a$  and contains  $b$  and vice versa.

We refer to  $a$  as the entry edge and  $b$  as the exit edge of the process fragment. The processes in Figure 1 gives an example of the dividing result of process fragments, and the process fragments are marked by dashed boxes. Note that the example process in Figure 1 has more fragments than those that are marked in boxes, e.g. the union of fragments I and R, denoted as  $I \cup R$ , as well as  $R \cup N$  are fragments. These marked fragments in boxes are called canonical, which are defined as follows. A fragment  $F$  is non-canonical if there are fragments  $X, Y, Z$  with  $X$  and  $Y$  are in sequence,  $F = X \cup Y$ , and  $F$  and  $Z$  are in sequence; otherwise  $F$  is called canonical fragment.

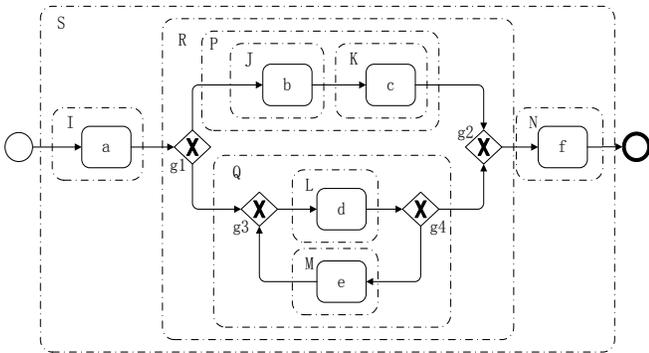


Figure 1. An Example of process fragment

It has been proved that two canonical fragments are either nested or disjoint [8], thus process fragments of a process model can constitute a tree structure which is called process structure tree, which can be derived in linear time using the cyclic equivalent algorithm [9]. The process structure tree of the example process is illustrated in Figure 2. Note that a node with single entry edge and single exit edge can also be regarded as a process fragment.

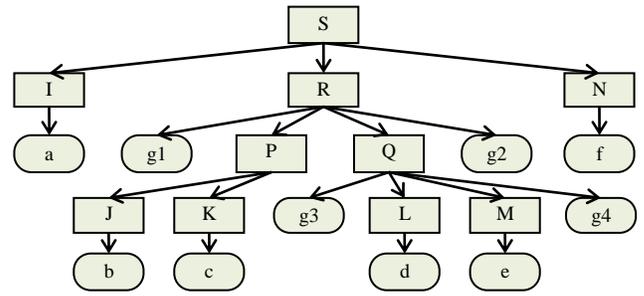


Figure 2. The process structure tree of the example process

### III. FRAGMENT-BASED STRUCTURAL SIMILARITY ANALYSIS

This section deals with that given two process models and a candidate set of matching relations between their fragments, how we can evaluate the similarity value of these two models. The method to construct matching relationship between process fragments will be illustrated later in the next section.

We use the concept of graph-edit distance [11] to evaluate the global matching score of two process models. The graph-edit distance between two graphs is the minimal number of edit operations that is necessary to get from one graph to the other. The existing graph-edit similarity metric do NOT support the consideration of matching relations between process fragments, so we extend the original definition based on the above edit operations we defined.

#### Definition 3 (Extended graph-edit distance similarity).

$G_1 = (N_1, E_1, T_1, \Omega_1, \alpha_1)$  and  $G_2 = (N_2, E_2, T_2, \Omega_2, \alpha_2)$  are two PMGs and let  $M$  be the subset of fragment matching relations. Let  $P_1 \subseteq G_1$  be a process fragment in  $G_1$ , and  $n_1, e_1$  be a node and an edge in fragment  $P_1$ , then  $P_1$  is a **substituted fragment** ( $n_1$  and  $e_1$  is **substituted node** and **substituted edge** respectively) iff  $\exists P_2 \subseteq G_2, M(P_1) = P_2$ . The set of nodes and edges in fragment  $P$  is denoted as  $N^P$  and  $E^P$  respectively. A node is a **skipped node** iff it's not a substituted node. Let process fragments  $P_{11}, P_{12} \subseteq G_1, P_{21}, P_{22} \subseteq G_2$ , and  $e_1 \in E_1$  is the link between  $P_{11}$  and  $P_{12}$ , i.e. be the exit edge of  $P_{11}$  and also the entry edge of  $P_{12}$ , denoted as  $(P_{11}, P_{12})$ .  $e_1$  is a **matched edge** iff  $\exists (P_{21}, P_{22}) \in E_2, M(P_{11}) = P_{21} \wedge M(P_{12}) = P_{22}$ . Note that we consider that the matched edges also exist in the other model. Any other edge except for substituted edge and matched edge is called **skipped edge**. Let  $subn, skipn, sube$  and  $skipe$  be the sets of each kinds of nodes and edges, and  $\omega_{subn}, \omega_{skipn}, \omega_{sube}$  and  $\omega_{skipe}$  be the weights of substituted nodes, skipped nodes, substituted edges and skipped edges respectively. The **extended graph-edit distance similarity** of  $G_1$  and  $G_2$  induced by the mapping  $M$  is:

$$EGSim(G_1, G_2, M) = 1.0 - \frac{\omega_{subn} \cdot f_{subn} + \omega_{skipn} \cdot f_{skipn} + \omega_{sube} \cdot f_{sube} + \omega_{skipe} \cdot f_{skipe}}{\omega_{subn} + \omega_{skipn} + \omega_{sube} + \omega_{skipe}}$$

$$f_{subn} = \frac{\sum_{(P_1, P_2) \in M} [1.0 - Sim(P_1, P_2)] \cdot (|N^{P_1}| + |N^{P_2}|)}{|subn|} \quad f_{skipn} = \frac{|skipn|}{|N_1| + |N_2|}$$

$$f_{skipe} = \frac{|skipe|}{|E_1| + |E_2|} \quad f_{sube} = \frac{\sum_{(P_1, P_2) \in M} [1.0 - Sim(P_1, P_2)] \cdot (|E^{P_1}| + |E^{P_2}|)}{|sube|}$$

Here we illustrate the basic procedure based on a greedy strategy to compute the maximum metric score by selecting an optimal subset. Initially, all candidate matching relations are added to the CanPairs and the optimal set  $M$  is empty. In each

iteration, a new EGSim is computed for adding every pair in CanPairs into M to see which pair lead to a highest EGSim. This pair is added to the optimal mapping M and any other pair overlapping with this pair is removed from the CanPairs. The algorithm terminates when there is no group pair in CanPairs that can increase the metric score EGSim any more.

#### IV. CONSTRUCTING MATCHES OF PROCESS FRAGMENTS

Since there also exists some complex matching relations (i.e. matching between more than one node in each model) between process models, these matching relations need to be identified based on the node matches. Details about similarity metric between nodes can be achieved in [1].

It is clear that not all types of compositions of nodes have a sense, so we propose the prerequisite of grouping elements based on the concept of single-entry-single-exit (SESE) process fragments [8][9]. A canonical process fragment or consecutive canonical fragments may express complete and independent business logic to a great extent. Therefore, all process fragments are regarded as candidates of node combination. Considering the process in Figure 1, the candidate set of fragments includes: {J}, {K}, {L}, {M}, {P}, {Q}, {R}, {I}, {N}, {I, R}, {R, N}, {I, R, N}.

Based on the candidate set obtained by the above condition, we need a criterion of whether there exists a matching relation between two fragments in distinct process models. The matching relations between leaf fragments have been achieved in node matching relations set. For these non-leaf fragments, we hold that the similarity of two fragments should at least higher than the similarity of matching relations between their children fragments. Besides, to reduce searching space, one basic prerequisite is proposed, i.e. there should be at least one matching relation among their containing fragments.

The criterion requires a definition of similarity evaluation between two process fragments. Referring to node similarity, the similarity between groups, based on their attributes and the context of adjacent nodes, can be defined analogously. The major challenge of attribute similarity part is that different attributes may have different composing policies, e.g. the attribute LABEL of a group can be simply merged by the name label of the containing nodes while the merging of attribute INPUT or OUTPUT should ignore the input or output data produced or consumed inside the group itself.

Here we can summarize the algorithm of discovering fragment matching relations. First of all, the process structure trees (PST) of two process models are constructed by the cyclic equivalent algorithm [9], while the detail of this algorithm is not presented here. Then we can get the candidate set of fragments for each process model in a bottom-up order of the tree. Finally, the fragments in two candidate sets are checked one by one according to their sequence, and if the pair satisfies the criterion of building matches, the fragment pair with their similarity value is added to the fragment matching relations set.

#### V. EXPERIMENTAL EVALUATION

We evaluate our approach by the real-world process models from BPM Academic Initiative (BPM AI) [12] which is a joint effort of academic and industry partners that offers a process

modeling platform for teaching and research purposes. We selected 40 pairs of process models captured in BPMN and these model pairs are in various degree of similarity. We obtained the similarity assessment value of each process model pair by using a questionnaire that was distributed to 10 academic process modelers, including researchers and post-graduate students in the realm of process modeling. In the questionnaire, the modelers were asked to decide the similar degree from score 1 to 6. We obtained the manual similarity assessment value by a weighted average of the scores provided by the 10 participants according to their modeling experience.

We create two different analyzers to conduct a comparative experiment. The **Analyzer A** measures graph-edit distance similarity based on node matching relations which is the method in [1][3]. The **Analyzer B** applies our fragment-based similarity measuring method. Our experimental evaluation is conducted by computing the similarity value of each process model pair using these two analyzers we defined and then their computing efficiency and correlation with the manual analysis results can be obtained. We will apply the weight parameters achieved by our earlier research work [13] to these analyzers.

TABLE 1. COMPARATIVE RESULTS OF SIMILARITY COMPUTING

Analyzer name	Correlation coefficient	Time cost
Analyzer A	0.805	628 ms
Analyzer B	0.868	1817 ms

The results are shown in Table 1. Note that all the I/O operations (e.g. reading process models) are excluded from the cost time statistics. From the results we can see that the **Analyzer B** which fully applied our approach shows a better correlation coefficient result and the time efficiency of the method is also acceptable, i.e. about 45 milliseconds for each model pair in average. Figure 3 shows the correlation between the similarity degree (using the optimal correlation of analyzer B) and the similarity assessment as obtained from the questionnaire.

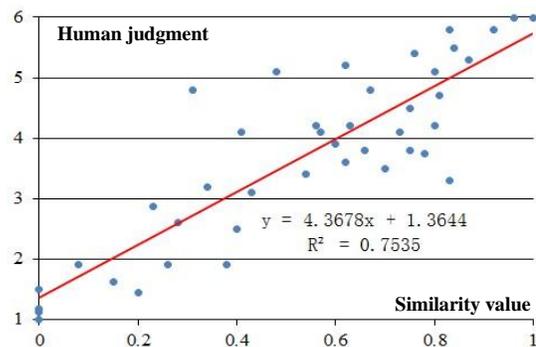


Figure 3. Correlation analysis result of Analyzer B

#### VI. RELATED WORK

Our approach is related to two closely interrelated topics in the research domain of managing large collections of business process models [7], i.e. business process model alignment and business process similarity search.

The business process model alignment determines which elements in one business process model correspond to which elements in another [14]. Three approaches based on lexical or graph matching are compared in [3], and the result showed that a greedy graph matching technique produces the best effectiveness. The ICoP framework [2] enables the optional creation of matchers from the reusable components which can support complex matching detecting, but the algorithms of the components are not illustrated clearly. Our earlier research work [13] proposes a matching technique to support fast detecting complex matches. Our fragment-based method could probably improve these techniques.

Business process similarity search is another closely related topic. A previous paper [15] of our research group utilizes the idea of similarity propagation to measure the process similarity, but the method is restricted to a specific modeling language and only node matches are supported. Remco Dijkman etc. presented and evaluated three similarity metrics and the result showed that the structural similarity slightly outperforming the node matching and behavioral similarity [1]. A fast business process similarity estimating based on characteristic model features presented three techniques to improve the efficiency of current graph-based algorithm [4]. Four major graph matching algorithms for business process model similarity search are evaluated in [16] and result shows that a greedy algorithm produces an effective result with best efficiency. Some novel efficient algorithms for similarity search have been proposed recently. There is an initiative to conduct similarity measures based on a tree-based index organized structure of process models in a collection [17] and clustering techniques were used to group similar process models in [18] to enable the comparison of process clusters rather than each individual process in isolation.

## VII. CONCLUSION

Various techniques have been used to manage the large collections of process models. This paper focuses the problem of how to measure the similarity and construct matching relations more effectively between two process models. To resolve the problem of exponential complexity to construct appropriate matching relations between node compositions, we proposed a graph-edit distance similarity metric based on the SESE process fragments, and a greedy algorithm is employed to compute the similarity value and construct the optimal matching relations. As a matching relation candidate set of fragments is required for similarity evaluation, we present a method to construct matching relations of fragments by process structure tree. A comparative experiment is performed to evaluate the approach in this paper. The result shows that our approach have a positive impact on the accuracy of similarity evaluation comparing with current methods based on node matching, and the efficiency of our technique is acceptable.

In future research, we aim to improve the effectiveness of our approach by conducting a more comprehensive experiment evaluation by huge amount of process models in various model repositories. Secondly, the further accuracy and efficiency improvement for the existing structural process similarity evaluation is another direction.

## VIII. ACKNOWLEDGMENTS

This work is supported by Chinese National Natural Science Foundation (Grant No. 61170087 and Grant No. 61370058). Thanks for the valuable comments of Professor Xavier Blanc from University Bordeaux 1.

## REFERENCES

- [1] Dijkman, Remco, et al. Similarity of business process models: Metrics and evaluation. *Information Systems* 36.2 (2011): 498-516.
- [2] Weidlich, Matthias, Remco Dijkman, and Jan Mendling. The ICoP Framework: identification of correspondences between process models. *Advanced Information Systems Engineering*. Springer Berlin Heidelberg, 2010.
- [3] Dijkman, Remco, et al. Aligning business process models. *Enterprise Distributed Object Computing Conference, 2009. EDOC'09*. IEEE International. IEEE, 2009.
- [4] Yan, Zhiqiang, Remco Dijkman, and Paul Grefen. Fast business process similarity search. *Distributed and Parallel Databases* 30.2 (2012): 105-144.
- [5] van Dongen, Boudewijn, Remco Dijkman, and Jan Mendling. Measuring similarity between business process models. *Advanced Information Systems Engineering*. Springer Berlin Heidelberg, 2008.
- [6] Kunze, Matthias, Matthias Weidlich, and Mathias Weske. Behavioral similarity—a proper metric. *Business Process Management*. Springer Berlin Heidelberg, 2011. 166-181.
- [7] Dijkman, Remco M., Marcello La Rosa, and Hajo A. Reijers. Managing large collections of business process models - current techniques and challenges. *Computers in Industry* 63.2 (2012): 91-97.
- [8] Vanhatalo, Jussi, Hagen Vdzer, and Frank Leymann. Faster and more focused control-flow analysis for business process models through sese decomposition. *Service-Oriented Computing—ICSOC 2007*. Springer Berlin Heidelberg, 2007. 43-55.
- [9] Johnson, Richard, David Pearson, and Keshav Pingali. The program structure tree: Computing control regions in linear time. *ACM SigPlan Notices*. Vol. 29. No. 6. ACM, 1994.
- [10] Thom L H, Reichert M, Iochpe C. Activity patterns in process-aware information systems: basic concepts and empirical evidence. *International Journal of Business Process Integration and Management*, 2009, 4(2): 93-110.
- [11] Hart, Peter E., Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on* 4.2 (1968): 100-107.
- [12] BPM Academic Initiative. <http://bpt.hpi.uni-potsdam.de/BPMAcademicInitiative>. (accessed 9 Dec 2013).
- [13] Ling, Jimin, Li Zhang, and Qi Feng. Business Process Model Alignment: An Approach to Support Fast Discovering Complex Matches. *Enterprise Interoperability VI*. Springer International Publishing, 2014. 41-51.
- [14] Brockmans, Saartje, et al. Semantic Alignment of Business Processes. *ICEIS* (3). 2006.
- [15] Gao, Juntao, and Li Zhang. On Measuring Semantic Similarity of Business Process Models. *Interoperability for Enterprise Software and Applications China, 2009. IESA'09*. International Conference on. IEEE, 2009.
- [16] Dijkman, Remco, Marlon Dumas, and Luciano Garc a-Banelos. Graph matching algorithms for business process model similarity search. *Business Process Management*. Springer Berlin Heidelberg, 2009. 48-63.
- [17] Kunze, Matthias, and Mathias Weske. Metric trees for efficient similarity search in large process model repositories. *Business Process Management Workshops*. Springer Berlin Heidelberg, 2011.
- [18] Rinderle-Ma, Stefanie, Sonja Kabicher, and Linh Thao Ly. Activity-Oriented Clustering Techniques in Large Process and Compliance Rule Repositories. *Business Process Management Workshops*. Springer Berlin Heidelberg, 2012.

# A Method for Verifying the Consistency of Business Rules Using Alloy

Denilson dos Santos Guimarães

Post-Graduate Program in Informatics (PPGI)  
Federal University of Rio de Janeiro  
Rio de Janeiro, Brazil  
denilson.guimaraes@ppgi.ufrj.br

Eber Assis Schmitz and Antônio Juarez Alencar

Post-Graduate Program in Informatics (PPGI)  
Federal University of Rio de Janeiro  
Rio de Janeiro, Brazil  
{eber , juarez}@nce.ufrj.br

Priscila M.V.Lima and Alexandre Luiz Correa

Post-Graduate Program in Informatics (PPGI)  
Federal University of Rio de Janeiro  
Rio de Janeiro, Brazil

**Abstract**—Agile enterprises have been investing heavily in the identification, representation and documentation of business rules (BRs), so that they can be known, understood and used all over the enterprise. In this respect a question of great concern in the development of information systems (IS) is the early identification of BR specification errors. This is due to its impact on the IS final cost, which can be considerable if the error identification occurs at a later time when the BRs have already been deployed. This paper presents a method for the verification of the logical consistency of a set of BRs. The method is particularly useful when the business domain encompasses a variety of types of BR specification errors. The set of BRs to be checked for consistency is translated into an Alloy Model, which the Alloy Analyzer uses as input to generate examples and counterexamples. The output of the Alloy Analyzer is then used for the identification of inconsistencies among the business rules. The proposed method makes it possible to have an early insight into inconsistencies that may exist in the rule model, allowing for a substantial reduction in rework costs.

*Business Rules; Verification; KnowledgeBase; Alloy; SBVR; ABRD*

## I. INTRODUCTION

Business rules (BR) represent one of the most effective ways to express business knowledge and, nowadays, agile enterprises are reaching to identify, represent and document them, so they can be known and clearly used all over the enterprise [10]. Thus, identifying mistakes in business rules is increasingly becoming a question of great concern for the development of information systems (IS), impacting on their final costs, especially whenever this identification occurs at a later time, increasing these figures to hundreds of billions of dollars per year.

The typical inconsistencies in business rules specification are: redundancy, overlapping, conflict and incompleteness. Redundancy occurs when two rules have the same set of conditions (with conditions possibly arranged in a different order) and the same conclusion. Then, one of them is said to be redundant [2]. Overlapping occurs when one rule is wholly or partly contained within another [6] or when two rules have the

same conclusion and one rule has a more restrictive condition. The more restrictive rule can be subsumed by the less restrictive rule on the grounds that whenever the former succeeds, the latter also succeeds [2]. Conflict occurs when two or more rules produce contradictory results [6] or their conditional parts consist of the same set, but the conclusions are mutually exclusive [2]. Incompleteness occurs when all the information necessary to produce a conclusion does not exist. This may be caused by gaps left inadvertently, uncertain knowledge or lost track of the grown knowledge base [2].

According to Hodges [8], beliefs can be expressed by declarative sentences. A set of beliefs is called *consistent* if there is some possible situation in which all the sentences are true.

Alonzo Church proved in 1936 that is impossible to prove a theorem by a systematic method for testing the consistency of sets of sentences written in first-order logic [8]. However, it's still possible to prove the inconsistency of these sets.

Similarly, Edsger W. Dijkstra stated that program testing can be used to show the presence of bugs, but never to show their absence [7]. Unfortunately, most bugs in code elude testing.

In order to test the consistency of a BR set, we make a simple assumption: all inconsistencies will involve a simple pair of rules. In this case, it will be necessary to check the consistency of pairs of sentences. The computational complexity of this calculation will be in the order of the quadratic growth, i.e., its values grow proportionally to the square of the function argument, represented by  $\Theta(n^2)$ . For instance, when  $n$  represents one thousand sentences, the number of operations will be in the order of one million [15]. This figure will certainly grow even more, when the consistency checking involves groups of three or more sentences.

In order to facilitate this great number of necessary operations, it's recommended to use a different approach, like the *instance finding*, proposed by the analysis underlying the Alloy tool. Rather than attempting to construct a proof that an assertion holds, it looks for a refutation, by checking the assertion, looking for a particular case in which the assertion is

found not to hold. That case is reported as a *counterexample* [7].

Instance finding has far more extensive coverage than traditional testing and it tends to be much more effective at finding bugs, because most bugs have small counterexamples. In other words, only a small counterexample is necessary to prove that an assertion is invalid. To make the instance finding feasible, it's needed to define a *scope* that limits the size of instances considered. Even a small scope usually defines a huge space of instances. Systems that fail on large instances almost always would fail on small ones with similar properties, even if such small instances don't occur in practice. So, by checking all small instances, the large ones will be effectively checked also. This is called by Jackson as the "*small scope hypothesis*" [7].

Different approaches to the automated verification of a set of BRs have been presented in the literature but all of them, at a certain point, demand complex implementation or have logic limitations, making their use non-intuitive or limited to skilled programmers, with a non-novice knowledge of mathematical logic and proving methods.

This article intends to present an easy-to-use method that is suitable to the early verification of the consistency of a BR rule set. The method can precisely show the inconsistency generated by a BR specification. Moreover, the inclusion of new rules or the updating of the existing ones can be easily absorbed by the verifying procedure. This turns the method into a robust and reliable option for BRs analysts.

The remainder of this paper is organized as follows. Section 2 presents the conceptual framework. Section 3 describes the conceptual method to verify a set of BRs using the Alloy tool. Section 4 illustrates the method with an example. Finally, Section 5 presents the final remarks.

#### A. Related Work

EVA Project [1] is a reference in the verification and validation of BRs. The use of meta-knowledge (i.e. knowledge about the knowledge) describes restrictions necessary to validate the redundancy, consistency, completeness and correction of a knowledge base. Besides being very complete (with 28 different criteria), this project was extended to non-monotonic logic, bringing it closer to the real world. The processing output is, consequently, complex. The method proposed in this paper explores the utilization of a "model finder", whose output is based on intuitive examples and counterexamples.

PREPARE method [2] uses predicate/transition nets to represent knowledge. These nets are special classes of Petri nets and verification is done through syntactic patterns of recognition. These nets are a graphic representation of predicate logic. Consequently, these graphic representations don't support universal quantifiers, as the Alloy tool does.

NuSMV2 [3] is a tool that extends the original SMV of CMU. With a very efficient BDD algorithm, it was used successfully in hardware and software verification. Unfortunately, its programming language doesn't support more complex structural types, such as relational, for example. This forces the developer to codify relations using arrays of primitive types. The Alloy tool offers the developer relational, arithmetic and logic types.

NIET method [4] is particularly critical with inference machine usage. It reduces the significance of these machines to merely correct the ordination of BRs. It proposes an IDE as repository, verification, validation and code generation in different languages of BRs. Through this method, the ordination solution is turned into a development concern. There is great concern with the performance and cost of the inference machines and with their replacement by an IDE, proposed by this method. The adopted strategy in our work doesn't require the usage of any IDE and performance concerns aren't addressed in our work.

CPKSA [5] is study that proposes the use of group decision, with an algorithm that has a confidence factor. It's only suitable in situations with BRs conflicts that cannot be resolved.

NL2AlloyviaOCL [16] is a tool that generates Alloy expressions from English text statements and a corresponding UML model. But it doesn't have a method to analyze the consistence of BRs generated using this tool.

There are several commercial BR management and simulation tools, but they do not focus on the early verification checking of BR models.

## II. CONCEPTUAL FRAMEWORK

The method presented here for verifying a set of business rules, in order to allow for the early identification of inconsistencies into these rules, requires the specification of a representation of BR, the development of the Alloy Model and the use of a BR Cycle.

### A. BR Representation

BR are declarations that constrain, derive and give conditions for existence, representing the knowledge of the business. BR are not descriptions of a process or processing. Rather, they define the conditions under which a process is carried out or the new conditions that will exist after a process has been completed [6]. BR can be represented using Semantics of Business Vocabulary and Rules (SBVR) [12] or using business language. SBVR use is suitable for business experts, since it allows the representation of business vocabulary and rules using controlled natural language. Business language is more intuitive to business stakeholders and business analysts, although it may be used in an unstructured way. Both representations generate the business vocabulary, composed by terms and facts about them, which will be used to formulate the BR.

### B. Alloy Model

Alloy is a language for describing structures and a tool for exploring them. An Alloy Model is a collection of constraints that describes (implicitly) a set of structures. Alloy's tool, the Alloy Analyzer, is a solver that takes the constraints of a model and finds structures that satisfy them. It can be used both to explore the model, by generating sample structures (here called examples), and to check properties of the model by generating counterexamples [11].

The gross structure of an Alloy Model consists of [7]:

- Signature declarations, labeled by the keyword *sig*, each one representing a set of atoms, and may also introduce some fields, each representing a relation.
- Constraint paragraphs, labeled by the keywords *fact*, *fun* and *pred*, which record various forms of constraints and expressions.
- Assertions, labeled by the keyword *assert*, that record properties that are expected to hold.
- Commands, labeled by the keywords *run* and *check*, with the instructions to the Alloy Analyzer.

The signature declarations set up a classification hierarchy which can be extended into disjoint subsets of these sets, using the keyword *extend*. Marking an extended signature declaration using the keyword *abstract*, indicates that the set has no elements of its own that do not belong to its extensions [7].

A *fact* paragraph records a constraint that is assumed always to hold. An *assert* paragraph introduces a constraint that is intended to follow from the facts of the model. A *pred* paragraph defines a constraint, in the form of a logical predicate. The *run* command instructs the Alloy Analyzer to attempt to find a solution (or example) to the constraints, usually grouped together in a test predicates. The *check* command tells the Alloy Analyzer to find a counterexample to an assertion [7].

### C. BR development cycle

For developing business rules-based applications, it's recommended using a well documented and structured approach. The Agile BR Development methodology (ABRD) is the industry's first free, vendor-neutral methodology and provides a full rule lifecycle, from discovery to governance, by using an "agile", iterative approach. ABRD activities fall into five categories described below, each of these being run multiple times as the cycle is executed [13]:

- Rule Discovery: aims to develop simple modeling artifacts (like rule descriptions, business entity diagrams and business process maps) and is an iterative process, that will identify a subset of rules and document them, as opposed to spending months figuring out all the rules up front and producing a huge document [14].
- Rule Analysis: the goal is to understand the meaning of the rule as stated by the business person and subject matter experts and to remove any ambiguity and semantic issue, preparing the rules for the future implementation [14].
- Rule Design: the purpose is to take a first complete pass through the development process, confront the main design issues and lay the groundwork for future refinements, including architecture aspects and initial prototyping [14].
- Rule Authoring: looks at the general issues surrounding expressing BR in a technology-independent way, particularly addressing those related to the business vocabulary itself and to the rule structures [14].
- Rule Validation: develops unit tests to assess the rule outcome and executes the tests, until all tests succeed,

ensuring that the rules accurately reflect the business intent [14].

- Rule Deployment: addresses common issues around rule deployment, packaging and integration, looking at major deployment and integration considerations, such as transaction support, scalability and data access [14].

The method proposed in this paper is particularly applicable at the Rule Analysis activity.

### D. The Verification of BR

Assuring the quality of BR falls into two general areas: validation and verification [9].

Validation means ensuring the correctness of BR with respect to the business purpose, with the goal to assure that when they are applied, the results will be appropriate in all relevant circumstances. Validation is largely a matter of human inspection and can be achieved by using diagrams, test scenarios and individual analysis of each business rule. Rule validation is about checking whether there are the *right* BR [9].

Verification, on the other hand, means assessing fitness with respect to logical consistency and involves discovering BR (usually two or more in combination) that exhibit some anomaly. Rule verification is concerned about whether the BR are *correct* [9].

In the method proposed in this paper, we will use the Alloy tool to support and automate the verification activity described above. We will focus on the discovery of redundant, overlapping and conflicting rules. The incomplete rules are out of the scope of the method, since it pertains to the validation activity. We will treat the problems of the overlapping and the conflicting rules in the same manner, as they are implicitly related. The mapping of BR artifacts into Alloy structures is not being addressed in this paper, because it would exceed the limit of pages allowed for the paper.

## III. THE METHOD

### A. Method Steps

The proposed method for verifying the consistency of a set of BR consists of the following steps:

1. Identify the BRs and specify them in a structured way, using business language for representation.
2. Build a conceptual model, describing the business concepts and their relations.
3. Transform the rules into an Alloy Model, building a consistent set of BR, as described in Section III-B.
4. Using one rule at a time, introduce the new BR in the model, as described in step 2 of Section III-B, generating possible inconsistency situations. When this happens, update the set of BR, as described in Section III-C.
5. Repeat step 4, until no inconsistencies in BR are found and the set of BR returns to the previous consistent situation.

### B. Generation of the Alloy Model

This section describes the steps necessary to generate the Alloy Model, as described in Section II-B, representing the

consistent set of BR that will be evaluated. The method consists of the following steps:

1. Represent in the Alloy Model every concept and relation built in step 2 of Section III-A, as sets of structures, in the form of Alloy paragraphs *sig*, *extend* and *abstract*, generating a Consistent BR Model (CBRM)
2. Each rule to be introduced in the model is converted to a paragraph *pred*.
3. Execute the *run* command of the Alloy, searching for valid examples of the BR model. If no valid example is found, the rule created an conflict and the conflicting rule must be found, the model must be adjusted.
4. Define an assertion paragraph, negating the same constraint used in previously step 2 and execute the *check* command of the Alloy, searching for invalid examples for the set of BR modeled. If an invalid example is found, then the rule is redundant and the model must be adjusted.
5. Since the rule is consistent and non-redundant with the BR model, it can converted to an Alloy paragraph *fact*, generating a new CBRM.

### C. Generation of the Inconsistency Scenarios

Given a CBRM generated through the mechanism described in Section III-B, each new BR must be introduced in this model using the same concepts and the rules already represented in Alloy Model.

The method allows the detection of the following inconsistency scenarios :

1. Conflicting rules (step 3 of Section III-B): when the Alloy Analyzer indicates that the model is inconsistent. In this case, the rule must be checked against every other possibly conflicting rule in the model. This can be done by creating a test predicate containing the disjunction of the rule and the possibly conflicting rule. The *run* command is executed again and, if the model becomes consistent, the conflict between the rules is proven.
2. Redundant rules (step 4 of Section III-B): The same procedure described above is repeated using a new test assertion containing the rule and the negation of the possibly redundant rule and the *check* command must be executed. If counterexamples are found, the redundancy is proven.

## IV. EXAMPLE

This example focuses on the rules set of a fictional loan process, similar to the one presented in [6]. The verification will be executed following the proposed method, in order to find any inconsistencies that may occur in the set of rules.

### A. Identify the BR

In this step, we specify all the business terms, facts and rules. These rules are listed below, structurally expressed in business language:

A loan exceeding \$1,000 must be approved by a branch manager or above.

A loan exceeding \$1,000 must be approved by a regular manager.

A new loan may not be offered to an overdrawn customer.

An overdrawn customer may not be offered a new loan.

### B. Build the domain conceptual model

Figure 1 shows the class diagram representing the conceptual model of the loan process domain, proposed in [6].

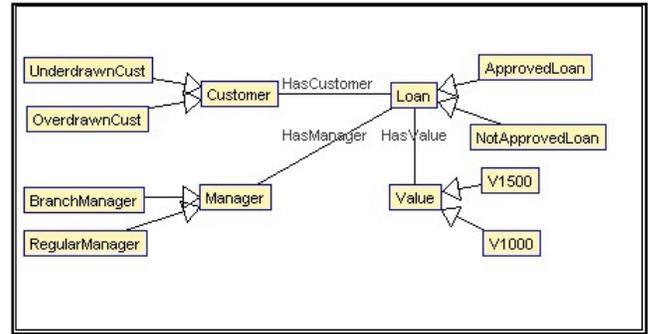


Figure 1. The Conceptual Model

### C. Build the Alloy Model

At this stage of the method, the Alloy Model will be built, representing the conceptual model. For example, from Section IV-B, the concepts Customer, OverdrawnCust and UnderdrawnCust will be written in the Alloy Model, as follows:

```
abstract sig Customer { }
sig OverdrawnCust extends Customer { }
sig UnderdrawnCust extends Customer { }
```

The complete Alloy Model representing the conceptual model is shown in figure 2.

```
module loan

abstract sig Customer { }
sig OverdrawnCust extends Customer { }
sig UnderdrawnCust extends Customer { }

abstract sig Loan {v: Value, c:Customer, mg:Manager}
sig ApprovedLoan extends Loan { }
sig NotApprovedLoan extends Loan { }

abstract sig Value { }
sig V1000 extends Value { }
sig V1500 extends Value { }

abstract sig Manager { }
sig BranchManager extends Manager { }
sig RegularManager extends Manager { }
```

Figure 2. The Alloy Model representing the Cconceptual Model

The consolidated rules will be written in sequence. For each pair of rules described in Section IV-A, the first one will be used to represent the consolidated rule set. For example, from Section IV-A, the rule “A new loan may not be offered to an

overdrawn customer” will be written in the Alloy Model, as a *fact* paragraph, as follows:

```
fact NoLoanOverdrawn {no z:Loan | z.c in OverdrawnCust}.
```

In sequence, a simple constraint (as simple and simultaneously complete as possible, according to the "small scope hypothesis" described in Section I) will be defined. In this example, it will state that there will always be a non-empty set of Loans in our model. This constraint is now written in the Alloy Model as followed:

```
pred SomeLoans {some s:Loan | s in ApprovedLoan one s:Loan | s in NotApprovedLoan}.
```

According to the step 3 of the method, as presented in Section III-A, the *run* command is executed. The Alloy Analyzer informs that the model is consistent, showing instances that exemplify it, as shown in figure 3.

```
module loan
-- A loan exceeding $1,000 must be approved by a branch manager or above.
fact RuleLoan10Approver{all x:Loan | x.v = V1000 implies x.mg in BranchManager}

-- A new loan may not be offered to an overdrawn customer
fact NoLoanOverdrawn {no z:Loan | z.c in OverdrawnCust}

pred SomeLoans {
some s:Loan | s in ApprovedLoan
one s:Loan | s in NotApprovedLoan
}

run SomeLoans for 3 but 2 Value

assert Counterexample {SomeLoans}
check Counterexample for 3 but 2 Value

Executing "Run SomeLoans for 3 but 2 Value"
Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
440 vars. 49 primary vars. 656 clauses. 0ms.
Instance found. Predicate is consistent. 31ms.
```

Figure 3. The Alloy Model and the snapshot after execution of the Alloy Analyzer

#### D. Generate the Inconsistency Scenarios

Now, according to step 4 of the method, as presented in Section III-A, new rules will be introduced, representing the inconsistency scenarios to be evaluated (conflict and redundancy). For each pair of rules described in Section IV-A, we will include the second one in the consolidated rule set generated in step 3 of the method.

#### E. Execute the Alloy Analyzer

The first inconsistency scenario is generated with the inclusion of the conflicting rule “A loan exceeding \$1,000 must be approved by a regular manager.”, described in Section IV-A, written as a *pred* paragraph, as follows:

```
pred RuleLoanRegularApprover {all x:Loan | x.v = V1000 implies x.mg in RegularManager}.
```

According to step 4 of the method, as presented in Section III-A, the *run* command is executed. The Alloy Analyzer informs that the model is still consistent, finding instances that satisfy both conflicting rules. But, when the *assert* and *check* commands are executed, a counterexample is found, proving that the assertion is invalid, as shown in figure 4.

```
module loan
-- A loan exceeding $1,000 must be approved by a branch manager or above.
fact RuleLoanBranchApprover{all x:Loan | x.v = V1000 implies x.mg in BranchManager}

-- A loan exceeding $1,000 must be approved by a regular manager.
pred RuleLoanRegularApprover{all x:Loan | x.v = V1000 implies x.mg in RegularManager}

-- A new loan may not be offered to an overdrawn customer
fact NoLoanOverdrawn {no z:Loan | z.c = OverdrawnCust}

run RuleLoanRegularApprover for 3 but 2 Value

assert Counterexample {RuleLoanRegularApprover}
check Counterexample for 3 but 2 Value

Executing "Run RuleLoanRegularApprover for 3 but 2 Value"
Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
423 vars. 44 primary vars. 644 clauses. 16ms.
Instance found. Predicate is consistent. 0ms.

Executing "Check Counterexample for 3 but 2 Value"
Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
458 vars. 47 primary vars. 716 clauses. 0ms.
Counterexample found. Assertion is invalid. 15ms.
```

Figure 4. The Alloy Model with the conflicting scenario and the snapshot of the execution of the Alloy Analyzer

By updating the model, joining the conflicting rules, using the *or* connector and executing the *run* command again, the Alloy Analyzer informs that the model is now consistent. Executing the *assert* and *check* commands, no counterexample is found, proving that the assertion is valid. However, the instances found only exemplify the ambivalence of both rules written in the model, demonstrating the conflict generated, as shown in figure 5. This inconsistency must be resolved in the validation activity of the BR Cycle.

The next inconsistency scenario is generated with the inclusion of the redundant rule “An overdrawn customer may not be offered a new loan.”, described in Section IV-A, written as a *pred* paragraph, as follows:

```
pred NoLoanOverdrawn02 {all od:OverdrawnCust | #od.l = 0}
```

According to step 4 of the method, as presented in Section III-A, the *run* command is executed. The Alloy Analyzer informs that the model is still consistent, finding instances that satisfy both redundant rules. Executing the *assert* and *check* commands, no counterexample is found, indicating that the inclusion of the new rule didn’t affect the results of the execution of the original rules set, as shown in the figure 6.

By updating the model, denying the original rule, using the *not* connector and executing the *run* command again, the Alloy Analyzer informs that the model is now inconsistent, not being possible to find any instances that satisfy both redundant rules. Executing the *assert* and *check* commands, a counterexample is found, informing that the assertion is invalid and proving the redundancy, as shown in figure 7.

```

module loan
-- A loan exceeding $1,000 must be approved by a branch manager or above.
fact RuleLoanBranchApprover{
all x:Loan | x.v = V1000 implies x.mg = BranchManager
}

-- A loan exceeding $1,000 must be approved by a regular manager or by a branch manager or above.
pred RuleLoanRegularApprover{
all x:Loan | x.v = V1000 implies x.mg = BranchManager or x.mg = RegularManager
}

-- A new loan may not be offered to an overdrawn customer
fact NoLoanOverdrawn {
no z:Loan | z.c = OverdrawnCust
}

run RuleLoanRegularApprover for 3 but 2 Value

assert Counterexample {RuleLoanRegularApprover}
check Counterexample for 3 but 2 Value

Executing "Run RuleLoanRegularApprover for 3 but 2 Value"
Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
453 vars. 44 primary vars. 692 clauses. 0ms.
Instance found. Predicate is consistent. 15ms.

Executing "Check Counterexample for 3 but 2 Value"
Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
488 vars. 47 primary vars. 783 clauses. 0ms.
No counterexample found. Assertion may be valid. 0ms.

```

Figure 5. The Alloy Model with the conflicting scenario corrected and the snapshot of the execution of the Alloy Analyzer

```

module loan
-- A loan exceeding $1,000 must be approved by a branch manager or a regular manager.
fact RuleLoanBranchApprover{all x:Loan | x.v = V1000 implies x.mg in BranchManager or x.mg in RegularManager}

-- A new loan may not be offered to an overdrawn customer
fact NoLoanOverdrawn {no z:Loan | z.c = OverdrawnCust}

-- An overdrawn customer may not be offered a new loan
pred NoLoanOverdrawn02 {all od:OverdrawnCust | #od.l = 0}

run NoLoanOverdrawn02 for 3 but 1 Customer

assert Counterexample {NoLoanOverdrawn02}
check Counterexample for 3 but 1 Customer

Executing "Run NoLoanOverdrawn02 for 3 but 1 Customer"
Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
378 vars. 41 primary vars. 597 clauses. 16ms.
Instance found. Predicate is consistent. 46ms.

Executing "Check Counterexample for 3 but 1 Customer"
Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
384 vars. 42 primary vars. 609 clauses. 16ms.
No counterexample found. Assertion may be valid. 0ms.

```

Figure 6. The Alloy Model with the redundant scenario and the snapshot of the execution of the Alloy Analyzer

## V. CONCLUSIONS

The method proposed in this paper provides both the business analyst and developers with a tool that allows a rapid evaluation of the consistency of a set of BR, in a logical point of view. It also proposes an incremental form of process evaluation of a set of BR, as this set is continuously updated through the lifecycle of BR applications.

By using this method, the most typical inconsistencies in BR (overlapping, redundancy and conflict) can be found, showing the counterexamples found and making the detection and correction easier and faster.

Thus, a consistent set of BR can be generated more easily, since the automated rule verification activity, provided by this method, can guarantee the rule validation activity.

The next challenges are concentrated in evolving this method to make its using easier to non-technical people and provide ways to automate the identification of the rules that become inconsistent with the inclusion of new rules in the consistent set of BR.

```

module loan
-- A loan exceeding $1,000 must be approved by a branch manager or
-- a regular manager.
fact RuleLoanBranchApprover{
all x:Loan | x.v = V1000 implies x.mg in BranchManager or x.mg in RegularManager
}

-- A new loan may not be offered to an overdrawn customer
fact NoLoanOverdrawn {not no z:Loan | z.c = OverdrawnCust}

-- An overdrawn customer may not be offered a new loan
pred NoLoanOverdrawn02 {all od:OverdrawnCust | #od.l = 0}

run NoLoanOverdrawn02 for 3 but 1 Customer

assert Counterexample {NoLoanOverdrawn02}
check Counterexample for 3 but 1 Customer

Executing "Run NoLoanOverdrawn02 for 3 but 1 Customer"
Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
394 vars. 44 primary vars. 626 clauses. 16ms.
No instance found. Predicate may be inconsistent. 0ms.

Executing "Check Counterexample for 3 but 1 Customer"
Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
400 vars. 45 primary vars. 638 clauses. 16ms.
Counterexample found. Assertion is invalid. 31ms.

```

Figure 7. The Alloy Model with the redundant scenario corrected and the snapshot of the execution of the Alloy Analyzer

## ACKNOWLEDGMENT

Our thanks to Fabrício de Martino, for the related work research and the experiments with the Alloy tool.

## REFERENCES

- Chang, C.L. A report on the expert systems validation associate (EVA). Expert Systems with Applications, Volume 1, Issue 3, 1990, 217–230.
- Nguyen, D. and Zang, D. PREPARE: A Tool for knowledge base, verification. IEEE Transactions on Knowledge and Data Engineering, Volume 6, Issue 6, (Dec. 1994), 983-989.
- Cimatti, A., Clarke, E. M., Giunchiglia, E., Giunchiglia, F., Pistore, M., Rovere, M., Sebastiani, R., and Tacchella, A. NuSMV 2: An OpenSource Tool for Symbolic Model Checking. Computer Science Department, 2002, Paper 340.
- Hicks, R.C. The no inference engine theory – performing conflict resolution during development. Decision Support Systems, 2007, vol 43, 435-444.
- Huang, C. and Ling, Y. A Conflict Treatment Model for Uncertainty Rule-based Knowledge. Innovative Computing, Information and Control (ICICIC '07), 2007.
- Morgan, T. Business Rules and information systems – Aligning IT with business goals. Addison-Wesley, 2002.
- Jackson, D. Software Abstractions: Logic, Language and Analysis. The MIT Press, Cambridge, MA, 2006
- Hodges, W. Logic. Penguin Books, 2001
- Ross, R.G. Business Rule Concepts: Getting to the Point of Knowledge. Business Rules Solutions, LLC, 2013
- Souza, M. G. Uma abordagem de regras de negócio baseada em linguagem natural estruturada, 2002, Dissertation (Master in Informatics). Post-Graduation Program in Informatics, Federal University of Rio de Janeiro. Rio de Janeiro, Brazil.
- Jackson, D. Alloy: a Language & Tool for Relational Models, 2012. In: <http://alloy.mit.edu/alloy/>
- Object Management Group. SBVR 1.1 specification, 2013. In: <http://www.omg.org/spec/SBVR/1.1/PDF/>.
- Eclipse Foundation. ABRD 1.2, 2009. In: <http://epf.eclipse.org/wikis/epfpractices/>.
- Boyer, J. and Mili, H. Agile Business Rule Development: Process, Architecture and JRules Examples. Springer, 2011
- Moore, C., Mertens, S. The Nature of Computation, Oxford University Press, 2011.
- University of Birmingham (UK). NL2OCL Project, 2013. In: <http://www.cs.bham.ac.uk/~bxb/NL2OCLviaSBVR/NL2Alloy.html>.

# Making a link between strategy and process model collections: a multi-layered approach

Felipe Diniz Dallilo, João Porto de Albuquerque  
Dept. of Computer Systems, University of São Paulo  
São Carlos, Brazil  
{fdallilo, jporto}@icmc.usp.br

Marcelo Fantinato  
School of Arts, Sciences and Humanities, University of  
São Paulo, São Paulo, Brazil  
m.fantinato@usp.br

**Abstract**— This paper proposes a multi-level approach that links strategic goals with an organization's collection processes, based on the Business Motivation Model (BMM) and the Business Process Management Notation (BPMN). A multi-layered metamodel is proposed together with a web-based support tool to manage the entire approach. The approach is illustrated and validated through a proof of concept application in a real process of a large company that provides IT services, where the advantages of the proposed model are made clear. Furthermore, the approach is evaluated by means of a systematic comparison with related work and through an empirical evaluation involving users in a real organization.

**Keywords**- Business Process Management; BPMN; BMM; Multilayer; Process Collections; Strategic Alignment

## I. INTRODUCTION

A number of organizations seek to establish a process-based framework to manage their business effectively. Given the advantages of having a process-based organization, the Business Process Management (BPM) approach has become highly valued by managers [2]. This is because it allows the measurement, monitoring, control and analysis of business processes, and can thus enable a company to deliver value to its customers by ensuring continuous improvement in processes.

Before companies can model and manage their processes effectively, they must define those that are efficient and effective with regard to achieving organizational goals and meeting customer expectations [1]. However, organizations often have to handle a large number of cases, and this factor not only impedes the management of specific processes but also the relationship between low-level models and the strategic objectives of the organization. The lack of tools that are able to manage collections of processes exacerbates the difficulties of management, since it is hard to know whether the data from processes can be gathered simply by setting out the strategies involved.

This paper proposes an approach for managing various levels of business processes, and thus allows strategic business objectives to be systematically tied to business process models. The approach consists of a multi-layered metamodel that is supported by a web-based model-editing tool. In the remainder of this paper, Section 2 reviews some related work. The proposed approach is outlined in Section 3. In Section 4, there

is an examination of the tool that is developed to support the approach, Section 5 presents a proof of concept of the approach and tool. Experimental validation is shown in Section 6, while Section 7 finally concludes the paper with suggestions for future work.

## II. BACKGROUND

Any modeling that fails to meet the strategic objectives of the organization can have a serious effect on the subsequent phases of BPM [6]. In the light of this, Lind and Seigerroth [4] propose an approach at several levels, consisting of a model of the overall macro process and a business model that is designed to describe every part of the process in detail. The collection of data needed for the proposed process modeling is carried out together with the different stakeholders in a collaborative way, to avoid any incompatibility between the strategic plans and conceptual models at different levels.

The same authors have conducted research in this area, in other studies [5] where the term "multilayer-thinking" is employed to mitigate the differences between business processes and technology. In addition, they propose a representation of processes at different levels of abstraction that link the business perspective to an IT perspective. In a similar way, the work of Nuffel and Backer [3] adopts a multi-tiered approach, although they also emphasize the need to process multiple perspectives as this can assist in their management.

The research studies outlined above show that the multi-layered approach can be useful for connecting strategies to process models, but what is still missing are research approaches that provide tool support for this form of process modeling [3, 4, 5]. This is important because a change in a high-level process can trigger a chain reaction at other levels and this cannot be mapped or perceived by making manual connections between the levels. Some commercial software packages do include partial support for multi-layered modeling - e.g. ARIS[9], ArchiMate[11] and BEN[10]. However, they fail to include a strict definition of the modeling syntax and do not offer web-support to enable collaborative process modeling (a systematic comparison of our work with these approaches is provided below).

## III. APPROACH

Our approach seeks to fill in some of the gaps pointed out in the previous section, and consists of a multi-layered model,

which is supported by a web-based model-editing tool. This model builds on the multilevel approaches proposed by Mikael Lind and Ulf Seigerroth [4] and Nuffel and Backer [3]. However, in devising our metamodel we employ the Business Motivation Model (BMM) and the Business Process Model and Notation (BPMN) [7, 8], because BPMN is a standard notation in the market and known to be accessible to end users and BMM is of a standard that is recommended by the Object Management Group (OMG). To the best of our knowledge, there are still no approaches that combine BMM with BPMN in a multi-layered model.

In our approach, we represent high-level, strategic information which is linked to a business process diagram in a single unified way. This is accomplished through a representation at levels that allow the high-level information to be linked to the tasks performed in the lower-level process. We decided to combine the first two layers proposed by [4] and [3], since BMM can represent them in a horizontal way, which means our proposal consists of 4 layers, as illustrated in Figure 1.

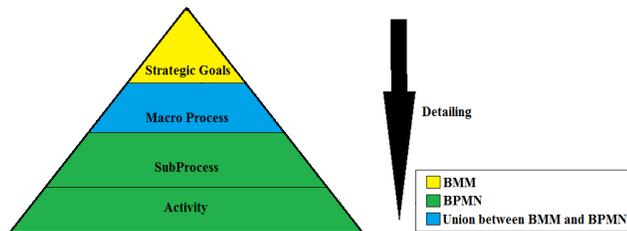


Figure 1: Overview of our approach

This approach allows elements of business plans and strategies and tactics (at the top level) to be connected with the models of business processes, so that the information contained in the process flow can be compared with the strategies and tactics set out for the process. Since it provides traceability, this form of representation enables a manager at the strategic level of an organization to visualize how its strategies and tactics are being tackled in the processes. Conversely, the person who is performing the process at the operational level, is also able to visualize the effect of their tasks on strategic planning.

### A. Metamodel Approach

Figure 2 represents the metamodel of our approach. It consists of a combination of the metamodels of BMM and BPMN, with some additional object types. In this manner, the metamodel makes it possible to define the strategic elements of BMM and establish a connection with the process models defined in the organization with BPMN.

### B. First Layer - Strategic Goals

The first level of our approach is represented by the BMM model, which is similar to organizational reference architectures such as the Zachman Framework, TOGAF, and ARIS Enterprise Architecture Framework [4, 3]. These models give prominence to high-level information that is not represented in the current modeling process notations, e.g. BPMN.

The first level thus aims at employing BMM to survey the business strategic information and define how this information is linked to business processes. This level is important because

it provides a way of adding information to business processes, which is critical to their development and implicit in current process modeling.

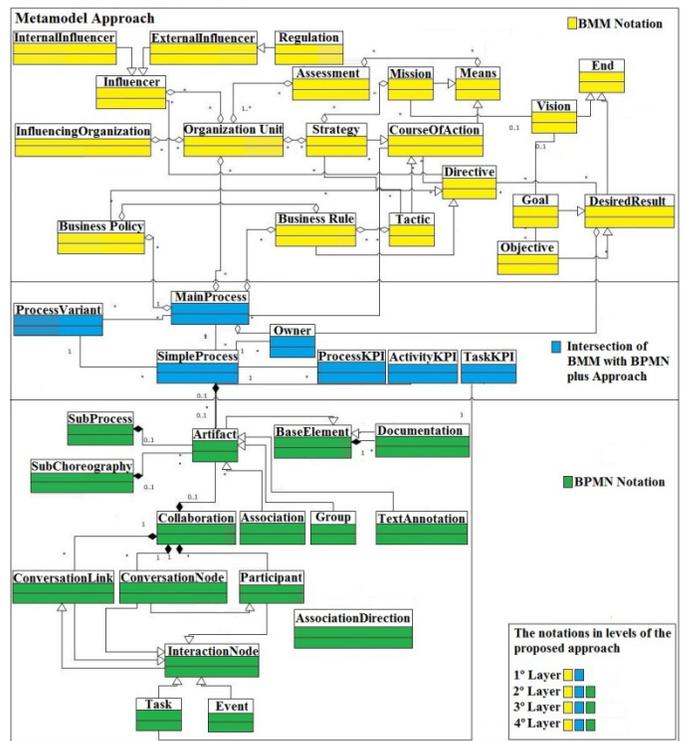


Figure 2: Representation of the metamodel

The following information is included at this level: a) the Business Rules and Policies of Organizational Units or Departments, which make it easier to visualize interdepartmental processes as processes that a particular department operates; b) Strategies and tactics; c) 'Influencers' within internal and external departments; d) KPIs for responsible sourcing. The process is represented at this level as a black box, with all details of the flow of activities being omitted and this responsibility being delegated to lower levels.

At this level, model elements were assigned to represent the owner of the process and the key performance indicators (KPIs) for the process level, activity level and task level. This procedure is based on the assumption that a KPI of the process can be divided into smaller indicators when they are defined at the level of activities and later on, even more fine-grained indicators, when they arrive at the atomic level of the approach.

A process may have N relationships with each element and these elements can be contained in N processes. Figure 3 represents the first level of our modeling approach.

### C. Second Layer - Macro Process

After defining the name of the process and the strategic attributes that it will contain, it is necessary to define its workflow. The second layer of the model represents the specific connections of certain BMM attributes and specifications. At this level, the flow of a business process is represented in detail by means of BPMN, which displays the flow of the process from start to end and its relationships with specific BMM attributes. Another important feature is the

adoption of a good modeling practice that starts at this level. The purpose of this is to show the end-to-end process flow without the need to address the question of its implementation in detail.

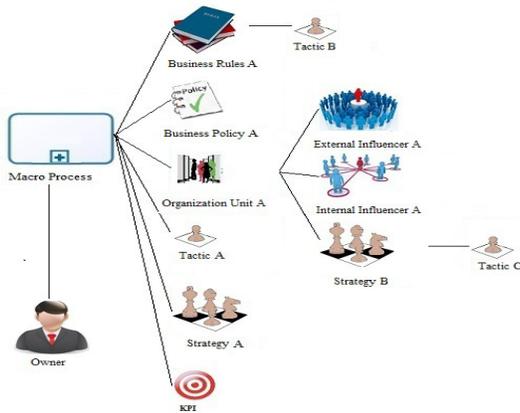


Figure 3: Representation of the first layer of the approach

For this reason, the second level can simplify the description of activities through the creation of sub-processes, while deferring to the next level the definition of the detailed flow of tasks of the sub-process. As such, the second level represents a simplified flow of the process and its relationship with the attributes of Business Rules, Business Policy, Strategy and Tactics, which are linked to the process in the same way as the modeling performed at the first level. The approach converts the Organizational Units that are defined in the first level into lanes in BPMN and these lanes are connected to their influencers and departmental strategies. The process has a key performance indicator to define units of time, although this indicator can be represented in other ways, such as goods produced in a given period of time. This indicator can also be divided to indicate the performance of the sub-processes contained in this process. In addition, it shows the responsibility of the process and the steps that precede and follow the processes studied. Figure 4 represents the second layer of the approach.

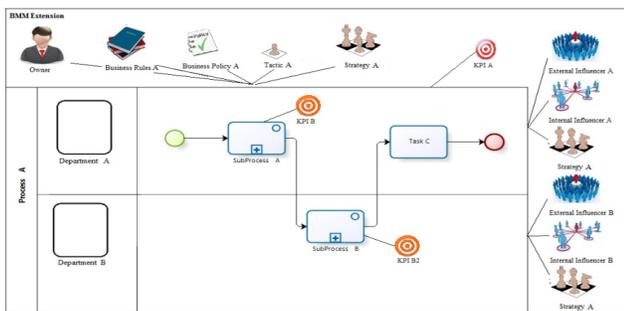


Figure 4: Example of the second layer of the approach

#### D. Third Layer - Sub Process

The third layer is a process involving the further segmentation of the levels of complexity and is based on the BPMN attributes. Basically this level involves the set of activities and tasks that are contained in the sub-process.

One of the benefits of the approach can be compared with the use of similar methods in a programming language. The reason for this is that, if the element needs additional information, it is possible to understand what the method does by just reading one's name. As well as this, the method can be analyzed in detail, and as in the case of the 'sub', the activities and tasks can be analyzed if a greater level of detail is required.

This level displays the activities and tasks contained in the sub without losing the necessary high-level connections obtained from merging the BMM and BPMN.

Moreover, it omits the influencers and strategies employed in the other departments of the process. It also determines the modularization of business processes, and allows the data to be divided so as to cater for individual needs, since low-level details can complicate the interpretation of the process flow. As in the case of the other levels, the attributes can be viewed both before and afterwards. Figure 5 represents the third-layer approach.

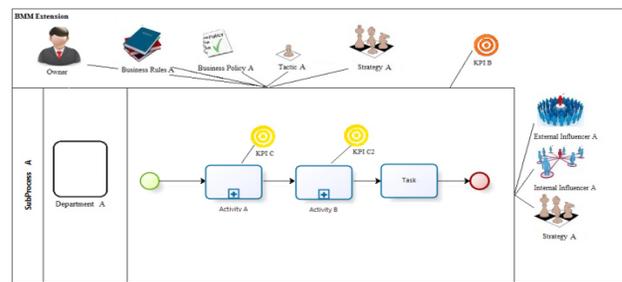


Figure 5: Representation of the third layer of the approach

#### E. Fourth Layer - Activity

The fourth layer represents the atomic level of the segmentation process at different levels of complexity.

This level shows the flow of execution of the tasks involved in the process. As illustrated in Levels 2 and 3 of the approach, at this level there is also a connection with the high-level attributes of the BMM. As in the previous levels, this level acts as the key indicator of performance-linked tasks. Figure 6 represents the fourth layer approach.

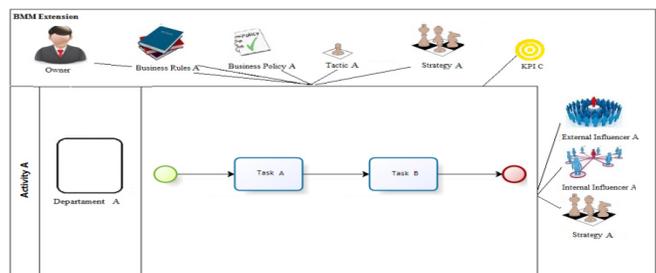


Figure 6: Representation of the fourth layer of the approach

## IV. TOOL

As discussed earlier, we developed a web-based tool to support the whole of the proposed approach. The web-based development was chosen because it allows easy access to users who can model processes collaboratively, by working together with other users and the same model.

The tool was developed in a Java framework with GWT (Google Web Toolkit) for the view layer and Hibernate as the persistence layer data. GWT was selected because it only uses one language (Java) for the application layer client server model by automatically converting all the code for the Javascript client at runtime. As a result, the objects used in the communication layer between the client and the application server, are mapped and remain with Hibernate, thus eliminating the need to convert the objects into a relational structure.

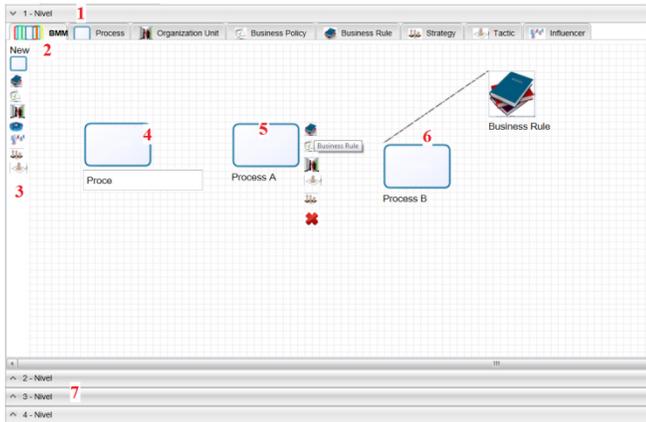


Figure 7: Interface of the support tool

Figure 7 shows the tool interface. The numbers in Figure 7 correspond to the following features:

1) The tool has different panels for each model level, and thus the modeler is only concerned with the current level of modeling and can choose to compare it with other levels when necessary.

2) Within the first ‘accordion’, there are tabs that are responsible for a change in perspective. The first tab carries out an entire modeling of the top-level elements of the process and the tool automatically shows the perspective of the elements by creating collections by type.

3) The side menu allows a new element to be included in the modeling.

4) **Item presents** an example of a newly-created process.

5) After defining the element, the user can select it and see all the possible relationships of the selected object.

6) Example of creating a process with a business rule.

7) When someone selects a specific process, its detailed workflow can be modeled at Level 2. The same is true for the sub-process activities at Level 3 and Level 4.

All of the examples in this paper, together with the proof of concept discussed below, were modeled with the developed tool.

With the aid of the tool, the proposed approach can be represented in a better way, and allow an effective control system to be operated between the attributes and attribute changes. The reason for this is that when creating a strategic element, it is possible to check if any process is listening to it and make any necessary change. An element can be seen where the processes will have an impact, (as is displayed in the

diagram which shows the connecting element together with the accompanying processes).

## V. PROOF OF CONCEPT

Our proof of concept included the modeling of the actual processes found in a large IT service organization in Brazil. The company already uses BPM as an effective management system and BPMN for modeling processes. The high-level strategies of the organization are currently documented in the machines of any managers who complain about the lack of tools or means to connect this information to the process models.

The approach was applied to a process designed to support a customer call center. This process is triggered when a problem arises in systems used by the call center. When confronted with this situation, this center records the problem in a service tool and the support team of the company that provides the service, analyzes the incident. There is a SLA (Service Level Agreement) that sets the maximum amount of time that the service provider has to perform the service required.

Every problem in the call center systems that causes an incident, contains information relating to the systemic failure or inconsistency that prevents an operation from being carried out within the client system.

If a problem is already known, the conventional way of handling an incident is to correct the inconsistency at the database of the system. This is undertaken in accordance with what has been mapped in the Log's Problem by the Second Level (2N) team, which refers to the lessons learned about each problem in the Customer's system. If a problem is not yet known, the Third Level (3N) has to debug the system in the same scenario as the attendant call center. The purpose of this is to identify why the system is not keeping up with the desired activity, and after the problem has been detected, it is necessary to document the information in a log of problems that will be submitted to the Fourth Level (4N) for approval, and to contact the customer (if necessary).

If a particular incident is concerned with a high volumetric demand, an attempt should be made to survey all the incorrect data in this scenario so that there can be a ‘batch’ correction of the problem. The batch correction is carried out from the script, which is a set of SQL commands executed in the database of the client.

This enables the development process to convey Script within the process of the call center support to modify the script of the incident within a specific scenario. The strategy process provides with a fully proactive service for correcting root problems and avoiding various incidents that may have a direct impact on the client's operation. Since this information is related to the clients of the organization who are provided with the services, (through exchanging production data of extreme importance), a number of security measures are taken to secure the endorsement of the customer for their execution.

### A. First Layer

As described earlier, it was evident that the current processes of the organization are not divided into levels and do

not have a relationship with high-level information. In view of this, Figure 8 illustrates how the process of the call center support tool has been modeled on the first level of the proposed approach.

At this level it is possible to understand certain factors that were not previously clear e.g. the role of all the influencers of the various departments. The key performance indicator of this process is a requirement that the incident should be handled within a strict deadline of 72 hours. The business rule requires that the procedure should only be triggered by an incident; the business policy states that the client should be alerted to any changes of the data and responsible for the process (the name has been omitted for reasons of confidentiality).

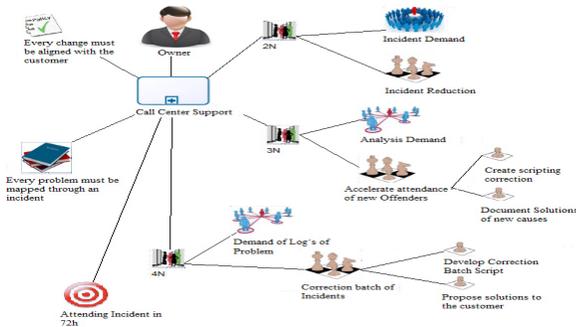


Figure 8: Example of the model in the first layer

### B. Second Layer

Figure 9 shows the entire process that has been modeled on the support call center and its relationship with the BMM attributes. It is also possible to identify the fragmentation of process KPI in Sub-processes KPI and the Owner of this Process.

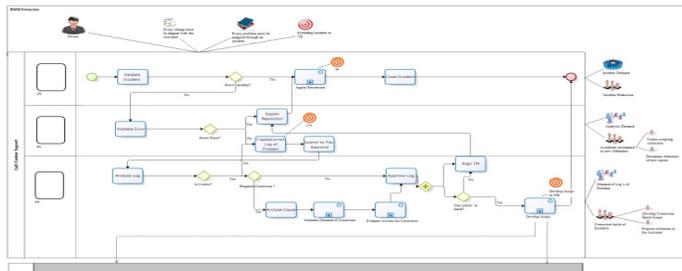


Figure 9: Example of a model in the second layer

### C. Third Layer

Among the sub-processes and tasks defined in the process support call center, is the sub-process 'development Script' that was reconfigured in a more consistent way, since both the processes were considered to be at the same level.

As seen in Figure 10, the department is influenced internally by the demand for Log problems since the sub-process that was analyzed, occurs at the 'team room' level and also has an external influencer. This is because the execution that is based on the production data obtained from the created script, can only be carried out with the client's approval. Finally, the sub-process, together with the performance indicators for creating a script, should take less than 40 hours

and its activities were fragmented by KPIs so that they could handle the performance.

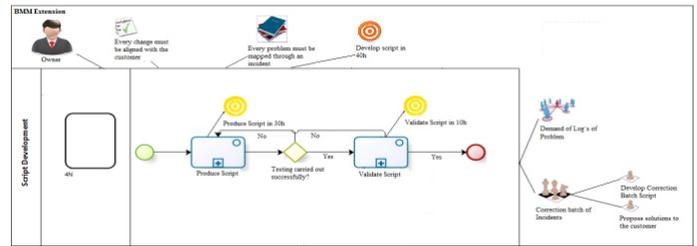


Figure 10: Example of a model in the third layer approach

### D. Fourth Layer

As can be seen in "Produce Script", in the development process, there are several tasks involved in defining a script. With this level of detail, it is possible to have a clear sense of how the tasks are being carried out and also a refined view of the influencers and strategies of this department, (as can be seen in Figure 11). There is also a more detailed flow where an activity can be observed that includes tasks for development, testing and evidence of testing, which results in the production of artifacts of scripts and test plans. This activity has led to the definition of a performance indicator.

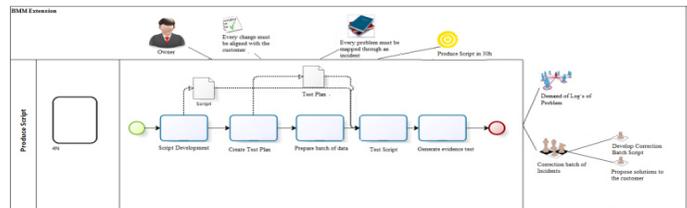


Figure 11: Example of the model in the fourth layer

## VI. EVALUATION

The proposed approach was evaluated by means of a two-fold strategy as follows: a systematic comparison with related work and an experimental evaluation.

### A. Systematic Comparison

Table 1 shows a systematic comparison of our approach with related studies (columns): Aris [9]; (N&B) Multi- abstraction layered business process modeling [3]; (L&S) Multi-Layered Process Modeling for Business and IT Alignment [4]; (Archi) Archimate [11]; BEN [10]; (This) This study.

The criteria for the comparison (the rows in Table 1) were defined as follows: (a) the tool support is available to assist in defining the models; (b) the tool allows collaborative modeling by several different users in the same diagram; (c) the tool must be open source; (d) the tool offers flexibility in choosing the desired language for the modeling; (e) the tool is web-based; (f) the approach provides a framework for representing Business Information; (g) the approach provides a framework for combining information modeling with Business Processes; (h) the approach provides a framework for combining information modeling with Business Process Activities; (i) the approach provides a framework for combining Information modeling with Business Process Tasks; (j) the approach includes a

rigorous metamodel (e.g. in UML); (l) the approach has been applied to real case studies; (m) the approach was recommended by researchers or statements issued by organizations; (n) the approach assists in the management of process collections, and allows the various processes of the organization to be modeled and visualized.

TABLE I: SYSTEMATIC COMPARASION WITH RELATED WORK

	(Aris)	(N&B)	(L&S)	(Archi)	(BEN)	(This)
(a)	X			X	X	X
(b)	X			X		X
(c)				X		X
(d)	X			X		
(e)						X
(f)	X			X		X
(g)						X
(h)						X
(i)						X
(j)		X	X	X	X	X
(l)	X	X	X	X	X	X
(m)	X			X		
(n)	X		X	X	X	X
Total	7/13	2/13	3/13	9/13	4/13	11/13

The comparison thus shows that the approach proposed here clearly outperforms those of existing research studies, and has decisive advantages vis-à-vis commercial software packages. At the same time, there are features of our approach that need improvement such as the flexibility of the modeling language (d) and further empirical evaluation (m).

### B. Experimental Evaluation

After modeling the processes in the approach, an experimental evaluation was carried out. Semi-structured interviews were conducted with people who work with the modeled process. In all, 18 participants were selected who had 3 to 10 years' experience with the processes. The number of participants in the experiment was selected on the basis of their position in the hierarchy of the organization (in Strategic, Tactical and Operational areas), so that they could be representative of the participants at all levels. The interviews were conducted individually and on average lasted 40 minutes.

The results showed that the multi-layered representation can be regarded as useful at the strategic level, since it avoids giving a detailed account of how the processes are run and allows managers to focus on strategies and the process as a whole. It was also considered helpful for people at the operational level as it enabled them to obtain details of how processes are implemented and who it is most suitable for in the wider organizational context. Finally, interviews at a tactical level showed that our approach can help people

understand expectations better and align the strategic level with the operational level through modeling.

Interviewees at the operational level said that if they knew what strategies are linked to their tasks, they could serve as guidelines when making strategic changes. At the same time, people at the strategic level were pleased that they had the ability to monitor what was taking place at the tactical and operational level.

It was clear that with the web-based support tool, the same model could be made available to all employees and also allows diagrams to be edited in a collaborative way. The participants also considered the BMM representation to be intuitive and easy to learn. However, the participants lacked a suitable mechanism for tracking changes in the process models or issuing a system of automatic warnings via e-mail. Another potential area of improvement that was highlighted in the interviews was the need for BMM elements to have more descriptive fields. This could allow a detailed description of the entities, attachment of files, definition of deadlines, dates of creation and the disclosure of other information.

## VII. CONCLUSION

The multi-layered approach proposed in this paper allows higher-level strategic objectives and their relation to the lower-level business processes to be defined by means of a suitable, web-based editing tool. The proof of concept application in a real-world process, together with the experimental evaluation, have shown that the approach is a valuable means of making the link between strategic goals and business process collections. Future work should concentrate on improving the model editor so that it can include other features such as descriptive fields, the tracking of changes, as well as supplying process owners with automated notifications.

## REFERENCES

- [1] Albuquerque, J. P. de ; Christ, M. . Revealing the Sociotechnical Complexity of Business Process Modeling An Actor-Network Theory Approach. In: AMCIS 2012 Proceedings. (2012)
- [2] Alibabaei, Ahmad and Bandara, Wasana and Aghdasi, Mohammad.: Means Of Achieving Business Process Management Success Factors. In MCIS 2009 Proceedings, <http://aisel.aisnet.org/mcis2009/122> (2009)
- [3] Dieter Van Nuffel, Manu De Backer.: Multi-abstraction layered business process modeling, Computers in Industry 63, p. 131-147 (2012)
- [4] Lind, Mikael and Seigerroth, Ulf .: Multi-Layered Process Modeling for Business and IT Alignment, 43rd Hawaii International Conference on System Sciences, IEEE. <http://ieeexplore.ieee.org/> (2010)
- [5] Lind, Mikael and Seigerroth, Ulf .: Process Models as Transformation Vehicle for strategic alignment, MCIS 2010 Proceedings, <http://aisel.aisnet.org/mcis2010/56> (2010)
- [6] Melao, N. and Pidd, M.: A conceptual framework for understanding business processes and business process modelling, Information Systems Journal, pp.105{129 (2000).
- [7]OMG.: Specification Business Motivation Model (BMM). <http://www.omg.org/spec/BMM/> (2010)
- [8] OMG.: Specification Business Process Model and Notation (BPMN). <http://www.omg.org/spec/BPMN/index.htm> (2011)
- [9] Scheer, A. Aris - Business process modeling. Berlin: Springer (2000)
- [10] Winter, R. Business engineering navigator (BEN), gestaltung und analyse von geschäftslosungen "business-to-it", universitat st. gallen, muller-friedberg-str. 8, st. gallen, 9000, switzerland Springer (2011).
- [11] The Open Group Archimate. <http://www.archimate.nl/en/home/> (2008)

# Formal Verification of Coordination Systems' Requirements - A Case Study on the European Train Control System\*

Huu Nghia Nguyen and Ana Cavalli

SAMOVAR UMR 5157 CNRS; Institute Télécom SudParis, Évry, France  
{huu-nghia.nguyen, ana.cavalli}@telecom-sudparis.eu

**Abstract**—Formal verification techniques of system requirements such as model-checking and theorem proving aims to show that the requirements satisfy some properties. Consequently, their success depends on the quality of the properties formulation. We propose an approach to verify requirements of coordination systems by generating automatically the properties to be verified from the requirements themselves. The requirement specifications of a system are provided at two different levels. The coordination specification gives a global overview of the system, in terms of the different roles participating to it, with their goals and needs and with their mutual dependencies and expectations. The process specification shows how a local participant of the system performs its activities. We exploit model checking techniques for verifying the process requirements against the properties generated by the coordination requirements. In addition to provide a theoretical framework, we show how to apply this methodology on the verification of the System Requirement Specification of the European Train Control System. It is complemented with a toolchain.

## I. INTRODUCTION

**Context & Motivation.** Requirements engineering is the first phase of system development process in the waterfall method. Some recent development methods assume that it continues through the lifetime of a system. Its activities vary widely, depending on the type of system being developed. Basically, it refers to the process of specifying user requirements about future systems, and then to map them to specifications of the required system behaviors. System requirements verifications intend to identify problems early in this phase. Usually, System Requirement Specifications (SRSs) are described only in natural language and formal verification techniques are rarely applied. Specification errors and ambiguities in requirements have been widely recognized as one of the major source of problems in systems development. Hence, the cost of systems development will be reduced if these problems are early detected. The verification can also improve the reliability, maintainability of the resulting system.

Formal verifications such as model-checking and theorem proving have successfully been used to verify system requirements. They intend to decide whether a requirement specification will meet some expected properties. The input to these processes is a formal requirement specification and a set of properties. A model-based tool to check the consistency of system in Büchi automata against system properties in Linear Temporal Logic is presented in [1]. However the success of these methods depends on the quality of the properties constructed, *e.g.*, their coverage, their significance.

Nowadays, systems are more and more complex, *e.g.*, mainly because the size of the systems. It consists of many components which are usually implemented in distributed environments, and are coordinated together to achieve a common goal. The computation of such coordination systems is thus divided in each of its components.

The requirement specifications of coordination systems can be captured from two levels: process (or local) and coordination (or global). The process requirements help developers to build a single computational component: a single-threaded, step-at-a-time computation. While the coordination requirements are the glue that binds separate components into an ensemble. Coordination requirements have to deal with the problem of managing interactions among participants, which are concurrent (and distributed) processes, hence some drawbacks can occur: deadlocks, starvation or incorrect multiple access to resources, ... So that, the coordination is the consistent organization of the communication and its effects, such that required cooperation between all components involved is established. It complicates the system design since the designer cannot only focus on activities of each process, but has to consider at the same time the compliance of these processes *wrt.* the coordination requirements.

There exist several works trying to apply formal approaches to verify the SRS of the European Train Control System (ETCS). Logic-base approaches were investigated by [2], [3]. The authors consider the ETCS as a parametric hybrid system. The system will be proven that it satisfies all relevant safety constraints, *e.g.*, collision-free of trains in [2], or the handover processes of Radio Block Center (RBC) in [3]. The duration calculus was used by [4] to model and analyze the treatment times of emergency messages. A toolchain dedicated for modeling and testing the software controller of ETCS Onboard Unit (OBU), is presented in [5]. It models the functional features of OBU, then generates the executable code for simulator, which will be used to do the V&V and the test tasks.

Our work is complementary of these works, it is based on model-checking that is cheaper than theorem proving [6]. In this approach, the properties to be checked are automatically generated from coordination requirements of systems. Furthermore, we do the V&V at model levels, *i.e.*, no need of interaction with real software implementation, hence it is possible to verify also the requirements for the non-software parts of the ETCS.

**Contributions.** We propose an approach, complemented with a toolchain, to verify formal requirement specifications of coordination systems. We then apply it to verify the SRS of the ETCS. Particularly, we introduce a simple algebra language to formulate requirements at both coordination and process aspects.

---

This work was funded by the "Direction Générale de la compétitivité, de l'industrie et des services" (DG CIS) (Grant No. 112930309) in the context of the ITEA2 project openETCS

We then verify the consistency among process requirements, such as deadlock-free, and the compliance between process requirements against coordination requirements. We exploit the IF model-checker. The process requirements are encoded into Intermediate Format (IF) specifications while the expected properties represented by IF observers are generated from coordination requirements. The IF model-checker will show whether the observers observe their expected behaviors from IF specifications. Once the check is passed, we will generate a set of test suites that can be used later by a global tester to test a real implementation of the system.

**Outline.** The rest of the paper is organized as follows. We propose in Section 2 our simple algebra language to formulate requirements for process and coordination features. Section 3 describes the formal verification based on the IF model-checker by encoding process requirement specifications into IF specifications and coordination ones into IF observers. Test generation is presented in Section 4. Section 5 presents a case study in which we have applied our approach to verify the System Requirement Specification of the ETCS. Finally, Section 6 draws conclusions and outlines future work.

## II. REQUIREMENTS MODELING

**General Syntax.** Process algebras have successfully been used to give a concise, readable and formal specification to SRS [7]. It helps to easily and quickly formulate and to verify the SRS [8]. We introduce a simple process algebra language that was used in our framework [9], [10] to verified the bisimulation conformance relations. Thus this paper will enrich the framework by appending the model-checking techniques and the generation of test cases. The language can be used to specify coordination systems with both coordination and process requirement aspects.

The basis of a behavioural description is the notion of activity. To promote generality, let us first suppose a set  $A$  (alphabet) of activities of interest which will be detailed later. The syntax of our specification language,  $L(A)$ , is given by:

$$L ::= \text{skip} \mid \alpha \mid x := e \mid L; L \mid L + L \mid L|L \mid [\phi] \triangleright L \mid [\phi] * L$$

In  $L(A)$  we can specify activities which are either simple or composite. A basic activity is either a do-nothing (`skip`), a basic activity ( $\alpha$ , with  $\alpha \in A$ ), or an assignment, e.g.,  $x := e$  represents that the data value of variable  $x$  is assigned by the one of expression  $e$ . We then have structuring operators, that can be used to specify composite activities: sequencing, e.g.,  $L_1; L_2$  the activities in  $L_1$  are executed before the ones in  $L_2$ ; non deterministic choice, e.g.,  $L_1 + L_2$  presents that either  $L_1$  or  $L_2$  is executed; and parallel activities, e.g.,  $L_1|L_2$  presents that  $L_1$  and  $L_2$  are interleaved. Furthermore, we have data-based conditional constructs, namely guards ( $\triangleright$ ) and while loops ( $*$ ), where  $\phi$  is a condition (a boolean expression).

**Basic Activities.** A basic activity captures a communication between two participants. At coordination levels, it represents an interaction, while at process levels, it represents a reception or a sending of a participant.

Let  $a, b, c, \dots \in \mathcal{R}$  be a finite set of roles,  $o, o_1, \dots \in \mathcal{O}$  be a finite set of operations, and  $x, y, z, x_1 \dots \in \mathcal{V}$  be a finite set of variables, a ground *interaction*  $o$  from role  $a$  to  $b$ , with  $a \neq b$ , is denoted by  $o^{[a,b]}$ , e.g.,  $\text{ACK}^{[a,b]}$ . A sending realized by

the participant  $a$  to  $b$  is denoted by  $o^{[a,b]}$ ! while  $o^{[a,b]}$ ? denotes a reception  $o$ , from  $a$ , of the participant  $b$ .

The interaction may carry data that are represented by a free variable,  $x$ , or a bound variable,  $\langle x \rangle$ , thus we have free interactions,  $o^{[a,b]}$ . $x$ , or bound interactions,  $o^{[a,b]}$ . $\langle x \rangle$  respectively. At local view, we introduce a (bound) reception  $o^{[a,b]}$ ? $\langle x \rangle$ , a free sending  $o^{[a,b]}$ ! $x$ , and a bound sending  $o^{[a,b]}$ ! $\langle x \rangle$ . They are used to represent reception or sending activities of each participant in coordination systems. Table I summaries these basic activities.

TABLE I. BASIC ACTIVITIES

$\alpha$	Name	Level
$o^{[a,b]}$ . $x$	Free Interaction	coordination description
$o^{[a,b]}$ . $\langle x \rangle$	Bound Interaction	
$o^{[a,b]}$ ? $\langle x \rangle$	(Bound) Reception	process description
$o^{[a,b]}$ ! $x$	Free Sending	
$o^{[a,b]}$ ! $\langle x \rangle$	Bound Sending	

The difference between free and bound activities relies on how their variables are interpreted. In the free interaction, the value of the variable  $x$  must be fixed before the occurrence of interaction  $o$ . While in the bound interaction, a new value will be bounded into the variable  $x$ , hence overwriting the current value of  $x$ , when the interaction occurs.

**Coordination Specification.** A coordination specification describes, with a global perspective, the legal interactions among roles played by the participants of a coordination system. Each role is identified by a unique name. The basis of an interaction-model choreography description is the interaction. A *coordination specification* is an element of  $L(A)$  with  $A = \{o^{[a,b]}$ . $x, o^{[a,b]}$ . $\langle x \rangle\}$ .

**Process Description.** A process description describes what is expected from a participant of the coordination system that would implement it. Furthermore, it represents the behavioural contracts or interfaces that participants advertise in order to foster reuse, composition, and adaptation. In a participant  $a$ , the primitives can be abstracted as sending and reception activities. Formally, a *local (or process) description* for an entity  $a$  is an element of  $L(A)$  with  $A = \{o^{[a,b]}$ ! $x, o^{[a,b]}$ ! $\langle x \rangle, o^{[b,a]}$ ? $\langle x \rangle\}$ .

## III. FORMAL VERIFICATION

The formal verification of requirements is done to ensure (a) the consistency among process requirements, e.g., deadlock-free, and (b) the compliance between a coordination requirement and a set of process requirements. We exploit the existing IF toolset [11] to realized the verification by encoding our formal descriptions into IF processes.

**IF toolset.** The IF toolset provides an environment for verification of IF specifications. IF is an intermediate representation dedicated for timed asynchronous communicating systems. It represent a system by a set of parallel processes communicating asynchronously through a set of buffers. An IF process is described as a timed automaton extended with data. It has an unique identifier, a set of control states, a private queue of pending messages (received and not yet consumed), and local data such as discrete variables and clocks. A transition between two states can be triggered either by (timed) guards or by an input message. The effect actions of a transition may include

$$\begin{array}{c}
\begin{array}{c}
\text{(ASSIGN)} \\
\frac{}{x := e \xrightarrow{/x := e} \text{skip}}
\end{array}
\qquad
\begin{array}{c}
\text{(RECEIVE)} \\
\frac{}{o^{[a,b]}? \langle x \rangle \xrightarrow{o^{[a,b]}? \langle x \rangle} \text{skip}}
\end{array}
\qquad
\begin{array}{c}
\text{(SEND}_1 \text{ - free sending)} \\
\frac{}{o^{[a,b]}!x \xrightarrow{/o^{[a,b]}!x} \text{skip}}
\end{array}
\qquad
\begin{array}{c}
\text{(SEND}_2 \text{ - bound sending)} \\
\frac{}{o^{[a,b]}! \langle x \rangle \xrightarrow{/new(x); o^{[a,b]}!x} \text{skip}}
\end{array}
\\
\begin{array}{c}
\text{(SKIP)} \\
\frac{}{\text{skip} \xrightarrow{\checkmark} \epsilon}
\end{array}
\qquad
\begin{array}{c}
\text{(SEQ}_1 \text{ - } L_1 \text{ does not end)} \\
\frac{L_1 \xrightarrow{[\phi] \alpha} L'_1}{L_1; L_2 \xrightarrow{[\phi] \alpha} L'_1; L_2}
\end{array}
\qquad
\begin{array}{c}
\text{(SEQ}_2 \text{ - } L_1 \text{ ends, begin } L_2) \\
\frac{L_1 \xrightarrow{[\phi_1] \checkmark} L'_1, L_2 \xrightarrow{[\phi_2] \hat{\alpha}} L'_2}{L_1; L_2 \xrightarrow{[\phi_1 \wedge \phi_2] \hat{\alpha}} L'_2}
\end{array}
\qquad
\begin{array}{c}
\text{(CHOICE}_1 \text{ - choose } L_1) \\
\frac{L_1 \xrightarrow{[\phi] \hat{\alpha}} L'_1}{L_1 + L_2 \xrightarrow{[\phi] \hat{\alpha}} L'_1}
\end{array}
\qquad
\begin{array}{c}
\text{(PAR}_1 \text{ - one step in } L_1) \\
\frac{L_1 \xrightarrow{[\phi] \alpha} L'_1}{L_1 | L_2 \xrightarrow{[\phi] \alpha} L'_1 | L_2}
\end{array}
\\
\begin{array}{c}
\text{(PAR}_3 \text{ - synchronous termination)} \\
\frac{L_1 \xrightarrow{[\phi_1] \checkmark} L'_1, L_2 \xrightarrow{[\phi_2] \checkmark} L'_2}{L_1 | L_2 \xrightarrow{[\phi_1 \wedge \phi_2] \checkmark} L'_1 | L'_2}
\end{array}
\qquad
\begin{array}{c}
\text{(GUARD)} \\
\frac{L \xrightarrow{[\phi'] \hat{\alpha}} L'}{[\phi] \triangleright L \xrightarrow{[\phi \wedge \phi'] \hat{\alpha}} L'}
\end{array}
\qquad
\begin{array}{c}
\text{(LOOP}_1 \text{ - one more iteration)} \\
\frac{L \xrightarrow{[\phi'] \hat{\alpha}} L'}{[\phi] * L \xrightarrow{[\phi \wedge \phi'] \hat{\alpha}} L'; [\phi] * L}
\end{array}
\qquad
\begin{array}{c}
\text{(LOOP}_2 \text{ - end of the loop)} \\
\frac{}{[\phi] * L \xrightarrow{[\neg \phi] \checkmark} \epsilon}
\end{array}
\end{array}$$

Fig. 1. Encoding from Local Description to IF Specification

sending output messages, setting/resetting clocks, assignment of variable values, and creation/destruction of processes.

**Encoding Process Requirements into IF Processes.** The encoding of local specifications into IF processes is presented in Figure 1. We denote  $\hat{\alpha}$  either  $\alpha$  or  $\checkmark$ . The  $\epsilon$  is used to denote the termination. The symmetric rules for  $\text{PAR}_1$  and  $\text{COM}_1$  can be inferred from them and are omitted here. In this figure, we denote a transition in IF process as  $s \xrightarrow{[\phi] \alpha / Act} s'$ , that is, when the trigger  $\alpha$  occurs and the guard  $\phi$  is validated then the effect actions in  $Act$  are done and the system will pass from state  $s$  to  $s'$ . The guard  $\phi$  is omitted when it is *true*. Let us go into detail, the encoding of the bound sending activity  $o^{[a,b]}! \langle x \rangle$  is presented by the rule  $\text{SEND}_2$ . We create an IF procedure, named `new`, to bound a new value to  $x$ . The new value will be then used by the free sending  $o^{[a,b]}!x$ .

**Observer Generation.** In the IF toolset, we can check whether an IF system holds some specific properties. The property is encoded in a special IF process - the so-call *observer* that runs in parallel with the IF system and checks if it can observe the expected behaviors such as input and output messages. In IF observer transition, a trigger is a match clause, *i.e.*, the transition is fired if there is an activity, either an input or an output, which matches the clause.

We create two transitions to observe an interaction  $o^{[a,b]}$ . The first transition is to match a sending in  $a$ , while the second one is to match a reception in  $b$ . The rules  $\text{INTER}_1$  and  $\text{INTER}_2$  in Figure 2 represent the observation of a free interaction and a bound interaction respectively. In the rule  $\text{INTER}_1$ , a guard  $[y = x]$  is created in the second transition since the free interaction transfers a pre-fixed value of a variable, then we need to verify if this value is the one observed in the first transition.

#### IV. TEST GENERATION

Once the coordination requirements are validated against process requirements, they can be used to generate test cases

$$\begin{array}{c}
\text{(INTER}_1 \text{ - free interaction)} \\
\frac{}{o^{[a,b]}!x \xrightarrow{o^{[a,b]}!y} o^{[a,b]}? \langle x \rangle \xrightarrow{[y=x] o^{[a,b]}? \langle y_1 \rangle} \text{skip}}
\end{array}
\\
\begin{array}{c}
\text{(INTER}_2 \text{ - bound interaction)} \\
\frac{}{o^{[a,b]}! \langle x \rangle \xrightarrow{o^{[a,b]}!y} o^{[a,b]}? \langle x \rangle \xrightarrow{o^{[a,b]}? \langle y_1 \rangle} \text{skip}}
\end{array}$$

Fig. 2. Generation of IF Observer

to test real implementations of components to ensure that they respect these requirements.

**TestGen-IF.** The tool [12] was developed to generate test cases from an IF specification and a set of test purposes. A test purpose is specified in TestGen-IF as a list of conditions. A condition is a conjunction of constraints about process instance identifiers, states, actions, and variable values. The tool can avoid the state space explosion or deadlock problems encountered in exhaustive or exclusively random searches. It constructs a partial graph of the IF specification by using a breath first search within a given depth. It then finds on the constructed graph a neighborhood from the current state (that is the initial state at the first time). If it reaches a state where one or more test purpose are satisfied, the result test sequence is updated and it starts to construct a new partial graph from this state. Otherwise, it constructs a new partial graph from a random leaf of the current partial graph. The researches stop when all the test purposes are satisfied or when there is no more transition to explore.

**From Coordination Requirements to Test Cases.** We presented, in Section 3, a generation of an IF observer from a coordination requirement. We also apply these rules, presented in Figure 1, to generate automatically an IF system corresponding to a coordination requirement. Particularly, we create a (process) requirement description for each participant from the coordination requirement. The requirement of a participant  $c$  in the coordination will be created by replacing a free interaction  $o^{[a,b]}!x$  (resp. a bound interaction  $o^{[a,b]}! \langle x \rangle$ ) in the coordination requirement by a free sending  $o^{[a,b]}!x$  (resp. a bound sending  $o^{[a,b]}! \langle x \rangle$ ) if  $c = a$ , or by a reception  $o^{[a,b]}? \langle x \rangle$  if  $c = b$ , or by a `skip` otherwise. After that, the rules in Figure 1 will be applied to generate an IF process. Based on this process and a given test purpose, TestGen-IF will generate a set of test sequences which can be used by a global tester to test a real implementation of the system.

#### V. CASE STUDY

**The European Train Control System.** We validate our approach to verify the System Requirement Specification of the ETCS. Over the past century, Europe's railways have been developed within national boundaries, resulting in a variety of different signaling and train control systems, which hampers cross-border traffic. The ETCS, part of ERTMS – the European Railway Traffic Management System, is a signalling, control and train protection system designed to replace the incompatible safety systems currently used by European railways.

The project openETCS, with the participation of 32 European partners from industry and research institutions, aims to develop an integrated modeling development, validation and testing framework for leveraging the cost-efficient and reliable implementation of ETCS.

**Local Requirements.** Basically, the ETCS at the final level<sup>1</sup> can be considered as a coordination system consisting of three components running in parallel in a distributed scenario. These components are equipped at trackside, *e.g.*, RBC and EuroBalise, and at trainside, *e.g.*, OBU. They communicate each others by message exchanges, *e.g.*, via Global System for Mobile Communications - Railway (GSM-R).

In ETCS, a train moves in a virtual *moving blocks* defined by RBC via Movement Authorities (MAs) that define the permission to run by giving a location (called End of Authority - EoA) to which the train is authorized to move. A MA contains also a limit speed at the EoA position the train must respect. The MAs will be updated periodically. Particularly, the Eurobalise provides to the OBU the current reference position of the train. The OBU sends information about its position to RBC and adjusts the current speed and acceleration of the train based on MAs received from RBC. Based on received information, the RBC determines MAs of the train and sends it to the OBU.

**Global Requirement.** Coordination requirements can be identified from the SRS of the ETCS that the three components above must satisfy. They guarantee the coordination between these components. We present a simple coordination requirement<sup>2</sup> that ensures the train always respect the speed given by OBU as the following, in which for the sake of simplicity we denote OBU, RBC and Eurobalise by  $o$ ,  $r$  and  $e$  respectively:

$$\text{EoA} := MA^{[r,o]}.(x) ; \text{Position}^{[e,o]}.(y) ; \text{Report}^{[o,r]}.(z) ; [x.\text{position} \leq y \wedge z.\text{speed} \leq x.\text{speed}] \triangleright \text{skip}$$

This requires a sequence of three interactions. First, a Movement Authority  $x$  is sent from RBC to OBU. Then the current position of the train  $y$  is given from Eurobalise to OBU. Finally OBU reports its information to RBC. The coordination finishes properly if the predicate  $[x.\text{position} \leq y \wedge z.\text{speed} \leq x.\text{speed}]$  holds. The predicate states the case where the train has passed the EoA and its current speed is lower than the one given in MA.

**Experimental Results.** The IF specifications of the components are automatically generated from its formal ones by our tool. Table II shows some metrics of the IF specifications. We have applied IF model-checker to verify the IF specifications against the observer. The verification used breadth-first search strategy. We obtain `pass` verdict after visiting 25308 states in 47 seconds on a 1.7 GHz Intel Core i5 system with 4 GB of memory.

TABLE II. IF SPECIFICATION SIZES

	#States	#Transitions	#Signals
OBU	39	107	9
RBC	7	13	4
Eurobalise	4	5	3

Based on the coordination requirement, test cases are generated by the tool TestGen-IF. The following shows a test

case that is generated from the coordination requirement for OBU using breadth-first search strategy.

```
?MA{100,120} ?Position{15} !Report{15,200}
```

This is a sequence of inputs and outputs. It represents the case in which the OBU received a MA with EoA is 100 and its limit speed is 120, and a Position with value is 15, then it must output a Report with the current position is 15 and the current speed is 200. The test cases will be used for testing the ETCS simulators implemented by other partners of the project.

## VI. CONCLUSION AND FUTURE WORK

We presented a framework to formally model and verify requirements of coordination systems. The requirements are specified from two levels of abstraction: coordination level that giving a global overview of system, and process level that focuses on each individual participant in the system. We then proposed a formal verification of compliance between these requirements. When a coordination requirement is satisfied, it can be used as test purposes to generate test cases which will test the real implementations of participants. We presented also a case study to model and verify the SRS of the ETCS.

As future work, we plan to (a) improve the technique used for test cases generation to deal with controllability and observability issues; (b) connect our formal language with the exist graphical modeling languages such as UML/SysML.

## REFERENCES

- [1] Salamah Salamah and M. Engskow, "Consistency Checks of System Properties Using LTL and Buchi Automata," in *Proc. of SEKE*, pp. 39–44, 2012.
- [2] A. Platzer and J.-D. Quesel, "European train control system: A case study in formal verification," in *Proc. of ICFEM*, pp. 246–265, 2009.
- [3] Y. Liu, T. Tang, J. Liu, L. Zhao, and T. Xu, "Formal Modeling and Verification of RBC Handover of ETCS Using Differential Dynamic Logic," in *Proc. of ISADS*, pp. 67–72, Ieee, Mar. 2011.
- [4] R. Meyer, J. Faber, J. Hoenicke, and A. Rybalchenko, "Model checking Duration Calculus: a practical approach," *Formal Aspects of Computing*, vol. 20, pp. 481–505, May 2008.
- [5] J. Feuser and J. Peleska, "Model Based Development and Tests for openETCS Applications – A Comprehensive Tool Chain," in *Proc. of FORMS/FORMAT*, pp. 1–9, 2012.
- [6] I. Timm and H. Fürstenau, "From Testing to Theorem Proving," in *Multiaagent Engineering*, 2006.
- [7] M. Bidoit, H.-J. Kreowski, P. Lescanne, F. Orejas, and D. Sannella, eds., *Algebraic system specification and development: A survey and annotated bibliography*. Springer Berlin Heidelberg, 1991.
- [8] H. Zhan, G. Yin, C. Sun, L. Shen, and J. Ni, "Process Algebra-Based Description for Software Requirement," Oct. 2008.
- [9] H. N. Nguyen, P. Poizat, and F. Zaïdi, "A Symbolic Framework for the Conformance Checking of Value-Passing Choreographies," in *Proc. of ICSSOC*, pp. 525–532, 2012.
- [10] H. N. Nguyen, P. Poizat, and F. Zaïdi, "Automatic Skeleton Generation for Data-Aware Service Choreographies," in *Proc. of ISSRE*, pp. 320–329, 2013.
- [11] M. Bozga, S. Graf, I. Ober, and J. Sifakis, "The IF toolset," in *Formal Methods for the Design of Real-Time Systems*, pp. 237–267, 2004.
- [12] A. Cavalli, E. M. D. Oca, W. Mallouli, and M. Lallali, "Two Complementary Tools for the Formal Testing of Distributed Systems with Time Constraints," in *Proc. of DS-RT*, pp. 315–318, 2008.

<sup>1</sup>Subset-026-2 of ERTMS/ETCS SRS, version 3, 2012

<sup>2</sup> Section 3.8.1.1 of the Subset-026

# Evaluating the Use of Model-Based Requirement Verification Method: An Empirical Study

Munmun Gupta, Daniel Aceituna, Gursimran S. Walia, Hyunsook Do  
North Dakota State University, Fargo, ND, USA  
{munmun.gupta, daniel.aceituna, gursimran.walia, hyunsook.do}@ndsu.edu

**Abstract**—Requirements engineering is a critical phase in software development that describes the customer needs and the specifications for the software solution. Requirements are gathered through various sources and the output is a list of requirements for a software product to be developed, written in Natural Language (NL). NL requirements are fault prone because stakeholders can interpret NL differently due to the inherent imprecision, ambiguity, and vagueness of NL. To address these problems, a model-based requirements verification method called NL to state transition diagram (STD) is proposed. This paper evaluates the ability of the NLtoSTD method to detect faults when used on NL requirements and to improve the software reliability. Overall, the result shows that the NLtoSTD is an effective requirements verification method.

## I. INTRODUCTION

To ensure software reliability, it is important to detect and prevent different types of faults during the development of various software artifacts. Requirements are gathered from different stakeholders (technical and non-technical) and recorded in natural language (NL), that describes the customer needs and the specifications for the software solution. The output of this phase, a *software requirement specification* (SRS - a means of communication among stakeholders), is especially fault-prone due to the inherent imprecision, ambiguity, and vagueness of NL. Requirement faults if undetected propagate to the later phases where they are difficult to find and fix [1-3].

To ensure high-quality requirements, numerous fault-based verification approaches have been developed and validated for fault-detection effectiveness [3, 8-11]. In particular, *software inspection*, have been empirically validated [3, 9] for early detection of faults in software artifacts. However, it is estimated that the software development effort is still spent on fixing problems that should have fixed early in the lifecycle [1, 2]. This rework stems from the fact that inspectors can have different interpretations of the requirements and may not notice the ambiguities and inconsistencies among other problems.

Model based approach, if applied to NL requirements can be used for verification of NL specifications [6, 10]. However, building a model from NL requirements is highly subjective. Consequently, an erroneous translation of NL requirements can result in the wrong model due to the inherent incompleteness and ambiguities of NL [11] and, thus, can eventually produce software that stakeholders do not want. To address this, several researchers have proposed modeling techniques using an automated NL translation approach [4,6,10,12]. These methods include approaches based on translating goals to state machines [4], and scenarios to state machines [12]. Automation can certainly improve the translation process, but complete and

error-free automation of this process is not possible because, often, NL requirements can be interpreted in multiple ways..

To address this problem, we propose a method that translates NL requirements into a State Transition Diagram (STD) in an incremental manner (NLtoSTD) and expose ambiguities, incompleteness, and inconsistencies in NL requirements. The NLtoSTD is carried out in two steps, where the first step turns each NL requirement into a STD building block (*NLtoSTD-BB*) and the second step then construct the STD using the STD-BBs (*STD-BBtoSTD*). Requirements engineers and stakeholders can detect faults during each step (NLtoSTD-BB and STD-BBtoSTD) and direct mapping from NL to model is preserved in the translation process. Each NL requirement becomes a segment of the STD so that adjustments made to the model can be directly made to the requirements, and visa-versa. The results from the previous study [5] validated the NLtoSTD-BB method and helped us make revisions. This paper presents an empirical study that evaluates the fault-detection ability of the revised NLtoSTD-BB method, and extends the research by evaluating the fault-detection ability of the STD-BBtoSTD method (used for the first time). Therefore, the complete NLtoSTD method (i.e., *NLtoSTD-BB + STD-BBtoSTD*) is evaluated in this paper.

## II. BACKGROUND

This section describes the basic concepts of the NLtoSTD method, the revised NLtoSTD-BB, and STD-BBtoSTD step.

### A. *NLtoSTD*: Basic Concepts

The basic idea of our NLtoSTD method is to translate the NL requirements into an STD, so that the ambiguity, incompleteness, and inconsistencies or any problem) in the NL requirements can be easily detected. The NL to STD translation first translates NL requirements into STD-BBs (*NLtoSTD-BB*) and then creates an STD using the STD-BBs (*STD-BBtoSTD*). A high-level overview of the NLtoSTD method is shown in Fig. 1. Fig.1 highlights the “*NLtoSTD-BB*” translation (Step1) and the “*STD-BBtoSTD*” construction (Step2). We hypothesize that the NLtoSTD helps discover the problems in the NL requirements by examining the individual STD-BBs and the resulting STD.

As shown in middle part of Fig. 1, the three elements that make up a STD-BB (i.e., *current state* (*Sc*), *next state* (*Sn*), and *transition* (*T*)) are precisely extracted from an individual requirement. The selection of these elements was based upon the characteristics of an ideal requirement and an inspection scheme that can help detect the problems that are otherwise left undetected using traditional inspection methods. As illustrated

This work was supported in part by NSF CAREER Award CCF-1149389 to North Dakota State University.

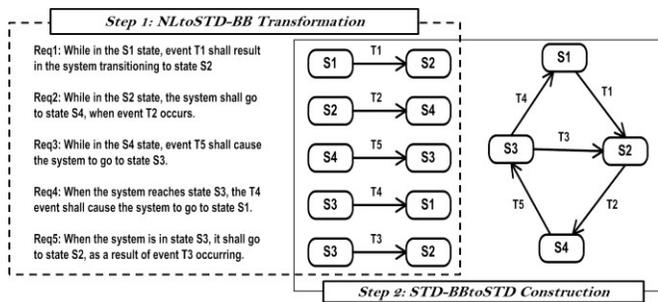


Fig. 1. NLtoSTD Method Overview

in Fig. 1, each requirement is stated so that it directly maps to an STD-BB. Each requirement explicitly states its precondition in the form of the current state ( $Sc$ ) and its post condition in the form of the next state ( $Sn$ ). However, typical NL requirements do not explicitly state current and next states, thus a requirement's preconditions and post conditions are often inferred causing ambiguities and incompleteness. Similarly, the absence of the explicit transition ( $T$ ) can cause difference in the interpretations of a requirement by different stakeholders. The NLtoSTD method requires the stakeholders to identify the aforementioned three elements so that they can detect the requirement faults while building the STD and ensures that the requirements are as consistent and concise as possible.

### B. Step 1 - NLtoSTD-BB: Application for Fault Detection

The first step of the NLtoSTD method transforms each NL requirement into an individual STD-BB. The STD-BBs act as a formalized version of the NL requirements and can lead to the detection of faults for two reasons: (1) a formalized version of the NL requirements has only one specific interpretation, exposing *ambiguities* in the NL requirements, and, (2) a formalized version exposes *missing* requirements more readily, as compared to a fault-checklist inspection of requirements.

#### 1) Original NLtoSTD-BB [5]

As shown in Fig. 1, each NL requirement is translated into an STD-BB by extracting three elements  $\{Sc, T, Sn\}$ . The basis for this transformation is that a functional requirement should describe an entity transitioning from one state to another. For example, the requirement "While the car is moving forward, the driver shall be able to stop it by applying the brake." would describe the Car (an entity) transitioning ( $T$ ) from moving ( $Sc$ ) to stopping ( $Sn$ ), using an STD-BB.

In the above example requirement, the three elements are explicitly stated, yielding definable values for  $Sc$ ,  $T$ , and  $Sn$ . In practice, however, requirements often ambiguously imply one or more values for  $Sc$ ,  $T$ , and  $Sn$ , thus identifying a value for each element would not be obvious. For instance, the prior requirement may have stated: "The driver shall stop the car by applying the brakes." Note that  $Sc$  is not explicitly stated as "moving" but, rather, implied. In our original STD-BB, we used question marks (???) to denote an element that is not documented. Thus, in this example, we would define the three elements as  $\{Sc: ???, T: Applying Brake, Sn: Stop\}$ . It may be safe to assume that the car is moving prior to stopping, but it requires an assumption. Undocumented assumptions can be erroneous and can cause serious defects (especially when the developers lack appropriate knowledge of the application

domain). In this example, it is not clear whether we assume "moving forward," "moving backward," or both. It is important to document what may seem obvious, instead of allowing the possibility of an erroneous assumption. Therefore, the NLtoSTD-BB helps to expose undocumented assumptions.

We developed a set of three questions to help users systematically identify the three elements for each requirement during this step [5]. Asking these three questions identifies explicit or undocumented values for  $\{Sc, T, Sn\}$ , resulting in an STD-BB. While the ambiguities and incompleteness may not be obvious in the NL requirements, they are made obvious in an STD-BB that stakeholders can work towards its completion.

#### 2) Revised NLtoSTD-BB

The NLtoSTD-BB used during the Sudy 1 showed that the method was significantly more effective than the fault-checklist based inspection, when the subjects correctly extracted the STD-BBs. The variations in performance during study [5] prompted us to re-evaluate the way that the three elements ( $Sc, T, Sn$ ) were determined. This section dicuss the changes and the revised NLtoSTD-BB method.

Fig. 2 illustrates the revised NLtoSTD-BB method using an example requirement. In the revised NLtoSTD-BB method, the three changes are briefly discussed along with their reasoning.

The **first change** is that we explicitly added an entity to a state to represent  $Sc$  and  $Sn$  as follows: entity (state). Allowing for multiple entities would alleviate the problem encountered with requirements that are not atomic. Separating the concepts of an entity and its given states, also makes it easier to derive  $Sc$  and  $Sn$ , since the user could first decide which entity is being affected and then determine the entity states before and after the effect. Fig. 2 shows that the revised NLtoSTD-BB method, can help identify three entities: unit, battery, and user. The **second change** in the revised NLtoSTD-BB method is allowing users to make an assumption. As shown in Fig. 2, "unit (normalOp?)" has a question mark. This indicates that the user can assume that it is the intended state and label it for follow up. This was done to improve the method's ability to expose ambiguities. The **third change** is to allow users to add conditions when they describe the transition ( $T$ ). This alleviates the problem when a requirement seems to state more than one transition. The revised NLtoSTD-BB method with five elements (*entity, entity's current state, entity's next state, transition, condition for transition*) is evaluated in this paper.

#### C. Step 2 - STD-BBtoSTD: Application and Tool Support:

The final step in the NLtoSTD method is the construction of STD based on the STD-BBs. The idea is to be able to both simulate the behaviours described in the requirements and to

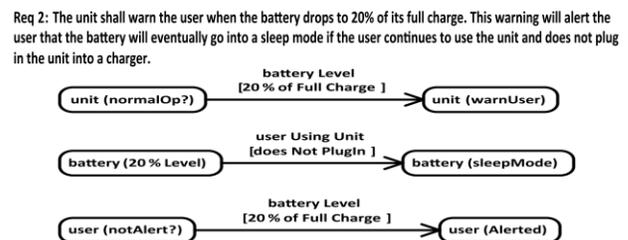


Fig.2. Revised NLtoSTD-BB

analyze these behaviours through the production of path traversals through the STD. This would potentially expose inconsistent and/or incorrect behaviours, that can be readily traced back to the requirements. An incomplete STD would expose incompleteness in the requirements verbage. Thus, the STD-BBtoSTD phase exposes potential faults by analyzing the STD's static and dynamic properties.

The STD's behavior was simulated by computer in order to expose faults that may not be evident unless the STD is enacted. There STD can be analyzed automatically by examining the path traversals, for desired behavior. The STD's construction was implemented through a software tool. The user enters the STD-BB data in an Excel spreadsheet, which is then read directly into the tool (by the tool's use of COM automation). The STD is then displayed in a separate window, and can be kept opened as more STD-BB data is being entered. The tool updates the STD, as changes are made to the STD-BB data. This allows the user to view the STD, make changes to the STD-BBs that would correct any STD structural faults, and see the results of those changes in real time. The subject can record faults during this step.

### III. EMPIRICAL STUDY

This study evaluated NLtoSTD-BB translation step at finding incompleteness and ambiguities during an inspection of NL requirements. This study also evaluated if additional new faults can be found during the construction of the STD. The complete NLtoSTD method was evaluated using a *repeated-measure design* in which different student teams (with varying number of members) developed requirement documents for different systems. Next, each participant individually inspects the requirement document (that was developed by them) using the NLtoSTD-BB step and kept a log of ambiguous, and missing requirements for respective documents. During the next step, the student individually worked to create STD from the STD-BBs (using an automated tool) and then analyzed the resulting STD to log new faults that were not found previously.

#### A. Research Questions and Hypotheses

The following research questions were investigated:

**RQ1:** Is the NLtoSTD-BB effective at detecting incomplete and ambiguous requirements during requirements inspection?

**RQ2:** Does creating the STD model result in detection of the faults in addition to those found during the NLtoSTD-BB?

**RQ3:** What are the problems faced by the subjects when using the complete NLtoSTD method?

#### B. Variables and Measures

Each subject performed an individual inspection of their requirements document using the NLtoSTD method. Our dependent variables include: *Effectiveness* - # of faults found and *Efficiency* - # of faults found per hour.

#### C. Participating Subjects and Requirement Artifacts

Sixteen computer science graduate students at North Dakota State University (NDSU) worked in teams to develop requirements document for different projects. Some students dropped the course resulting in this irregular size of student teams. There were two phases to this study. **First**, each team

TABLE I. ARTIFACTS DEVELOPED BY STUDENT TEAMS

Doc	Team #	No. of Subjects	System Description	Size (pages)
A	1	4	Parking lot availability system	25
B	2	4	Web portal for student residence	22
C	3	3	Virtual story board system	28
D	4	2	Matbus application for android	25
E	5	3	Professional development	32

developed a requirement document for a particular software system (Table I). **Second**, each subject inspected the requirement document developed by their team using the NLtoSTD-BB followed by STD-BBtoSTD method.

#### D. Study Procedure

The study details are provided in the following subsections.

1) *Phase I – Development of SRS documents:* The participants divided into 5 different teams of three or four participants developed the requirements documents for their identified software system. Details are provided in Table I.

2) *Phase II– Inspection using NLtoSTD:* During this step, the students in each team individually inspected their own SRS document using the NLtoSTD method.

a) *Training 1 -- NLtoSTD-BB:* The participants were first trained on how to map the NL requirements to STD-BBs. Next, the participants were instructed how to document the building block elements on a spreadsheet using few examples. Next, the participants were instructed how to record “ambiguities” and “incompleteness” or any other requirement faults that are found during the application of NLtoSTD-BB. The subjects were asked to translate few requirements into STD-BBs and document the faults using the same spreadsheet.

b) *Step 1-- Inspection using NLtoSTD-BB:* The participants used the information from Training 1 and individually inspected their own requirement document using the NLtoSTD-BB translation. This step resulted in a list of 16 individual spreadsheets that contained the STD-BB elements and the faults found (one per participant).

c) *Training 2 - tool support for STD Creation:* During this session, the participants learned about the STD tool. The subjects were instructed how to load the BBs (from step 1) into the tool and then, how to construct an STD from the BBs. The subjects were then instructed to examine the constructed STD. Finally, the participants learned to record the fault type in the fault spreadsheet. To ensure that subjects understood, the subjects practiced these steps through an example system.

d) *Step 2 – STD-BBtoSTD and inspecting STD:* The subjects constructed the STD diagram from STD-BB. The output of this step was 16 individual STD diagrams (one per participant). The resulting STD diagrams were analyzed for potential incompleteness, inconsistencies in the behaviours, or any other requirement faults. The participants analyzed and recorded the faults during and after the creation of STD. The students also documented the reason and classification of the fault (*incompleteness, ambiguous, inconsistency* or *other*) in the fault spreadsheet. This step resulted in 16 individual fault

lists. Finally, subjects provided feedback about the NLtoSTD-BB and the STD-BBtoSTD. An in-class discussion with subjects helped researchers better understand the results.

### E. Data Collection

The *quantitative* data included the *ambiguous*, *missing*, and *inconsistent* faults found by each subject in their SRS document during: a) translation of NL requirements to the STD-BBs, and b) construction of STD using the BBs. Each subject was provided 50 minutes during NLtoSTD-BB step and 30 minutes during STD-BBtoSTD step. The timing data was used for analyzing the efficiency values. The *qualitative* data included student's rating of NLtoSTD by answering a multi-question questionnaire based on a 5 point likert-scale. We also collected feedback post-study with participating subjects.

## IV. DATA ANALYSIS AND RESULTS

This section analyzed the data collected during NLtoSTD-BB, STD-BBtoSTD, post-study questionnaire and interviews.

### A. RQ 1: Effectiveness and Efficiency of NLtoSTD-BB

This section reports the *effectiveness* and *efficiency* of the NLtoSTD-BB during the requirements inspection. Before analyzing the fault data, the researchers determined the validity of the faults for each subject by reading through the fault spreadsheet reported by each participant to remove any false-positives (or if any faults were unclear). Next, the number of "Missing Functionality (MF)" and "Ambiguous Information (AI)" faults reported by each subject during the application of NLtoSTD-BB for their respective documents were counted.

Since each subject individually inspected their own document, the document (for each team) was inspected by all the subjects belonging to that team. Fig. 4 organizes results by the total number of AI and MF faults found by the member belonging to each team. Main observations from Fig. 4 follow:

- Fifteen out of sixteen subjects found faults (AI or MF) during the NLtoSTD-BB based inspection of their requirements document. The subjects (numbered 6) reported a lot of faults but none of them represented real problems;
- There were no consistent differences in the total number of AI vs. MF faults found by the subjects within each team. This was major improvement from the results in our previous study [5] where, the subjects using the NLtoSTD-BB method consistently found larger number of MF faults than the AI faults. This is a positive result that the improved heuristics were able to find both types of faults when constructing the STD-BBs.
- For each document, the average number of faults was calculated for each team by dividing the total number of unique faults by the number of subjects who inspected the document. The results showed that teams 1 through 5 found an average of 15, 12, 18, 9, and 22 faults respectively. This demonstrates an improvement in the performance of student teams from the first study [5] when using the original NLtoSTD-BB method (teams found an average of 7 faults at most) as well as a improvement when considering the inspection results in [5] when using the fault checklist method (an average of 5 faults).

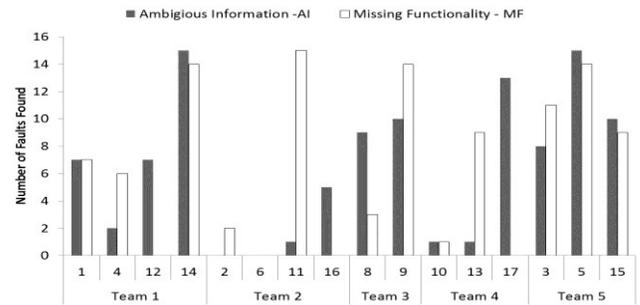


Fig. 4. Number of AI and MF Faults found by Subjects using NLtoSTD-BB

Therefore, based on these results, fault detection *effectiveness* of the NLtoSTD-BB method has improved from its first evaluation. Additionally, the distribution of faults is more consistent across both fault types (AI and MF). The revised NLtoSTD-BB heuristics were able to highlight hidden ambiguities in individual requirements which are otherwise not detected when performing a traditional inspection process.

Regarding the *efficiency* (faults/hour), the student teams found an average of 19, 7, 10, 10, and 26 faults per hour respectively. This is also an improvement in comparison to the results from the first study [5]. The high efficiency values reported in this study validate the ease of use of the revised NLtoSTD-BB method. Therefore, the NLtoSTD-BB is an effective and efficient method for verifying NL requirements.

### B. RQ 2: Fault Detection during the STD Creation

The translation of BBs into STD can highlight the ambiguities and incompleteness in the NL requirements by examining the gaps (or disconnections) and inconsistent path traversals in the STD, and to identify the *inconsistencies* in the requirements that are not a focus during the NLtoSTD-BB.

To investigate the validity of this step, we counted the number of new MF, AI and INC faults reported by each subject after the creation and analysis of STD for their respective documents. The result on the number of new AI, MF, and Inc faults found by each subject belonging to a team is shown in Fig. 5. Interestingly (Fig. 5), each subject found at least one new fault (AI or MF or Inc) after creating the STD. As expected, a larger number of "Inc" faults are found during this step as compared to the AI and MF faults. This is also consistent across all the teams. The subjects felt it was easy to observe the inconsistencies when looking at a complete STD as opposed to translating individual NL requirements (one at a time) to STD-BBs. Since, the loading of BBs to create the STD is an automated process (using a tool); it is not surprising that subjects were able to find additional faults by focusing their attention on examining the STD and recording faults.

To better understand the *effectiveness* of STD-BBtoSTD, we compared the percentage contribution of the STD-BBtoSTD relative to the overall NLtoSTD for each team. This was done by dividing the # of unique faults found during STD-BBtoSTD by sum of total # of unique faults found during NLtoSTD-BB and STD-BBtoSTD combined. The student teams 1-5, after the creation of STD, found 18%, 22%, 14%, 23%, and 12% of total faults respectively. To further verify the usefulness of the creation and analysis of STD, a one-sample t-

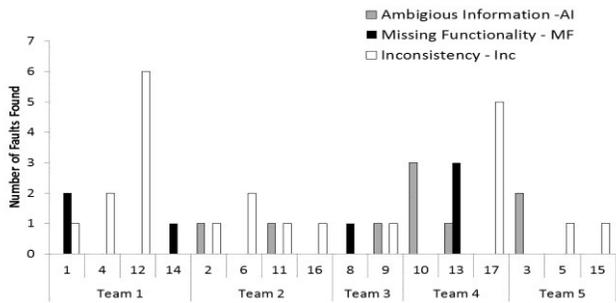


Fig. 5. Number of Faults found after the Creation of STD using the STD-BBs test was run separately for each team to determine whether the number of faults found during re-inspection using STD was significantly greater than zero (0). The result was found to be statistically significant ( $p < 0.001$ ) and indicate a benefit of creating STD using the BBs, for finding requirement faults.

### C. Difficulties Faced by Subjects Using the NLtoSTD Method

The qualitative data collected during this study evaluated the usability of the NLtoSTD-BB method. To do that, the participants were asked to rate the difficulty level for finding the “Entity”, “Initial Status”, “Changed Status”, “How is Status Changed”, and “Conditions for Change” for NL requirements.

Using a 5-point likert scale (1-very difficult to 5-very easy), the participants rated the difficulty level for each of the five elements of an STD-BB. A One-sample Wilcoxon Signed-Rank test determined whether the medians ratings were significantly greater than 3 (midpoint of the scale). The results showed that the NLtoSTD-BB method received positive ratings (i.e., median value greater than or equal to three), but not statistically significant. The subjects also rated the difficulty level during the construction of STD and analyzing the constructed STD for faults. The results showed that the STD creation received significantly positive ratings ( $p < 0.05$ ).

The complete NLtoSTD method was also evaluated using the feedback from subjects on the following seven attributes: *Simplicity*, *Ease of Understanding*, *Ease of Use*, *Intuitiveness*, *Comprehensiveness*, *Usefulness*, and *Ease of finding faults*. Each subject rated the attributes on a 5-point scale. The results from Wilcoxon Signed-Rank test revealed that the NLtoSTD received significantly positive ratings on *Ease of Understanding*, *Ease of Use*, and *Ease of finding faults*.

Overall, the subjects felt that the NLtoSTD process helped them understand the major problems in requirements, and that the effort spent during the NLtoSTD inspection process was worthwhile. The potential improvements regarding the tool and the guidance to help analyze the STD diagram will be implemented in future evaluations.

## V. DISCUSSION OF RESULTS

**RQ 1:** The NLtoSTD-BB method helped inspectors find inherent ambiguities and incompleteness in requirements. The comparison of the results against the previous research results [5] revealed that the subjects were able to find larger number of total faults (on average), and the distribution of faults across fault types (MF and AI) was more consistent. The results also showed that the NLtoSTD-BB method helped find the faults faster (i.e., efficiency) when compared to the results in [5].

**RQ 2:** Based on the results, additional MF and AI fault types are uncovered during the examination of STD constructed from the BBs. In particular, the creation of STD aids inspectors at detecting a large number of “Inc” that are otherwise not apparent when looking at individual requirements. The construction of STD is useful for overall inspection effectiveness using the complete NLtoSTD method.

**RQ 3:** The subjects provided insights in to the use of the NLtoSTD method and improvements that can help improve the performance in future studies. The subjects mentioned that the tool should guide the NLtoSTD-BBs translation and should at least highlight parts of STD that are completely disconnected. We plan to make this process as much automated as possible without losing the promise of inspections.

## VI. CONCLUSION AND FUTURE WORK

Based on these results, the NLtoSTD method is effective method to detect AI, MF, and Inconsistency fault types during an inspection of NL requirements. We also identified the areas of improvement that would benefit the performance of subjects using the method. Our future work would include more replications, with a classic control group design so that we can understand how many faults found during the Phase III are solely due to the STD creation and not just due to the re-inspection. We also wish to automate as much of the heuristics as possible, including the NLtoSTD building block portion. The STD analysis could be automated as well, using a reasoning engine written in a logic language such as Prolog, and this has already been achieved to a certain degree.

## REFERENCES

- [1] B. Boehm and V. Basili. Software fault reduction top 10 list. *IEEE Computer*, pages 135–137, January 2001.
- [2] B. Boehm, *Software Engineering Economics*, Prentice-Hall, 1981.
- [3] B. Brykczynski, A survey of software inspection checklists, *ACM SE Notes*, 24(1):82,1999.
- [4] C. Damas, B. Lambeau, P. Dupont, and A. Lamsweerde, Generating annotated behavior models from end-user scenarios,” *TSE*, 31(12):1056-1073, 2005.
- [5] D. Aceituna, H. Do, G. Walia, and S. Lee. Evaluating the use of model-based requirements verification method: A feasibility study. *EmpiRE*, 2011, pages 13-20, August 30, 2011.
- [6] D. Popescu, S. Rugaber, N. Medvidovic, and D. Berry, Reducing ambiguities in requirements specifications via automatically created object-oriented models, *Monterey Workshop on Computer Packaging*, pp. 103-124, 2007.
- [7] F. Chantree, B. Nuseibeh, A. de Roeck, and A. Willis. Identifying nocuous ambiguities in natural language requirements. *RE*, pp 59–68, 2006.
- [8] R. Chillarege, I. Bhandari, J. Chaar, M. Halliday, D. Moebus, B. Ray, and M. Wong, Orthogonal fault classification - A concept for in-process measurements. *TSE*, 18(11): 943-956, 1992.
- [9] S. Sakhivel. Survey of requirements verification techniques. *Journal of Information Technology*, pp. 68-79, 1991.
- [10] L. Kof, R. Gacitua, M. Rouncefield, and P. Sawyer, Ontology and model alignment as a means for requirements validation, *ICSC*, pp. 46-51, 2010.
- [11] D. Barry. Ambiguity in natural language requirements documents. *Lecture Notes in Computer Science, LNCS*, volume 5320, pages 1-7, 2008.
- [12] E. Letier, J. Kramer, J. Magee, and S. Uchitel, Monitoring and control in scenario-based requirements analysis. In *Proceedings of the 27<sup>th</sup> International Conference on Software Engineering*, pages 382–391, 2005.

# On the Requirements and Design Decisions of an In-House Component-Based SPL Automated Environment

Elder Rodrigues\*, Leonardo Passos<sup>†</sup>, Leopoldo Teixeira<sup>‡</sup>, Flávio Oliveira\*, Avelino Zorzo\*, Rodrigo da Silva Saad<sup>§</sup>

\*Pontifical Catholic University of Rio Grande do Sul  
Porto Alegre, RS, Brazil

{elder.rodrigues, flavio.oliveira, avelino.zorzo}@puccrs.br

<sup>†</sup>University of Waterloo Waterloo  
Canada  
lpassos@gsd.uwaterloo.ca

<sup>‡</sup>Universidade Federal de Pernambuco  
Recife, PE, Brazil  
lmt@cin.ufpe.br

<sup>§</sup>Dell Computers of Brazil Ltd.  
Porto Alegre, RS, Brazil, RS, Brazil  
rodrigo\_saad@dell.com

**Abstract**—Software product line adoption has many challenges in industrial settings. A particular challenge regards the use of *off-the-shelf* tools to support this process, since these tools usually do not fully address some company’s specific needs. To elicit concrete requirements and provide tool vendors and implementers with direct feedback, we avail from our experience in developing a software product line to derive testing tools for a laboratory of a global IT company (currently set as a pilot study). In this paper, we present such requirements and argue that existing tools fail to address all of them. In addition, we present our design decisions in creating an in-house solution meeting the specific needs of the partner company. We also highlight that these decisions help in building a body of knowledge that can be reused in different settings sharing similar requirements.

## I. INTRODUCTION

Software Product Line (SPL) has emerged as a promising technique to achieve systematic reuse and, at the same time, decrease development costs and time to market [13]. Although the use of SPL practices has significantly increased in recent years, with an extensive list of successful cases,<sup>1</sup> there are still many challenges when implementing SPLs in industrial settings [9].

One particular concern relates to the use of *off-the-shelf* tools (*e.g.*, pure::variants [2] and Gears [8]) supporting the SPL development life-cycle, since frequently these tools do not meet specific requirements of the companies aiming to adopt them. To overcome this, many companies set to develop in-house solutions to support their specific needs.

The SPL community, however, currently lacks evidence on the driving factors around custom-based solutions, which hinders tool vendors and implementers from having feedback from industrial clients outside their clientele. To mitigate this, we report our experience in implementing a SPL to derive testing tools for a laboratory of a global IT company. In particular, we describe the specific requirements of that company and argue that existing tools fail to support them. We also present the design decisions in creating a customized

solution supporting the target SPL. Our contribution are twofold: (i) we identify a set of requirements that, although specific to our research context, already point needs currently not addressed by existing tools, either commercial or open-source. Thus, we elicit practical scenarios that tool vendors and/or implementers may consider supporting; (ii) we report our design decisions in fulfilling these requirements for an in-house solution developed for our partner company. These decisions, in turn, may be reused or adapted to improve existing tools or when devising solutions targeting similar needs.

This paper is organized as follows. Section II presents some context relative to the company in place, from which Section III builds on. The latest one then enumerates the elicited requirements, which we address with specific design decisions, discussed in Section IV. Section V briefly presents our custom-made tool and its usage workflow. Section VI revisits related work, while Section VII concludes the paper and points out final remarks.

## II. CONTEXT

Our research group on Software Testing<sup>2</sup> works in cooperation with the Technology Development Lab (hereafter referred as TDL) of a global IT company, whose development and testing teams are located in different regions worldwide.

In the partner IT company, development is performed over different programming languages and IDEs (Integrated Development Environments), including Visual Studio (for Microsoft-based solutions), Eclipse (for Java-based implementations), Flash Builder, PHP and packaged applications (*e.g.*, Siebel and Oracle).

Testing teams use commercial and open source frameworks and tools to partially automate their testing activities. Frequently, however, due to the complexity of the testing in place, these teams create custom components to enable testing non-trivial applications. For example, testers may need to create web-service-based scripts to test back-end components that provide

<sup>1</sup><http://www.sei.cmu.edu/productlines/casestudies/>

<sup>2</sup>[www.cepes.puccrs.br](http://www.cepes.puccrs.br)

interfaces to the front-end of a given application, or components that simulate asynchronous messages for offline business processes. Currently, due to the distributed nature of the testing teams, custom components are rarely shared, thus leading to redundancy, little reuse, coding inconsistencies, higher development costs, etc.

To eliminate the effort of repeatedly creating custom infrastructure, we started a pilot study on SPL adoption with the TDL of our partner company. Upon success, we aim to replicate or apply it in different testing teams, if not all of them. In particular, we used a SPL called PLeTs [4] that aims to support the derivation of a particular testing infrastructure from a set of shared components (product configuration), which are then glued together (product derivation). In this SPL, derived products are testing tools that take behavioural models as input; these models denote specific test cases, thus leveraging testing teams to follow a model-based approach [18], a process that we describe elsewhere [4].

Prior to creating any tool to support product configuration and derivation in the target SPL, the TDL first considered the use of off-the-shelf solutions, provided that specific requirements were met. We describe this requirements list in Section III.

### III. REQUIREMENTS

This section enumerates the requirements (RQs) we collected from the TDL concerning the configuration (component selection) and derivation of products (gluing selected components), in addition to the evolution of the target SPL.

*RQ1) The adopted tool must not be bound to any IDE.* Although the TDL relies mostly on Microsoft-based solutions, they also use solutions from different distributors and vendors (e.g., Eclipse for Java). In that case, the adopted tool must not impose the use of any IDE, as that would require a specific platform expertise that could limit TDL's way of working, and also lead to training costs unrelated to product line adoption.

*RQ2) The adopted tool must support a graphical-based notation for designing feature models.* This requirement follows from the fact that features are an effective communication medium across different stakeholders and feature models (FMs) are a widespread mechanism to capture variability. Thus, the TDL requires the use of FMs, with the particular need of a graphical-based notation. According to the TDL, the use of a graphical-based FM notation is likely to facilitate communication with different stakeholders outside testing teams, including non-technical staff (e.g., project managers). Moreover, graphical FMs provide testers with a quick visualization of the existing features in the current snapshot of the testing infrastructure, and how each feature relates to others. The TDL explicitly states that relying on textual-based notations is not an option. They argue that textual languages impose linearity, as one element is only known when another one ends, hindering an immediate grasp of the underlying structure.

*RQ3) The adopted tool must support a graphical-based notation for designing structural architectural models.* Since testing teams can be geographically distributed, changes to the underlying test infrastructure must be documented at all times to facilitate communication and future maintenance. In this case,

in addition to keeping FMs, assets of shared components in the test infrastructure should also be documented in terms of structural architectural models (currently set to be component diagrams) that closely resemble their coding artifacts. These models capture fine-grained details that FMs alone would otherwise miss (e.g., a component port). The preference towards graphical notations is in tune with RQ2.

*RQ4) Structural architectural models must be kept in synchronization with the FM and code base (and vice-versa).* This requirement imposes a full round-trip between FMs and architectural models. As before, models should be presented and edited graphically.

*RQ5) The tool must be extensible to support different structural architectural models and FM notations.* The TDL states that structural architectural modeling should be centered around UML diagrams (see RQ3), but they point out the benefit of supporting other notations in the future (e.g., Domain Specific Languages). Likewise, one should also account for different FM notations, with extensions added as needed.

*RQ6) FMs should be derivable from structural architectural models.* In the TDL, most testers are familiar with standard UML models, but less so with FMs. To prevent initial mistakes and minimize the effort in extending the testing infrastructure, the tool must be able to derive a FM from the defined structural architecture, which in turn can be tuned accordingly.

*RQ7) Structural architectural models should be derivable from FMs.* As time progresses and FMs become more common among stakeholders, testers can start extending the testing infrastructure by first changing the FM, and then deriving the corresponding structural architectural model, which can be tuned accordingly (round-trip is already requested by RQ4).

*RQ8) For each product of the testing infrastructure, it should be possible to derive its corresponding structural architectural model.* Testing tools (products) are the result of selecting and combining components. For each product, it should be possible to generate its structural architectural model, which results from selecting specific elements from the architectural model of the whole SPL.

*RQ9) Traceability links among models and implementation assets/elements should require minimal human intervention/effort.* Traceability is an important concern for the TDL, as FMs and structural architectural models need to be mapped to implementation assets, and vice-versa. To prevent a high burden on manually keeping such links, any adopted solution must automate traceability as fully as possible.

*RQ10) When extending the existing infrastructure with new features (components), their implementation must adhere to specific interfaces. To decrease coding effort and avoid human mistakes while enforcing coding styles, an initial skeleton implementation should be automatically derived from specific code templates.* Currently, when testers evolve the testing infrastructure (e.g., when implementing the interface of a core capability of the testing infrastructure) they often copy and adapt an existing implementation or write a new one from scratch.

*RQ11) The adopted tool must allow the creation of new glue code generators, that should be pluggable into the system without intrusiveness changes.* The adopted tool must allow hooking code generators to produce glue code for specific target languages. This is aligned with the need to integrate with non-Microsoft solutions.

*Summary:* by evaluating existing tools for SPL adoption, we found that no existing off-the-shelf tool (either commercial or open-source) meets all of the presented requirements. Therefore, we and the TDL set to create an in-house component-based tool to support the target SPL. In Section IV, we describe our design decisions in building such a tool.

#### IV. DESIGN DECISIONS

In this section, we report our design decisions (DDs) in creating an in-house tool supporting the requirements previously discussed. For each design decision, we refer to the associated requirements.

*DD1) Build a plugin-based tool that is extensible (RQ5, RQ11).* To allow extensibility, we employ a modular solution with a plugin-based architecture. The tool is currently implemented in C#, as the partner company relies mostly on Microsoft-based technologies. The use of C# facilitates integration and long term maintenance, as different employers can potentially enhance the tool in the future, either by developing new plugins, or improving its core.

*DD2) No support for code editing (RQ1).* This makes the tool IDE-independent, and as such, individual developers or entire testing teams can continue to use their preferred coding editors and/or development environments.

*DD3) Provide an extensible environment for using different feature models notations, with the use of FODA-based notation [7] as the default one (RQ2, RQ5).* Currently, the tool ships with a feature model editor plugin, that in addition to FODA's syntax, also supports definition of abstract features [17].

*DD4) Support an extensible environment for using different structural models. In particular, we currently support UML component diagrams as the default structural architectural model (RQ3, RQ5).* In the partner company, component diagrams are the most common artifact for documenting the structure of any given software architecture (at least in the TDL). Due to its widespread use within the company and from the fact that UML is taught in universities in the country where the TDL is located, its use dispenses extensive training, and thus reduces both costs and time. It is worth noting that the component diagram plugin targets a particular UML profile—SMarty [12], due to its support for variability encoding in UML diagrams.

*DD5) Every concrete feature is mapped to exactly one component. In case of feature interactions, we rely on #ifdef-based annotations (RQ4, RQ9).* Such a simple design dispenses the maintenance of explicit traceability links, thus eliminating the associated burden. In our pilot study, we observed that most features (components) are coarse-grained, making an 1:1 mapping a suitable solution. Since, #ifdef annotations are rare,

the TDL is unlikely to face situations where maintenance is hindered by #ifdef complexity (known as #ifdef hell [15]).

Binding between components and features is performed by exact name matching, and the same occurs for #ifdef macros (one component matches exactly one macro name that cannot be redefined).

*DD6) We synchronize the feature and structural models to maintain consistency after performing changes in each of them (RQ4, RQ6, RQ7, RQ9).* Consistency, in this case, is eased by relying on a 1:1 mapping between feature and components, which makes transformation between models a straightforward task.

*DD7) Derivation of a FM from a component diagram (and vice-versa) follows from a direct mapping between FM elements and UML elements in the SMarty profile (RQ8).* The SMarty UML profile maps to FODA's syntax by a 1:1 correspondence. When deriving a FM from the corresponding component diagram (the inverse derivation), no abstract features will ever be created, as they do not have a corresponding element in a component diagram. Abstract features (see Section V) can still be preserved in the derived FM if one translates an existing FM into a component diagram, changes the latter, and derives the FM back. If the original FM contained abstract features, they will continue to exist, as long as they still root an element that maps to an element in the component diagram. This respect the full round-trip given by DD6.

*DD8) After the generation of each project, one can generate a corresponding component diagram matching its architecture.* By following the mapping between features and components, and with the component selection of a derived product, we take the product line component diagram (if non-existent, we derive it from the FM – see DD7), and resolve its variability according to the feature/component selection. That results in a component diagram documenting the architecture of the corresponding product.

*DD9) Every component corresponds to a single compilation unit. Again, the binding is performed by matching names. Moreover, based on the product configuration and on the 1:1 mapping between features and components, the tool creates, a single Visual Studio project for each selected component, allowing developers to have all the information required to build, maintain, test and evolve selected components (RQ10).* As most of the partner company relies on Microsoft-based technologies, our solution include a single plugin supporting Visual Studio-based projects. Other plugins can be developed for other IDEs, including Eclipse, Netbeans, among others.

*DD10) We define an extensible environment for using different target languages (RQ11).* Through a modular architecture, our tool support plugins that can work with different programming languages. As this is a pilot study, initial support has been developed for C# only.

#### V. THE IN-HOUSE TOOL

This section presents the in-house tool (PlugSPL) developed to meet the requirements set by the TDL. PlugSPL is a plugin-based tool written in C#, from which users design an SPL and develop its components. From that, valid combinations of

components are selected and their matching products generated (testing tools). Figure 1 shows the tool’s workflow, comprised of four main activities, namely, *SPL Design*, *Component Management*, *Product Configuration* and *Product Generation*. In this process, different stakeholders are involved and may overlap roles. Existing roles include testers, managers, developers, etc. For simplicity, we refer to them either as *domain engineers* (if related to the first two activities in PlugSPL), or as *application engineers* (if related to the last two activities).

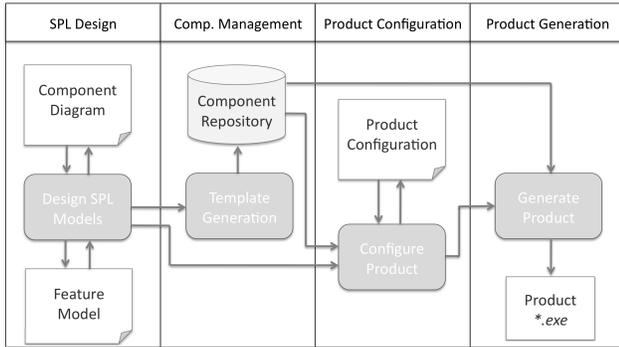


Fig. 1. PlugSPL activities

### A. SPL Design Activity

The SPL Design Activity is the starting point in PlugSPL and aims to support the design of the SPL by means of a graphical FM notation or an UML component diagram.

Although PlugSPL supports FMs, the tool ultimately operates on the level of components, and requires the understanding of concepts such as components, interfaces and realization, as these drive later activities. For domain engineers with a strong background on FMs, but less so in component diagrams, the tool alleviates the modeling activity by supporting the automatic generation of a component diagram from the designed FM. In such case, engineers still manipulate components in other activities, but do not perform any modeling activity in terms of UML component diagrams. Similarly, for those with a strong background in UML component diagrams, but less so in FMs, the tool also supports the automatic generation of a FM from an existing component diagram. In both cases, edits in generated models are automatically synchronized with the models from which they are created, and vice-versa, along with their constraints (full round-tripping). By supporting both FMs and component diagrams and automatic conversion between them, PlugSPL allows an effective communication among different stakeholders in the TDL, with different modeling expertise.

To design an SPL using FMs, domain engineers rely on the FM graphical editor plugin (see Fig. 2). Besides supporting FODA elements (except or-groups), the editor allows marking features as abstract [17] (features are set to be concrete by default). Abstract features exist only to improve the organization of the FM, and are not mapped to any implementation element (class, interface, macro, etc.). Concrete features, on the other hand, follow a 1:1 mapping to a corresponding implementation component, and ultimately to a whole compilation unit. This mapping allows PlugSPL to trace a feature throughout its lifecycle.

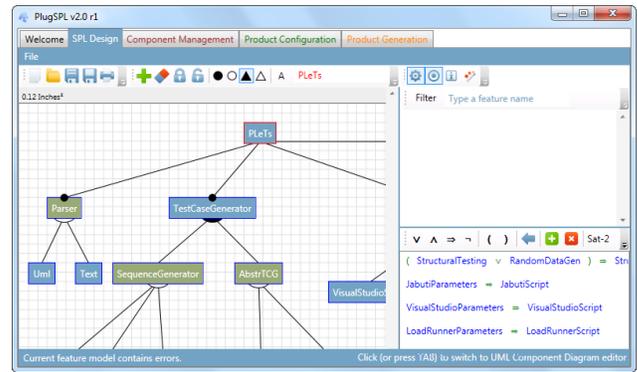


Fig. 2. PlugSPL feature model editor

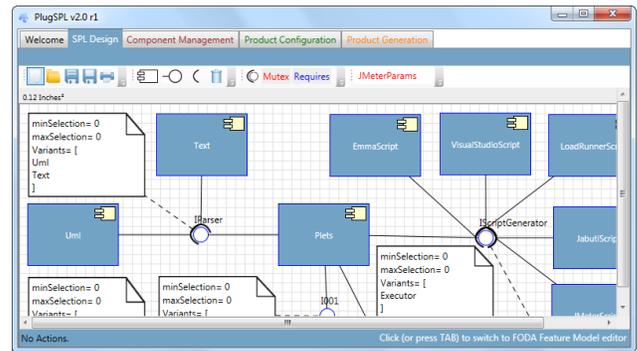


Fig. 3. PlugSPL component diagram editor

For the cases where domain engineers choose to model the SPL using component diagrams, they first select that diagram type (the UML component diagram editor is shown in Fig. 3). In this modeling approach, features are represented as components, which connect to other components by realizing their required interfaces. Since more than one component can realize a given interface, a required interface defines a variation point, and connecting components denote specific variants. These variation points can be further detailed by means of *tags* (UML comments, shown in Fig. 3) and *stereotypes* (e.g., <<Mutex>> group, and <<Requires>> dependencies), allowing a fine-grain control over the variability in place. Tags allow engineers to control the cardinality of instances of each connecting component (captured as *minSelection* and *maxSelection*) and specify the set of possible variants. The value of *minSelection*/*maxSelection* is either zero or one, with the exception that *minSelection* and *maxSelection* are never both zero, and that *minSelection* is always less than or equal to *maxSelection*. Hence, this captures mandatory (*minSelection* = *maxSelection* = 1) and optional features (*minSelection* = 0, *maxSelection* = 1), but prevents the existence of or-groups. The absence of or-groups is currently a limitation, as PlugSPL cannot resolve which variant instance to use when integrating it with a given component. PlugSPL relies on the SMarty variability UML profile [12] as an annotation scheme.

Following a plugin-based architecture, the design activity in PlugSPL can be extended with other plugins supporting different FM modeling notations (e.g., cardinality-based FM [5]) or UML diagrams (e.g., class diagram). It can also be

extended to support different file formats (e.g., SPLOT [11]).

### B. Component Management Activity

In PlugSPL, the component management activity assists domain engineers in the implementation of the SPL components. Given the set of previously defined interfaces, domain engineers define their method signatures (operations) by importing external files (see Fig. 4). Not favoring any specific editor, even a built-in one, allows testing teams to continue using their preferred IDE or editor. Once the interfaces are defined, given the set of declared components, their interfaces, and their connections, PlugSPL generates an initial set of classes that conforms to them; still, these classes are not runnable, but rather skeletons whose associated methods are empty.

In the current C# plugin supporting this activity, each component results in a Visual Studio project, and each interface/class matches exactly one component. These projects are then distributed among different developers and/or testing teams, which then complete their implementation. In this process, developers instantiate interfaces through *fake* statements that are later replaced during *Product Generation*. This is due to the fact that developers cannot (and should not) predict which component can provide the contract of any given interface. Figure 6 illustrates this: instantiation of an `IParser` is done by instantiating the `DummyIParser` interface, a statement that is semantically incorrect, as interfaces cannot be directly instantiated (they only state a contract, and thus lack any behaviour on their own). This resembles the dependency injection pattern [14], while avoiding the burden of keeping XML configuration files, as required by many existing frameworks (e.g., Spring [19]). The penalty, in this, case, is that variability is resolved at an early stage (during *Product Generation*), and not during runtime. Once the implementation of components is completed (they are now in the form of complete Visual Studio projects), they are fed back to PlugSPL, which in turn saves them in the component repository, provided no integration problem occurs.

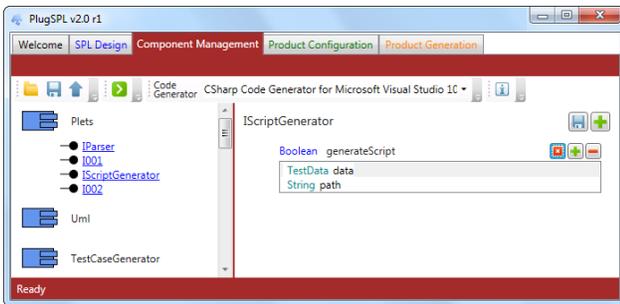


Fig. 4. PlugSPL Component management

### C. Product Configuration Activity

In this activity, application engineers select the components that should comprise a target product (see Fig. 5). To allow such configuration, PlugSPL relies on the feature or component models previously designed, along with the components stored in the project workspace.

PlugSPL generates a tree view of the project's components and their interfaces, along with the set of components that

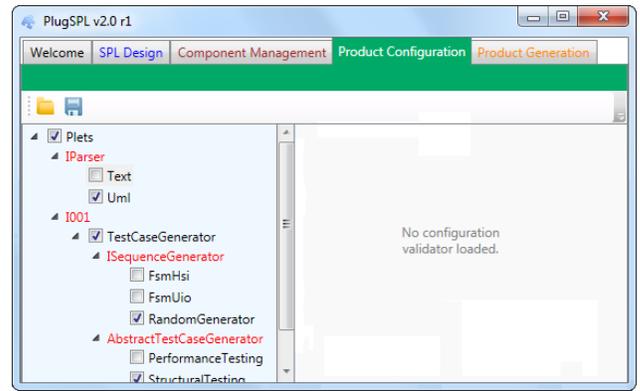


Fig. 5. PlugSPL Product configuration

can connect to each such interface. For instance, following the UML component diagram in Fig. 3, two components implement `IParser`, and serve as its variants: `Uml` and `Text`. In that case, `Uml` and `Text` appear as child nodes of `IParser` in the tree view in Fig. 5.

During configuration, application engineers select at most one component for each provided interface. In accordance with the constraints defined during the SPL design, PlugSPL automatically manages dependencies for selected components. The only exception occurs when configuration conflicts arise, which are then reported and must be manually fixed. Once a product is configured, the configuration is saved in the project workspace, and application engineers proceed to generate target products.

### D. Product Generation Activity

In the product generation activity, from an existing product configuration and its chosen components, PlugSPL selects the corresponding Visual Studio projects generated during *Component Management*. PlugSPL then copies the source code of the selected components from the project workspace to a specified output folder, where components are then glued together. Gluing is performed by replacing extension points that instantiate interfaces (*fake* statements as previously discussed) by the instantiation of the concrete components in the configuration that support such interfaces. Figure 7 illustrates this: on line 5, the instantiation of `DummyIParser` (previously shown in Fig. 6) is replaced by the instantiation of the `Uml` concrete class. Gluing also sets dependencies among different Visual Studio projects, i.e., among related components. From the compilation of all components results a final product (executable testing infrastructure). As in other activities, the plugin supporting this activity is specific to C#-based projects.

## VI. RELATED WORK

The SPL community lacks studies that explicit state the requirements surrounding tool adoption and the corresponding design decisions in the case of custom-made solutions. The few studies attempting to tackle the first part (requirements) are based on collected interviews and surveys [1], [3], and aim to undercover particular challenges that could be the starting point for better tools and methodologies. Our study, although restricted to a single company and its specific requirements,

```

1 public class PLeTs{
2     string fileName, newFileName;
3     //omitted code
4     Console.WriteLine("Initializing <Parser> component...");
5     IParser parser = new DummyIParser();
6     parser.LoadDocument(fileName);
7     parser.ConvertStructures();
8     parser.SaveDocument(newFileName);
9     //omitted code
10 }

```

Fig. 6. Code before the replacement

```

1 public class PLeTs{
2     string fileName, newFileName;
3     //omitted code
4     Console.WriteLine("Initializing <Parser> component...");
5     IParser parser = new Uml();
6     parser.LoadDocument(fileName);
7     parser.ConvertStructures();
8     parser.SaveDocument(newFileName);
9     //omitted code
10 }

```

Fig. 7. Code after the replacement

provides an in-depth discussion over its requirements and context at place. Such requirements have not been fully exploited in the SPL literature, nor have they been fully addressed by existing tools (most notably, full traceability and round-trip over different models). Some teams report some of their design decisions when creating SPL-related tools, e.g., Feature IDE [16]. However, decisions are not explicitly backed up by any industry-set requirements, but rather, from the creators own experience [16].

Other researchers investigate differences among existing tools [6], [10], but do not collect feedback based on industrial cases where these tools are used, or to which extent they succeed or fail when supporting SPL adoption.

On the tool development side, different solutions have been proposed, including both commercial and open-source. The two most popular commercial products today are pure::variants [2], from pure::systems, and Gears [8], from Big Lever Software Inc. Although they represent the most complete toolset for product line adoption, the specificity of the TDL's requirements make them unsuitable. The open-source arena is no different, although a plethora of solutions exist, ranging from web-based solutions [11], to Eclipse plugins [16]. A comprehensive list of existing tools, either commercial or open-source, is presented in [10].

## VII. CONCLUSION AND FUTURE WORK

This paper presented a set of requirements elicited in the context of an industrial partner and its Technology Development Lab. Through a pilot study, we collected specific needs targeting tool adoption for implementing a SPL-based solution for test products, and argue that existing tools, either commercial or open-source, do not meet the specificity of the requirements at hand. We then presented our design decisions when creating an in-house tool to fulfill our partner needs, along with a brief discussion of its supported workflow. We claim the reported requirements and design decisions as the two contributions of this work, as currently, few studies bring such discussion. Our work adds to that in the sense that the elicited requirements show practical scenarios that tool vendors and/or implementers may consider supporting; the design decisions, in turn, may be reused or adapted to improve existing tools or devise new ones targeting similar requirements.

As future work, we aim to keep track of our partner company needs and elicit new requirements as the SPL goes beyond the current pilot study, ideally being adopted by all testing teams.

## VIII. ACKNOWLEDGEMENTS

Elder Rodrigues is a researcher at the Center of Competence in Performance Testing, a partnership between Dell and PUCRS.

## REFERENCES

- [1] T. Berger, R. Rublack, D. Nair, J. M. Atlee, M. Becker, K. Czarnecki, and A. Wkasowski. A survey of variability modeling in industrial practice. In *Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems*, 2013.
- [2] D. Beuche. Modeling and building software product lines with pure::variants. In *Proceedings of the 16th International Software Product Line Conference - Volume 2*, 2012.
- [3] L. Chen and M. Babar. Variability management in software product lines: An investigation of contemporary industrial challenges. In *Software Product Lines: Going Beyond*, Lecture Notes in Computer Science. Springer, 2010.
- [4] L. Costa, E. Rodrigues, R. Czekster, F. Oliveira, M. Silveira, and A. Zorzo. Generating performance test scripts and scenarios based on abstract intermediate models. In *Proceedings of the 24th International Conference on Software Engineering and Knowledge Engineering*, 2012.
- [5] K. Czarnecki, S. Helsen, and U. W. Eisenecker. Formalizing cardinality-based feature models and their specialization. *Software Process: Improvement and Practice*, 2005.
- [6] M. Dammagh and O. Troyer. Feature modeling tools: Evaluation and lessons learned. In *Advances in Conceptual Modeling. Recent Developments and New Directions*, Lecture Notes in Computer Science. Springer, 2011.
- [7] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson. Feature-oriented domain analysis (FODA) feasibility study. Technical report, Carnegie-Mellon University Software Engineering Institute, 1990.
- [8] C. Krueger and P. Clements. Systems and software product line engineering with BigLever software Gears. In *Proceedings of the 16th International Software Product Line Conference*, 2012.
- [9] C. W. Krueger. New methods in software product line practice. *Communications of the ACM*, 2006.
- [10] L. B. Lisboa, V. C. Garcia, D. Lucrédio, E. S. de Almeida, S. R. de Lemos Meira, and R. P. de Mattos Fortes. A systematic review of domain analysis tools. *Information and Software Technology*, 52(1):1–13, 2010.
- [11] M. Mendonca, M. Branco, and D. Cowan. S.P.L.O.T.: software product lines online tools. In *OOPSLA Companion*, 2009.
- [12] E. Oliveira, I. M. Gimenes, and J. Maldonado. Systematic management of variability in UML-based software product lines. *Journal of Universal Computer Science*, 2010.
- [13] K. Pohl, G. Böckle, and F. J. v. d. Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, 2005.
- [14] D. R. Prasanna. *Dependency Injection*. Manning Publications Co., 1st edition, 2009.
- [15] H. Spencer and G. Collyer. #ifdef Considered harmful, or portability experience with C news. In *USENIX*, 1992.
- [16] T. Thüm, C. Kästner, F. Benduhn, J. Meinicke, G. Saake, and T. Leich. FeatureIDE: An extensible framework for feature-oriented software development. *Science of Computer Programming*, 2012.
- [17] T. Thum, C. Kastner, S. Erdweg, and N. Siegmund. Abstract features in feature modeling. In *Proceedings of the 15th International Software Product Line Conference*, 2011.
- [18] M. Utting and B. Leggard. *Practical model-based testing: a tools approach*. Morgan Kaufmann, 2006.
- [19] C. Walls and R. Breidenbach. *Spring in action*. Manning Publications Co., 2007.

# A COSMIC Measurement Procedure for BPMN Diagrams

Beatriz Marín, José Quinteros  
Facultad de Ingeniería  
Universidad Diego Portales  
Ejército 441, Santiago, Chile  
{beatriz.marin, jose.quinteros}@mail.udp.cl

**Abstract**—In recent years, the estimation of effort, budget and time has been a fundamental step in the software project life cycle. Having this knowledge it is possible to manage and control a software project. So that, it is important to have these estimations as early as possible in order to produce a sound schedule for the project. Taking into account that BPMN diagrams are widely used in industry for analysis, in this paper we present a functional size measurement procedure based in the COSMIC method that measure these diagrams. With this size it is possible to obtain estimations at early stages of software cycle.

**Keywords**—BPMN; COSMIC; Estimation, Functional Size; Functional Size Measurement; Business Process

## I. INTRODUCTION

In the last years, the estimation of effort, budget and time has become an important input in the software project management [1], since having this knowledge it is possible to better planning and control a software project. Thus, it is important to have these estimations as early as possible in order to produce a sound schedule for the project.

Nowadays, there are several methods that allows the estimation of budget, effort, and time; which are mainly based on COCOMO [2]. These methods present late estimations since they used lines of code (LOC) as input in the measurement process [3], resulting in measures that are dependents of the programming language. Other estimation methods are based on Function Points Analysis (FPA) [4]. FPA proposes the functional size measurement, which use design artifacts as input in the measurement process, so that FPA-based methods provide more early measurements than COCOMO-based methods, and also FPA-based methods are independent of the programming language [5].

To measure the functional size of software applications, there are five standard measurement methods [6], [7], [8], [9] and [10]. These methods have been illustrated in the measurement of the functional size of final applications. However, project leaders need indicators in early stages of software cycle for a better management of software projects. Thus, it is necessary to define how the standards can be applied to the diagrams that are used in early stages of software cycle. A measurement procedure corresponds precisely to this specification [9].

We have selected the COSMIC measurement method to specify a measurement procedure since, in contrast to other

standard functional size measurement methods, COSMIC can be applied to any type of software (including MIS, real-time software, embedded software, and hybrid software), and it also allows the measurement of multi-layer applications.

Taking into account that BPMN diagrams [11] are widely used in industry for analysis, in this paper we present a functional size measurement procedure based in COSMIC that measure the functional size of BPMN diagrams. With this size, project leaders can calculate productivity indicators, the price to be charged to clients, effort [12], etc.

The rest of the paper is organized as follows: Section 2 presents the background of this work. Section 3 presents the design of a measurement procedure. Section 4 presents the automation of the procedure and its application to an example. Finally, Section 5 presents some conclusions and further work.

## II. BACKGROUND

This section presents the main characteristics of BPMN and COSMIC, and some relevant related works.

### A. BPMN

The Business Process Model and Notation (BPMN) [11] was recognized as standar by OMG in 2011. The main goal of BPMN is to provide a notation that will be understandable by all business users, such as the analysts that creates the processes, the technical developers that provides the technical platform for these processes, and the business people who will manage and will monitor these processes. Thus, the main advantage of using BPMN is that it creates a standardized bridge between the design and execution of business processes.

The main elements of BPMN diagrams are the following:

- Flow objects are used to define the behaviour of business process. They can be events (start, end), activities (processes, tasks), or gateways.
- Data is represented with four elements: data objects, data inputs, data outputs, and data stores.
- There are four connecting objects that allow the connection among flow objects and other information: sequence flows, message flows, associations, and data associations.
- There are two ways of grouping the primary modeling elements through swimlanes: pools and lanes.

- Artifacts are used to provide additional information about the process. They are groups and text annotations.

### B. COSMIC

The COSMIC measurement method version 3.0.1 [13] has three phases: the measurement strategy phase, the mapping phase, and the measurement phase.

In the strategy phase, the purpose, the scope and the level of granularity of the measurement must be determined. In addition, functional users must be identified. Functional users are defined as a type of user that send or receive data to/from the functional processes of a piece of software.

In the mapping phase, functional processes and the data groups involved in these functional processes must be identified. A functional process is a basic component of the set of requirements that describe what the software shall do. A functional process comprises a single, cohesive, and independent set of data movements. An optional step of this phase is the identification of data attributes that are related to the data groups.

In the measurement phase, the identification of data movements must be carried out, and the measurement function must be applied. A data movement moves one or more types of data attributes belonging to a only one type of data group. There are four types of data movements, which are defined as:

- **Entry:** It is a type of data movement that moves a data group from the functional user across the boundary within the functional process that is required.
- **Exit:** It is a type of data movement that moves a data group from a functional process across the boundary to the functional user that needs it.
- **Read:** It is a type of data movement that moves a data group from the storage to the functional process that requires it.
- **Write:** It is a type of data movement that moves a data located in a functional process inside of a storage.

The measurement function assigns 1 CFP (COSMIC Function Point) to each data movement. Then, the results are aggregated in order to obtain the functional size of the software measured.

### C. Related works

There are some works that measure the functional size of software from business models. In [14] an approach for measuring the functional size of ERP systems is presented by applying IFPUG FPA to business models. Therefore, in contrast to approaches based in COSMIC, it has the limitation related to the precision of measurements of IFPUG FPA.

In [15], a measurement approach based on COSMIC was presented. This approach uses as input UML use-case, activity and class diagrams. However, it does not present a mapping among business and COSMIC concepts, so that it is difficult to apply to BPMN diagrams.

A recent COSMIC measurement procedure for BPMN diagrams was presented in [16]. It presents rules that map two

different concepts of COSMIC to the same concept of BPMN, which can produce ambiguities. The application of this work is manual, which is an error-prone and time consuming task.

## III. DESIGN OF E-FSMP

This section presents E-FSMP, meaning Early Functional Size Measurement Procedure, which is a procedure based in COSMIC that allows the measurement of the functional size of BPMN diagrams. Since COSMIC is comprised of three phases, the E-FSMP is also comprised of three phases.

### A. Measurement strategy phase

In this phase, the strategy of the effort of measurement is defined by the identification of the purpose, the scope, the level of granularity, and the functional users.

The **purpose** of E-FSMP has been defined as: To measure the functional size of the requirements specifyied in BPMN diagrams. The **scope** corresponds to the BPMN diagrams contractually agree as the requirements.

The **level of granularity** is related to the requirements and the natural evolving of some requirements especifyied in the software artifacts to be measured. The standard level of granularity is as at which individual functional processes have been identified and their data movements identified.

The **functional users** correspond to any type of user that interacts with the software, which is specified in the FUR. In BPMN diagrams, these users correspond to the role defined to execute the processes in the intended software system. These users are functional users because they send (or receive) data to (from) these processes.

The **boundary** is defined as a conceptual interface between the software being measured and its functional users. In BPMN diagrams, this boundary corresponds to the pool. To avoid mistakes in the identification of the functional users and the boundaries of a BPMN diagram, Table 1 shows the rules that have been defined to identify the functional users (Rule 1) and the boundary (Rule 2).

TABLE I. E-FSMP MEASUREMENT STRATEGY RULES

Number	Description
Rule 1	Identify 1 functional user for each role in the BPMN diagram.
Rule 2	Identify 1 boundary between a functional user and a <b>pool</b> of a BPMN diagram.

### B. Mapping phase

When the measurement strategy has been established, the mapping phase is carried out. In this phase, the functional processes and data groups must be identified.

A **functional process** is a set of functionalities of the software system that allows the achievement of a functional requirement. Generally, in BPMN diagrams, the functional requirements are presented as activities. Thus, the activities of the BPMN diagram are considered as functional processes.

However, the functional processes can be identified several times if they are accessed from more than one role. To avoid duplicity, each activity is identified as a functional process only for the first role that access it.

Later, the **data groups** that participates in the functional processes must be identified. Data groups correspond to the incoming and outgoing documents of each activity of the BPMN diagram. Table 2 shows the rules that have been defined to identify the functional processes (Rule 3), to avoid duplicity in the identification of the functional processes (Rule 4) and to identify data groups (Rule 5).

TABLE II. E-FSMP MAPPING RULES

Number	Description
Rule 3	Identify 1 functional process for each <b>activity</b> that can be accessed by a role of the BPMN diagram.
Rule 4	Identify an activity as a functional process only once.
Rule 5	Identify 1 data group for each <b>data input</b> or <b>data output</b> of the activities specified in the BPMN diagram.

### C. Measurement phase

In this phase, data movements of each functional process are identified. Then, the measurement function is applied, and the results are aggregated to obtain the functional size of each functional process, which later are aggregated to obtain the functional size of software.

Each functional process move two or more data movements. Each **data movement** moves one data group. Data movements can be entry, exit, read, and write. An entry data movement corresponds to the movement of an input data object to an activity. An exit data movement corresponds to the movement of an output data object from an activity. A read data movement corresponds to a movement of a data object from the database to the scope of an activity. A write data movement corresponds to a movement from an activity to the database.

When all the data movements have been identified, the **measurement function** is applied. This function assigns 1 CFP to each data movement identified. Once the measurement function has been applied, the measures can be aggregated to obtain the functional size of each functional process of the application as well as the whole application. Table 3 shows the rules defined for the identification of data movements (Rule 6, 7, 8, and 9); and the measurement rules (Rule 10 and 11).

TABLE III. E-FSMP MEASUREMENT RULES

Number	Description
Rule 6	Identify 1 entry data movement for an input data object to an activity.
Rule 7	Identify 1 exit data movement for an output data object from an activity.
Rule 8	Identify 1 read data movement for each different data object from the database to an activity.
Rule 9	Identify 1 write data movement for each different data object that goes from an activity to the database.
Rule 10	Aggregate the CFP related to the <b>data movements</b> identified in each activity to obtain the functional size of that process.
Rule 11	Aggregate the CFP related to the <b>functional processes</b> identified to obtain the functional size of the software.

With the 11 rules presented it is possible to measure the functional size of software from their BPMN diagrams by the application of COSMIC FSM method.

## IV. AUTOMATION OF E-FSMP

The manual measurement of functional size is generally very time-consuming and has many precision errors. For this reason, it is necessary to automate the measurement process to obtain a solution that can be applied in industrial contexts.

The tool that implements the E-FSMP was developed using Laravel [17], which is a PHP framework to develop web applications. This tool allows the measurement of BPMN or use case diagrams (see Fig. 1). The first step using the E-FSMP tool is to charge the XML file with the specification of a BPMN diagram. This XML file was automatically created by Bizagi tool from the BPMN specification.

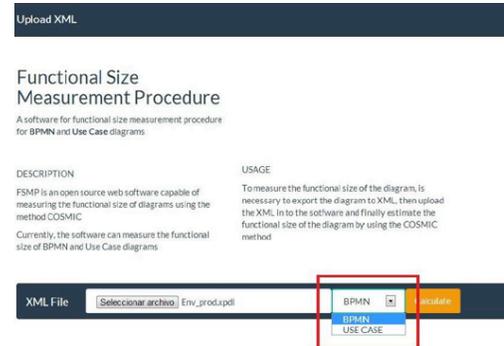


Fig. 1. Home page of E-FSMP tool.

Then, the tool apply the rules defined by E-FSMP in order to obtain the functional size. To do this, the tool first verify the structure of the XML file. If the file does not correspond to a BPMN diagram, then the tool shows an error message. If the XML file corresponds to an specification of a BPMN diagram, the tool identify tasks in the XML description, and then, it identify data objects (inputs and outputs) and databases.

Later, the E-FSMP tool assigns 1 CFP to each data movement identified. Finally, the tool aggregates the results in order to obtain the functional size of the software that is measured.

Fig. 2 shows an example of BPMN diagram. This diagram correspond to the requirements specification for the sent process of a post office. The system is called SIGECO (Post Office Management System), which supports sent and reception of postal packages. Regarding the sent process (see Fig. 2), sender goes to the post office and requests to send a product. The post officer verify the product and the shipment data, and then register the shipment. Then, sender receives a code that will use to track the product. Later the post officer send the product, which is received in a different post office, which is responsible of the delivery process to carry out the product to the final receptor.

This example was manually measured applying the E-FSMP obtaining 24 CFP. Finally, the xml representation of the sent process of SIGECO was measured using the E-FSMP tool. The functional size obtained by the E-FSMP tool corresponds to 24 CFP, which is the same size obtained manually. The report of the E-FSMP tool shows the functional size measured, the tasks recognized as functional processes, and the data movements that can occur in each functional process.

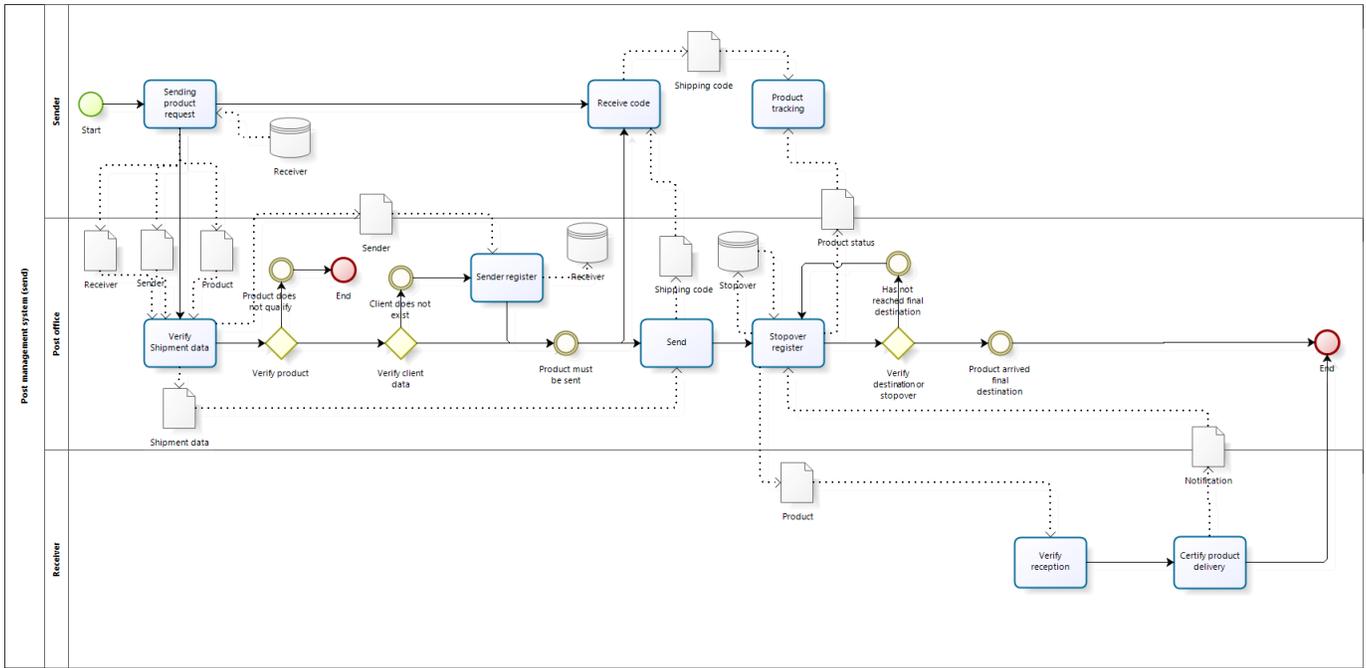


Fig. 2. Sent process of the post office.

## V. CONCLUSIONS AND FURTHER WORK

In this paper, a functional size measurement procedure called E-FSMP has been presented. E-FSMP applies the standard COSMIC measurement method to BPMN diagrams by means of the application of 11 rules.

In addition, a tool that automates the application of E-FSMP has been presented. With this tool, measurement results can be obtained quickly and avoiding precision errors that commonly occurs with manual measurements. Thus, the main contribution of this work is the specification of E-FSMP and the tool that automates the measurement of functional size at early stages in software development.

Further work includes the execution of real case studies that allows the validation of E-FSMP and verification of the scalability and the performance of the E-FSMP tool.

### ACKNOWLEDGMENT

This work was funded by FONDECYT project TESTMODE (Ref. 11121395, 2012-2015).

### REFERENCES

[1] R. Meli, A. Abran, T. Ho Vinh, and S. Oligny, On the Applicability of COSMIC-FFP for Measuring Software Throughout its Life Cycle, 11th European Software Control and Metrics Conference Munich 2000.  
 [2] B. W. Boehm, C. Abts, A. Winsor Brown, S. Chulani, B. K. Clark, E. Horowitz, et al., *Software cost estimation with cocomo II*: Prentice Hall, 2000.  
 [3] S. D. Conte, H. E. Dunsmore, and V. Y. Shen, *Software Engineering Metrics and Models*. Redwood City, CA, USA: Benjamin-Cummings Publishing Co., Inc., 1986.  
 [4] A. Albrecht, Measuring Application Development Productivity, IBM Applications Development Symposium, 1979.

[5] B. Marín, O. Pastor, and A. Abran, Towards an accurate functional size measurement procedure for conceptual models in an MDA environment, *Data & Knowledge Engineering*, vol. 69, pp. 472–490, 2010.  
 [6] ISO/IEC, ISO/IEC 20926, Software Engineering – IFPUG 4.1 Unadjusted Functional Size Measurement Method – Counting Practices Manual, 2003.  
 [7] ISO/IEC, ISO/IEC 20968, Software Engineering – Mk II Function Point Analysis – Counting Practices Manual, 2002.  
 [8] ISO/IEC, ISO/IEC 24570, Software Engineering – NESMA Functional Size Measurement Method version 2.1 – Definitions and Counting Guidelines for the application of Function Point Analysis, 2005.  
 [9] ISO/IEC, ISO/IEC 29881, Software Engineering – FiSMA Functional Size Measurement Method version 1.1, 2008.  
 [10] ISO/IEC, ISO/IEC 19761, Software Engineering – COSMIC-FFP – A Functional Size Measurement Method, 2003.  
 [11] OMG, Business Process Model and Notation (BPMN) 2.0, 2011.  
 [12] C. Gencel, How to Use COSMIC Functional Size in Effort Estimation Models?, IWSM / MetriKon / Mensura, 2008.  
 [13] T. C. M. P. Committee, The COSMIC Functional Size Measurement Method - Version 3.0.1, The Common Software Measurement International Consortium (COSMIC)2009.  
 [14] M. Daneva, Measuring reuse of SAP requirements: A model-based approach, Proceedings of the 1999 Symposium on Software reusability, Los Angeles, California, 1999.  
 [15] K. G. van den Berg, T. Dekkers, and R. Oudshoorn, Functional size measurement applied to UML-based user requirements, Proc. Software Measurement European Forum (SMEF 2005), 2005.  
 [16] C. Monsalve, A. Abran, and A. April, Measuring software functional size from business process models, *International Journal of Software Engineering and Knowledge Engineering*, vol. 21, pp. 1-28, 2011.  
 [17] SolSPACE. *Laravel Framework*. Available: <http://laravel.com/>

# Proposing a Software Process Model for Follow the Sun Development

Josiane Kroll<sup>1</sup>, Ita Richardson<sup>2</sup>, and Jorge L. N. Audy<sup>1</sup>

<sup>1</sup>Pontifical University Catholic of Rio Grande do Sul (PUCRS), Porto Alegre, Brazil

<sup>2</sup>Lero - The Irish Software Engineering Research Centre, University of Limerick (UL), Limerick, Ireland

josi.unc@gmail.com, ita.richardson@ul.ie, audy@pucrs.br

**Abstract**— Many software organizations are restructuring their software development groups by extending operations to offshore software development centers. Thus, a Follow the Sun (FTS) development is a potential strategy for these organizations. FTS can help with reducing the software development life cycle duration and consequently the time-to-market. However, while the FTS concept looks promising in theory, it appears to be difficult in practice and software organizations have a pressing need for support in how to successfully implement FTS in a global software environment. In this paper, we combine the results from prior work in FTS and a design validation method conducted by experts to propose a software process model for FTS development, named FTS-SPM (Follow the Sun Software Process Model). Our paper describes how we built the FTS-SPM and draws recommendations for software organizations interested in practicing FTS.

**Keywords** - Follow the Sun; Global Software Engineering; Software process; Time zone management; Virtual teams.

## I. INTRODUCTION

Follow the Sun (FTS) development is an alternative for global software environments when trying to manage problems related to temporal distance. Its main purpose is to reduce software development life cycle duration or time-to-market [1]. However, while the FTS concept looks promising in theory, it appears to be difficult in practice. It was observed that there is a great interest from the software industry in practicing FTS, but the lack of theoretical studies combined with software processes, models and practices make its adoption difficult [3]. In addition, many software organizations have attempted to implement FTS, but have abandoned it after some point because of this difficulty [2].

In this paper, we propose a software process model to support FTS development in global software projects. This process was called FTS-SPM (Follow the Sun Software Process Model). FTS-SPM was built based on the results from prior work [4] [5] [6] [7] and based on the design validation method conducted with experts from Lero - The Irish Software Engineering Research Centre (Ireland) and MuNDDoS research group (Brazil).

## II. THEORETICAL BACKGROUND

### A. Follow the Sun Development

In Global Software Engineering (GSE), team members are distributed in different places, countries or even continents. In some cases, these teams may be from the same organizations.

In other cases, there is collaboration between teams from different organizations [8].

Follow the Sun (FTS) is a special case of GSE. It is applied in the context of GSE to take advantage of the temporal distance between several development sites located in different time zones. FTS is uniquely focused on speed of development [1]. It is applied to software projects when a software product needs to be developed quickly and the cost is irrelevant to the client. As team members are distributed across multiple time zones, organizations can develop software twenty-four hours continuously [3].

At the beginning and at the end of each working day shift there is a handoff. *Handoff* is a term adopted in the literature to define the transition process from one site to another [6]. Handoffs are performed on a daily basis to present a status update and to pass on unfinished tasks (project source) from one site to another [2].

### B. Related Work

Hess and Audy [9] proposed a software process for handoffs to alleviate difficulties faced by teams during the development phase of FTS projects. Findings from this study show that is possible to reduce development difficulties in FTS using the proposed process. Their process is based on Composite Persona (CP) and 24hr Design and Development concepts.

Richardson et al. [8] developed a software process called Global Teaming (GT). This process includes specific practices and sub-practices which detail specific recommendations for organizations that are implementing GSE. The main contribution of this study is a supporting mechanism for the implementation of global software projects.

Denny et al. [10] explored the utilization of agile practices for 24-Hour Knowledge Factory (24HrKF) environments. They aim to search for solutions that enable handoffs to be practiced effectively. Thus, this study describes a process called CPro that addresses several of the operational issues related to the 24HrKF environment.

Yap [11] also discusses agile methods, but with a different purpose. Her study describes the use of XP (Extreme Programming) to develop a globally distributed, around-the-clock software development project. In Yap's study, a programming team was distributed across three sites and they used collective ownership of code. At the end of her study, the author concludes that XP works for a globally distributed group

performing around-the-clock continuous development with a shared codebase.

Studies performed by Carmel, Espinosa, and Dubinsky [12] and Carmel, Dubinsky, and Espinosa [13] discuss mainly FTS definition, characteristics, and challenges. The first study provides a conceptual foundation and a formal definition of FTS. The second study, presents the details of the FTS concept and the outcomes of a first quasi-experiment designed to measure the speed of software work on FTS mode.

### III. PRIOR WORK

We conducted some prior work in FTS before carrying out the study presented in this paper [4] [5] [6] [7]. We present an overview of this prior work, focusing on the aspects relevant to the software process model. We have categorized our studies into 2 phases: Phase 1 - Exploratory studies and Phase 2 - Case studies.

#### A. Phase 1 – Exploratory studies

Kroll et al. [4] conducted a mapping study of the literature in GSE to identify best practices for FTS development. They limited their study to identify practices conducted in GSE and at the same time recommended for FTS. Although not described in the literature, the evidence demonstrates that FTS is carried out by software engineers, but only in part. Nine best practices and key aspects for FTS implementation were presented in this paper.

Kroll et al. [5] extends the study published by Kroll et al. [4]. This study provides new information about FTS best practices and challenges. They substantially extend the empirical evaluation of FTS which was conducted in that previous study. As a result, they identified 36 best practices and 17 challenges for FTS implementation.

#### B. Phase 2 – Case studies

Kroll et al. [6] conducted a case study at Infosys Technologies in Bangalore, India. This study examined the feasibility and outcomes of FTS. Infosys’ experts considered some best practices reported in the literature [4] [5] to design a software process for FTS. This study presents details of software practices, presents solutions performed to overcome the challenges when developing a software application in FTS mode and highlights eight lessons learned. The authors conclude this study by showing that FTS works for GSE projects with some evidence that FTS can be used to compress duration.

Kroll et al. [7] report how handoffs management should be performed in FTS development. They present an experience report describing handoffs development and management in a FTS software project. The results describe the participants' perception about software engineering activities performed, challenges faced and solutions performed to minimize these challenges. They also highlight management elements for handoffs.

### IV. RESEARCH METHODOLOGY

We proposed the initial software process model for FTS, called FTS-SPM (Follow the Sun Software Process Model) version 1, based on the results from prior work (see Figure 1).

Subsequently, we built a second and third version of the FTS-SPM, with a view to improving the initial software process model design. This was done by collecting input from research experts, and this design validation was carried out using expert input from those in the Lero and MuNDDos research groups.

We had the opportunity to discuss the initial proposal with researchers and visiting researchers at the Annual NUIG-UL (National University of Ireland, Galway/University of Limerick) Research Day held in Galway (Ireland). There, we collected data by notetaking from feedback provided by research experts. Based on these data, we propose the second version of our FTS-SPM.

Following the validation design planning, we presented the second version of the FTS-SPM at Lero workshops. We collected data during the workshops also by notetaking from feedback and further discussion with research experts. These data were also discussed in parallel with experts from the MuNDDos research group. Based on all data collected during the validation design, we proposed the third and final version of the FTS-SPM.

### V. SOFTWARE PROCESS MODEL FOR FOLLOW THE SUN

#### A. The Design Validation

The FTS-SPM first version is presented in Figure 1. Through in-depth analysis of results from prior work (Section 3), significant themes directly correlating with best practices and lessons learned emerged. To make sense, these themes were synthesized as sub processes (SP) in the proposed FTS-SPM.

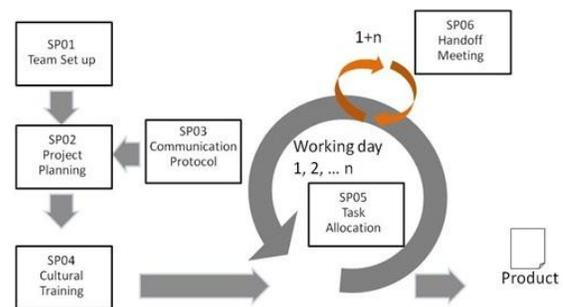


Figure 1. FTS-SPM version 1.

The proposed FTS-SPM version 1 comprised six sub processes: *SP01: Team Setup*, *SP02: Project Planning*, *SP03: Communication Protocol*, *SP04: Cultural Training*, *SP05: Task Allocation*, and *SP06: Handoff Meeting*. The sequence flow (arrows) between sub processes shows in which sequence each sub process is developed.

Figure 2, which is an evolution of the FTS-SPM version 1, shows modified sequence flows between sub processes. In addition, an initial and a final state were added to the FTS-SPM. We also changed SP06’s name and included arrows to show how the information moves through sub processes. SP06 was called Handoff Sessions on the FTS-SPM second version.

We presented the second version of the FTS-SPM at the Lero workshops and questions about the sequence flow between

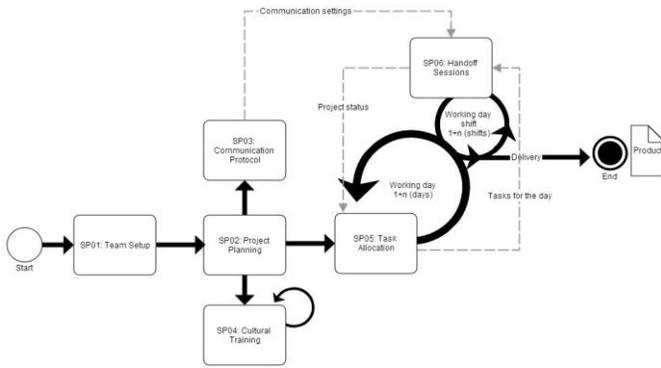


Figure 2. FTS-SPM version 2.

SP05 and SP06 emerged. However, the sequence flow between sub processes was considered inadequate by some experts from Lero. Following recommendations from Lero researchers, particularly those belonging to the Process Quality Research Group, we introduced new changes to the FTS-SPM version 2. These changes were made between SP05 and SP06 to support FTS characteristics. All changes in the process model were also supported by experts from the MuNDDos research group.

### B. The FTS-SPM Overview

The final version of the FTS-SPM (shown in Figure 3), comprises six sub processes: *SP01: Team Setup*, *SP02: Project Planning*, *SP03: Communication Protocol*, *SP04: Cultural Training*, *SP05: Task Allocation* and *SP06: Handoff Sessions*.

The FTS-SPM has an initial and final state. The initial state causes the process to start with Team Setup (SP01). The final state is the end of the process when all tasks were finished, at which point there is a software delivery. SP01, Team Setup, starts the process. It aims to identify available sites and allocates human resources for the project. Information about each site should be collected in order to make future decisions. It is important to verify if there are staff, cost or scope restrictions in each site. These restrictions and others related to project goals should be considered to define priorities in order to select appropriate sites.

SP02, where project planning is defined, is started following SP01. SP01 provides information to develop the project plans, and these are developed by the project manager.

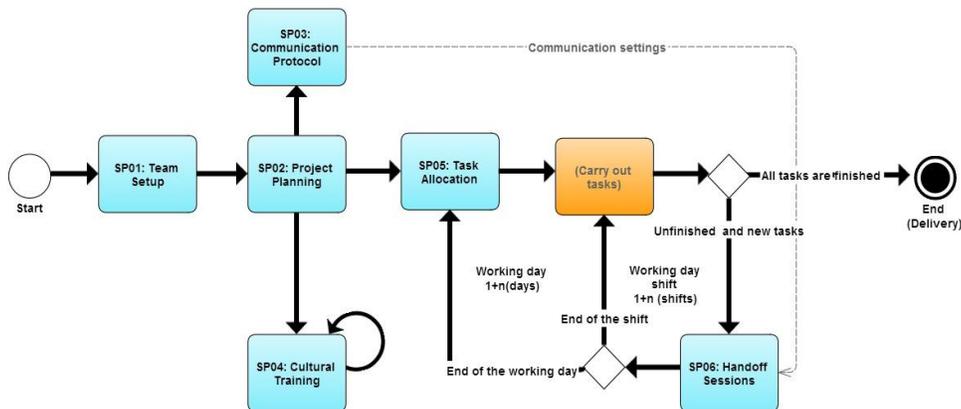


Figure 3. The Proposed FTS-SPM (Follow the Sun Software Process Model).

SP03, SP04 and SP05 are started in parallel following SP02. SP03 defines communication resources and the schedule for synchronous communication between sites. The project manager can suggest technologies or tools already used in other projects. SP04 develops cultural training sessions in order to establish trust between team members. SP04 may be developed many times during the project to re-establish the trust between team members (loop arrow).

At the beginning of each working day, SP05 is undertaken, as it provides tasks for the day. A software project may have many working days. Within SP05, the sequence and dependency relationships between tasks must be identified. All details about tasks sequence and dependency should be described during the project planning.

SP06 is started following SP05. SP06 aims to receive and to transfer tasks in progress, new tasks and project updates. At the beginning and at the end of each working day shift, SP06 is undertaken. One working day may have at least two working day shifts. The process finishes when at the end of a working day shift, there are no more tasks to develop.

'Carry out tasks' is an internal sub process of the organization. Each organization defines how it should be executed. We show this sub process in our FTS-SPM to represent how it is related to other sub processes.

In the first diamond, the process can finish if all tasks are finished or can start SP06, if there are unfinished or new tasks to transfer to another site. In the second diamond, a new working day shift starts if is end of the shift or else, if is the end of the working day, SP05 starts.

Arrows in the FTS-SPM show the sequence flows between sub processes. An additional arrow is included between SP03 to SP06 indicating the relationship between those sub processes. The communication settings defined in SP03 are used in SP06.

## VI. DISCUSSION

The proposed FTS-SPM comprises six sub processes. All sub processes were presented the second version of the FTS-SPM at the Lero workshops. However, questions about the sequence flow between SP05 and SP06 emerged. However, the sequence flow is also implemented in global software projects.

It makes sense, because FTS is a subset of GSE [1]. However, some characteristics of the FTS approach make its sub processes more difficult to implement. For example, in the FTS approach, team members work on the same tasks and daily handoffs are a strong characteristic of the FTS projects [1]. Thus, at the beginning and the end of the day there is a handoff to transfer unfinished or new tasks. In a traditional global project handoffs also happen, but with less frequency. In the absence of a structured handoff procedure, defects may occur due to lack of understanding of the current state of the task [10].

As shown in Figure 3, sub processes are related to each other. Thus, activities from a particular sub process can support one or more sub processes (e.g. activities from SP03, Communication protocol, supports the activities from SP06 – Handoff sessions). This is an important characteristic of software process models. Besides this, software process models help to better understand the software development process and present the possibility of discussing process improvements [8].

Through the analysis of the FTS-SPM, we highlight some recommendations for software organizations interested in practicing FTS. These recommendations aim to improve successful implementation of FTS: a) Project planning (SP02) should be performed after defining Team Setup (SP01). In some cases, to allow global teams to work in FTS mode, team members need to be re-allocated within different time zones. This makes time zones manageable; b) Communication protocol (SP03) is defined according to the team characteristics. Thus, cultural and temporal diversity in FTS projects requires more attention before selecting collaborative technologies for the project; c) An imbalance in team members' experience can be solved developing Cultural Training (SP04). This sub process is also used to define standards, rules and strategies to improve software product development.

Finally, we highlight the relevance of a software process model to bring theory closer to practice. Organizations can benefit from the FTS-SPM to establish their own process for FTS.

## VII. IMPLICATIONS AND CONCLUDING REMARKS

Our findings propose a software process model for FTS implementation, the FTS-SPM (Follow the Sun Software Process Model). Specifically, our qualitative evidence, gathered through prior work in FTS and design validation method, shows: 1) How best practices and lessons learned contribute to define sub processes; 2) How sub processes are related and in which sequence flow the process should be carried out; 3) How specific sub process status, SP05: Task allocation promotes SP06: Handoff sessions to determinate the sequence flow to SP05 or to determine the beginning or the end of the day (final state).

Our work has practical implications for organizations that want to adopt FTS. In order to make FTS work effectively, an organization should pay attention not only technical issues such as adopting rich collaborative technologies, but also to better allocate team members in different continents and time zones.

It is useful for improving coordination across development sites.

While different experience levels can make the transfer of knowledge between team members difficult, managers should consider it as an opportunity for spontaneous learning across team members. Moreover, an effective team can better collaborate if team members keep working together on other projects. Finally, we highlight the need for more work to identify how each sub process should be developed and maintained over the project. Other studies are being developed to fill this gap.

## ACKNOWLEDGMENT

The first author is supported by the PDTI program, financed by Dell Computers of Brazil Ltd. (Law 8.248/91) and the second author is partially supported by the Science Foundation Ireland grant 10/CE/I1855 to Lero ([www.lero.ie](http://www.lero.ie)).

## REFERENCES

- [1] E. Carmel and J. A. Espinosa, *I'm Working While They're Sleeping: Time Zone Separation Challenges and Solutions*, Kindle Edition, 2011, 188 p.
- [2] C. Visser and R. V. Solingen, "Selecting Locations for Follow-the-Sun Software Development: Towards A Routing Model", *International Conference on Global Software Engineering (ICGSE)*, 2009.
- [3] R. Prikładnicki and E. Carmel, "Is time-zone proximity an advantage for software development? The case of the Brazilian IT industry". *Proceedings of the 2013 International Conference on Software Engineering (ICSE '13)*, Piscataway, NJ, USA, 2013, pp. 973-981.
- [4] J. Kroll and J. L. N. Audy, "Mapping Global Software Development Practices for Follow-the-Sun Process.", *International Conference on Global Software Engineering (ICGSE)*, 2012, pp. 164–168.
- [5] J. Kroll, S. I. Hashmi, I. Richardson, and J. L. N. Audy, "A Systematic Literature Review of Best Practices and Challenges in Follow-the-Sun Software Development", In *Global Software Engineering Workshops (ICGSEW)*, 2013, pp. 18-23.
- [6] J. Kroll, R. Prikładnicki, J. L. N. Audy, E. Carmel, and J. Fernandez, "A Feasibility Study of Follow-the-sun Software Development for GSD Projects", *International Conference on Software Engineering (SEKE)*, 2013, Boston, USA.
- [7] J. Kroll, I. Richardson, J. L. N. Audy, and J. Fernandez, "Handoffs Management in Follow-the-Sun Software Projects: A Case Study", In *47th Hawaii International Conference on System Sciences (HICSS)*, Hawaii Island, USA, 2014.
- [8] I. Richardson, V. Casey, F. McCaffrey, J. Burton, and S. Beecham, "A Process Framework for Global Software Engineering Teams", *Information and Software Technology*, 2012, pp. 1175-1191
- [9] E. Hess and J. L. N. Audy, "FTSProc: A Process to Alleviate the Challenges of Projects that Use the Follow-the-Sun Strategy," *International Conference on Global Software Engineering (ICGSE)*, 2012, pp. 56-64.
- [10] N. Denny, S. Mani, R. Sheshu, M. Swaminathan, and J. Samdal, "Hybrid offshoring: Composite personae and evolving collaboration technologies," *Information Resources Management Journal*, 21, 1, 2008, pp. 89–104.
- [11] M. Yap, "Follow the sun: distributed extreme programming development", *Agile Conference Proceedings*, 2005, pp. 218- 224.
- [12] E. Carmel, J. A. Espinosa, and Y. Dubinsky, "Follow the Sun Workflow in Global Software Development", *Journal of Management Information Systems*, 2010, Vol. 27 No. 1, pp. 17 – 38.
- [13] E. Carmel, Y. Dubinsky, A. Espinosa, "Follow The Sun Software Development: New Perspectives, Conceptual Foundation, and Exploratory Field Study", *42nd Hawaii International Conference on System Sciences (HICSS)*, 2009.

# Efficient Points-To Analysis for Partial Call Graph Construction

Zhiyuan Wan<sup>1\*</sup>, Bo Zhou<sup>1†</sup>, Ye Wang<sup>2</sup>, Yuanhong Shen<sup>1</sup>

<sup>1</sup>College of Computer Science and Technology, Zhejiang University, China

<sup>2</sup>School of Computer Science and Information Engineering, Zhejiang Gongshang University, China

Email: {wanzhiyuan, bzhou}@zju.edu.cn, yewang@mail.zjgsu.edu.cn, yhshen@zju.edu.cn

## Abstract

*Many static analysis tools provide whole-program analysis to generate call graphs. However, the whole-program analysis suffers from scalability issue. The increasing size of the libraries exacerbates the issue. For many Web applications, the libraries (e.g. Servlet containers) are even not available for whole-program analysis. We present HyPta, a points-to analysis approach, to construct partial call graphs for Java programs. HyPta extends the standard points-to analysis by establishing a hybrid heap model. Since our approach does not analyze the method bodies of the library classes, the heap model distinguishes between the abstract memory locations in the application and those in the library. HyPta infers the pointer information in the library from the interactions between the application and the library. We implement HyPta based on Spark framework and evaluate it on 14 widely cited Java benchmarks. The evaluation shows that HyPta is faster than Averroes and Spark by a factor of 4.9x and 13.7x, respectively. Meanwhile, it constructs sound and precise partial call graphs.*

## 1. Introduction

A call graph is a crucial prerequisite for most inter-procedural static analyses used in compilers, verification tools and program understanding tools [6]. Many established static analysis tools and frameworks provide whole-program analysis to generate call graphs. However, for modern object-oriented programs, scalability of the whole-program analysis has been a significant hurdle for its practical use in real-world tools. The increasing size of the libraries exacerbates the scalability issue. The libraries include (1) standard libraries (e.g. Java J2SE or C++ STL), (2) domain-specific libraries (e.g. graphics and linear alge-

bra), and (3) extensible frameworks and middleware. The code of the library accounts for a large amount of the code in a program [13]. Moreover, for many Web applications, the libraries (e.g. Servlet containers) are not available for analysis since they depend on the deployment environment. Therefore, partial call graph construction is more practical and scalable than whole-program analysis.

To construct sound and precise partial call graphs, we propose an efficient points-to analysis approach named HyPta. HyPta analyzes the application classes, as well as the signatures, the fields and the class hierarchy of the library classes referenced by the application. Since the memory locations in the library are unknown to our approach, HyPta extends the standard points-to analysis by establishing a hybrid heap model. The heap model distinguishes between the abstract memory locations in the application and those in the library. This leads to a hybrid points-to set for each variable in the program. HyPta also analyzes the interactions between the application and the library to infer the abstract objects created in the library. The hybrid points-to set for the library is used to resolve the library call-back edges and restrict the abstract objects flow back to the application part. We implement HyPta based on the Soot [11] framework and evaluates it on 14 real-world Java programs. The experimental results show that HyPta reduces the execution time of Spark and Averroes by a factor of 13.7x and 4.9x, respectively. Meanwhile, HyPta generates sound and precise call graphs of the application part.

## 2. Proposed Approach

### 2.1. Overview

The points-to analysis is a subset-based, flow- and context-insensitive, and field-sensitive points-to analysis. It builds the call graphs on the fly. The input is a set of Java classes designated as the *application classes*. The application classes may depend on other Java classes designated as the *library classes*. The analysis only analyzes the method signatures, the fields and the class hierarchy of the library

\*This research is sponsored in part by NSFC Program (No.61103032) and National Key Technology R&D Program of the Ministry of Science and Technology of China (No2013BAH01B03).

†Corresponding Author

classes that are referenced by the application classes. Since the memory locations in the library is unknown, we propose a *hybrid heap model* to distinguish between the memory locations in the application and those in the library. We use allocation sites to represent the abstract objects created in the application and use class names to represent those created in the library. The hybrid heap model leads to a hybrid points-to set for each variable, which consists of two parts: the *application points-to set* and the *library points-to set*.

**Definition 1** (Application Points-to Set). *For each pointer variable  $v$  in the program, the application points-to set is defined as  $Pt_A(v)$ , which contains abstract objects that are created in the application and represented by their allocation sites.*

**Definition 2** (Library Points-to Set). *For each pointer variable  $v$  in the program, the library points-to set is defined as  $Pt_L(v)$ , which contains abstract objects that are created in the library and represented by their class names.*

**Sets and Notations.** The analyzed program is composed of a set of classes  $Cls$ . The classes define a set of methods *Method*. The classes declared in the application are denoted by  $Cls_A$  while those declared in the library are denoted by  $Cls_L$ . Each class  $cls \in Cls$  has a set of fields  $fields(cls) = \{f_0, f_1, \dots, f_{q-1}\}$ . Some of the fields are non-static and denoted by a set  $F$  while some are static and denoted by a set  $SF$ . The static fields can be viewed as some sort of global variables. An artificial field  $[]$  is introduced to model array cells. Each method  $m \in Method$  has a list of parameters, a set of local variables and a body. The set of variables in the application is represented by  $V$ . If a method  $m$  has  $k$  parameters, the parameters are in order  $p_0, p_1, \dots, p_{k-1}$ . For non-static methods,  $p_0$  is *this* parameter. In the Java programming language, parameters are a special case of local variables, i.e.,  $p_i \in V$ . Let  $O$  be the set of all abstract objects created in the program. We denote by  $O_A$  the set of all the abstract objects created in the application and  $O_L$  the set of those created in the library. The body of the method  $m$  contains a list of statements from the set *Statement*. For brevity, we discuss only a set of elementary statements that manipulate pointers shown in Table 1.  $v_i \in V$  denotes a local variable or a formal parameter (including *this*) in the program. Since our analysis is flow-insensitive, there is no need to consider control-flow statements such as branches and loops. CALL and RETURN statements handle the interprocedural control-flow. Each CALL statement corresponds to a call site. We denote by  $C$  the set of call sites in the application classes.

## 2.2. Program Representation

Our points-to analysis builds a *pointer assignment graph* (PAG) [7] to represent the program being analyzed. The

Table 1: Pointer assignment graph construction.

Type	Elementary Statement	PAG Entity
ASSIGN	$v_1 = v_2$	$v_2 \xrightarrow{assign} v_1$
NEW	$o : v = new X$	$o \xrightarrow{new} v$
NEW ARRAY	$o : v = new X[k]$	$o \xrightarrow{newarray} v$
INSTANCE STORE	$v_1.f = v_2$	$v_2 \xrightarrow{store} v_1.f$
STATIC STORE	$X.f = v$	$v \xrightarrow{assign} X.f$
ARRAY STORE	$v_1[i] = v_2$	$v_2 \xrightarrow{store} v_1.[i]$
INSTANCE LOAD	$v_2 = v_1.f$	$v_1.f \xrightarrow{load} v_2$
STATIC LOAD	$v = X.f$	$X.f \xrightarrow{assign} v$
ARRAY LOAD	$v_2 = v_1[i]$	$v_1.[i] \xrightarrow{load} v_2$
INSTANCE CALL	$v_R = v_0.m(v_1, \dots, v_j)$	$v_0 \xrightarrow{assign} this,$ $v_i \xrightarrow{assign} p_i \wedge 1 \leq i \leq j,$ $ret_m \xrightarrow{assign} v_R$
STATIC CALL	$v_R = X.m(v_1, \dots, v_j)$	$v_i \xrightarrow{assign} p_i \wedge 1 \leq i \leq j,$ $ret_m \xrightarrow{assign} v_R$
RETURN	<i>return v</i>	$v \xrightarrow{assign} ret_m$

PAG consists of three types of nodes: allocation nodes, variable nodes and field dereference nodes. Allocation nodes represent allocation sites and model the heap locations in the application. Variable nodes represent local variables, method parameters, return values and static fields. Field dereference nodes represent field access expressions and are parameterized by corresponding variable nodes that are dereferenced by the field access. Note that the allocation nodes in the PAG form the set of abstract objects  $O_A$ . The nodes in the PAG are connected with four types of edges reflecting the pointer flow: *new/newarray*, *assign*, *store*, and *load*.

We construct the PAG by associating elementary statements with different PAG entities. Table 1 shows the different elementary statements and corresponding PAG entities. Note that  $v, v_i, v_R, C.f$  and *this* denote the variable nodes,  $o$  denotes the allocation node, and  $v_i.f$  denotes field dereference nodes. In the handling of interprocedural flow, for each method  $m$ , the PAG has a node for each of  $m$ 's formal parameters and a special  $ret_m$  node for  $m$ 's return value. At each call site  $c$  of  $m$ , we add an *assign* edge from each actual parameter to its formal parameter, and an *assign* edge from the  $ret_m$  node to the caller's variable. Each node  $n$  in the PAG is associated with a points-to set, which contains the abstract objects that may be referenced by the node. The abstract objects in the points-to sets are propagated along the PAG edges.

## 2.3. Library Model

Since our analysis does not take into account the method bodies of the library classes, we propose a model to represent the library as below:

**Variables.** Let *Library* be an arbitrary variable in the library.

**Methods.** The analysis is defined in terms of  $m_L$  for an arbitrary method in the library.

**Points-to set.** A single points-to set  $Pt(Library)$  is constructed for the library, which consists of two parts. One is  $Pt_L(Library)$ , which contains  $o \in O_L$  that is referenced by the library. Since we assume that the library methods can create and reference the objects of any classes declared in the library,  $Pt_L(Library)$  contains the class names of all library classes implicitly. The other is  $Pt_A(Library)$ , which contains  $o \in O_A$  that is referenced by the library.

**Call sites.**  $c_L$  represents an arbitrary call site in the library and can invoke: 1) any visible method declared in any library class; 2) the method  $m$  declared in an application class  $cls$ , only if  $m$  is non-static and overrides a method originally declared in a library class, and there exists  $o \in Pt_A(Library)$  where  $StaticType(o)$  is  $cls$  or a subclass of  $cls$ . We use the notation  $StaticType(o)$  for the static type of object  $o$ .

The interaction between the application and the library determines the abstract objects that may escape from the application scope to the library, and vice versa. We infer the set of abstract objects  $O_L$  created in the library by analyzing the interaction. Furthermore, the library points-to sets of the variables are updated according to the sequence of rules that correspond to the types of elementary statements as below.

**Rule 1.** for  $v_1 = v_2.f$ , where the instance field  $f$  is declared in  $cls \in Cls_L$ :

$StaticType(f) \in O_L$ ;  $StaticType(f) \in Pt_L(v_1)$ .

The notation  $StaticType(v)$  denotes the static type of the variable  $v$ .

**Rule 2.** for  $v = X.f$ , where  $X \in Cls_L$ :

$StaticType(f) \in O_L$ ;  $StaticType(f) \in Pt_L(v)$ .

Consider a call to a static method  $m$  declared in a library class; the call occurs in an application class.

**Rule 3.** for  $w = X.m(v_1, \dots, v_k)$ , where  $X \in Cls_L$ :

$StaticType(ret_m) \in O_L$ ;  $StaticType(ret_m) \in Pt_L(w)$ .

In this rule,  $v_i$  is the argument of the call site. We use  $ret_m$  to denote the return value of  $X.m$ .

Consider an instance call  $w = v_0.m(v_1, \dots, v_k)$  occurring in a reachable method in the application, where  $m$  is a method declared in the library.

**Rule 4.** for  $w = v_0.m(v_1, \dots, v_k)$ , where  $v_0 \in V$ :

$StaticType(ret_m) \in O_L$ ;  $StaticType(ret_m) \in Pt_L(w)$ .

Consider an instance call occurring in the library with the target method  $m$ , which is also the target of a library call back edge. The method is originally declared in the library and overridden by the application.

**Rule 5.** for  $Library.m(v_1, \dots, v_k)$  :

$StaticType(this) \in O_L$ ;  $StaticType(this) \in Pt_L(this)$ ;

$StaticType(v_i) \in O_L$ ;  $StaticType(v_i) \in Pt_L(v_i) \wedge 1 \leq i \leq k$ .

## 2.4. Virtual Call Handling

To soundly and precisely handle virtual calls, we compute targets of virtual calls and construct the call graphs *on the fly*, i.e., as the relevant points-to sets of call site receivers are computed.

**Definition 3** (CallGraph). We denote by *CallGraph* the partial call graph of the application part. The *CallGraph* is comprised of a finite set of call edges. Each call edge connects a call site, which is a statement in some method, to a method that may be invoked from that call site. The call edge is represented by a 3-tuple  $(m_1, m_2, c)$  where  $m_1, m_2 \in \{m_L\} \cup Method$  and  $c \in \{c_L\} \cup C$ .

Consider a virtual call  $v.m(\dots)$  occurring in the method  $M$  declared in the application, the analysis resolves the call site  $c \in C$  on the receiver  $v$  by using the points-to set  $Pt_A(v)$  and  $Pt_L(v)$  according to the following two rules:

**Rule 6.** for each object  $o \in Pt_A(v)$ ,  $StaticLookup(cls, m) = m'$  where  $cls = StaticType(o) : (M, m', c) \in CallGraph$ .

**Rule 7.** for each object  $cls \in Pt_L(v)$ ,  $StaticLookup(cls, m) = m' : (M, m', c) \in CallGraph$ .

We use the notation  $StaticType(o)$  to denote the static type of the object  $o$ , and the notation  $StaticLookup(cls, m)$  to denote the definition (if any) of a method with name  $m$  that one finds when starting a static method lookup in the class  $cls$ .

For the virtual calls in the library, we define the following rule to compute library call back edges with  $Pt_A(Library)$ :

**Rule 8.** for each object  $o \in Pt_A(Library)$ ,  $m$  is declared in an application class  $cls$  where  $cls \in \{StaticType(o)\} \cup SuperTypes(StaticType(o))$  and overrides a library method:  $(m_L, m, c_L) \in CallGraph$ .

$SuperTypes(cls)$  denotes the supertypes of  $cls$ .

## 3. Evaluation

We evaluate HyPta by comparing its performance and generated call graphs with those of Spark and Averroes. Spark [7] implements the whole-program analysis to construct call graphs. Averroes [2] generates a placeholder library to enable Spark and other whole-program analysis frameworks to construct partial call graphs.<sup>1</sup>

**Experimental Setting.** The experiment is executed on a machine with Intel Core 2 2.13GHz p7450 CPU and 2 GB of RAM. We ran the experiment on the DaCapo benchmark program v. 2006-10-MR2 [5] and the SPEC jvm 98 benchmark program [8]. We use the same settings as earlier published work [1, 2]. The experiment analyzed the Java standard library from jre 1.4.2.11. The *fop* and *eclipse* benchmarks have been excluded because the incompleteness of referenced classes. The *ython* benchmark is also

<sup>1</sup>See <http://plg.uwaterloo.ca/~karim/projects/averroes>.

Table 2: Information about the benchmarks.

Program	# Application Classes	# Application Methods
antlr	202	1330
bloat	334	2526
chart	489	1042
hsqldb	389	3213
luindex	324	501
lusearch	324	896
pmd	527	1936
xalan	534	3107
compress	12	32
db	3	31
jack	56	292
javac	176	1105
jess	5	18
raytrace	25	159

excluded because sophisticated reflection forms are heavily used, which makes static analysis impractical. The detailed information of the benchmarks are presented in Table 2.

**Implementation.** We implemented HyPta on top of the Soot framework [11], version 2.5.0, and bootstrapped by Spark [7]. We defined the application part of a program as the analysis scope for Soot. We extend Spark by creating the hybrid points-to sets, propagating the objects in the hybrid points-to sets and resolving call sites using the points-to sets according to different rules.

### 3.1. Performance

Figure 1 shows the running time required for call graph construction by HyPta, Spark and Averroes. HyPta scales well taking under 32 seconds for each benchmark. We found HyPta is faster than Averroes by a factor of 4.9x on average (min: 1.9x, max: 33.9x, geometric mean: 4.9x) and faster than Spark by a factor of 13.7x on average (min: 3.3x, max: 131.2x, geometric mean: 13.7x). HyPta improves the performance of Spark significantly. This improvement is achieved by generating call graphs without analyzing the method bodies of the library classes. The running time for Averroes breaks into two components: *pre-analysis time* (denoted by *Averroes<sup>Placeholder</sup>* in Figure 1) is the time required for Averroes to generate placeholder library; and *analysis time* (denoted by *Averroes<sup>Analysis</sup>* in Figure 1) is the time required for Spark to generate call graphs with the Averroes placeholder library. The slowdown of Averroes is due to the generation of the placeholder library.

### 3.2. Call Graph Soundness

We evaluated the soundness of the static call graphs by counting the call edges that are present in the dynamic call graphs collected by \*J [4] but missing from the static call graphs generated by static tools. The dynamic call graphs are observed during the dynamic execution of the benchmarks. The results are shown in Table 3. The row

“Dynamic” shows the number of call edges in the application part of the benchmarks. The rows “Dynamic\HyPta”, “Dynamic\Averroes” and “Dynamic\Spark” show that the number of dynamic call edges that are missing in the static call graphs generated by HyPta, Spark and Averroes. The rows “Dynamic\HyPta” and “Dynamic\Averroes” show only two library call graph edges in *lusearch* and *xalan* are missing from the call graphs of all the benchmarks generated by HyPta and Averroes. In *lusearch*, a *NullPointerException* is thrown at runtime and a call to the constructor of this exception is recorded in the dynamic call graph. In *xalan*, a call to *java.lang.ref.Finalizer.register* is missing. The reason is that both HyPta and Averroes do not model the complete runtime behavior of the Java virtual machine. Furthermore, the row “Dynamic\Spark” shows that Spark is missing a significant number of call edges for the benchmarks that make heavy use of reflection. This is because Spark does not handle reflection while HyPta and Averroes make use of the information of reflection collected by TamiFlex [3].

### 3.3. Call Graph Precision

We compared the number of call edges in the call graphs generated by HyPta, Averroes and Spark with respect to three categories of call edges. Spark implements a whole-program analysis call graph construction algorithm that analyzes the library thoroughly, whereas HyPta and Averroes construct partial call graphs without analyzing the method bodies of the library classes. Therefore, we say that HyPta or Averroes is precise when the generated call graphs are identical to those generated by Spark. In addition, since we found some dynamic call edges are missing from the call graphs generated by HyPta, Averroes and Spark, we add the missing edges to the static call graphs first to avoid confounding due to the differences in soundness. The rows “HyPta\Spark” and “Averroes\Spark” in Table 4, 5 and 6 represent the numbers of call edges in the static call graphs generated by HyPta and Averroes that are missing in the call graphs generated by Spark, and are not present in the dynamic call graphs.

**Application Call Graph Edges.** Table 4 shows that HyPta generates precise call graphs with respect to application call graph edges for *luindex*, *compress*, *db*, *jess* and *raytrace*. For all benchmarks, HyPta generates an average of 343 and a median of 22 extra application call graph edges (min: 0, max: 1821, average: 343, median: 22) in comparison with Spark. The median is negligible as it represents 0.49% of the median number of application call graph edges generated by Spark. The *bloat*, *hsqldb* and *xalan* benchmarks have the highest frequency of imprecise call edges. This is due to the fact that these benchmarks have large subsystems that implement the library interfaces, *java.util.\**, JDBC and XML, respectively. Moreover, the average is smaller than

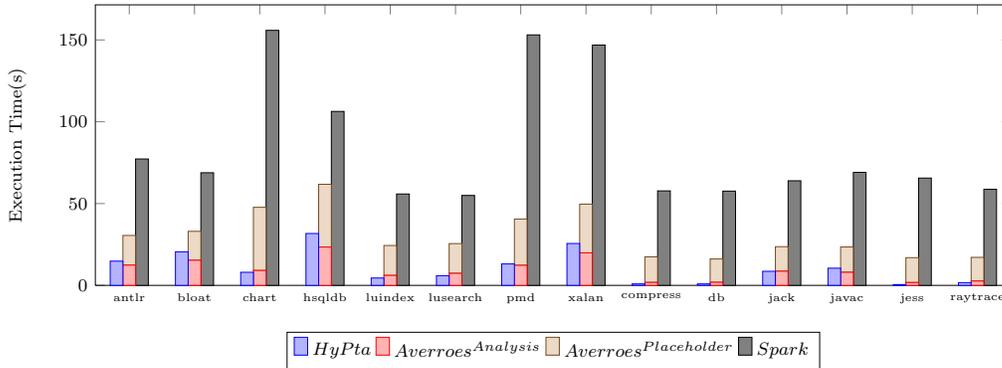


Figure 1: Comparing the time taken by the analysis in HyPta, Averroes and Spark for each benchmark program.

Table 3: Comparing the soundness of HyPta, Averroes and Spark.

	antlr	bloat	chart	hsqldb	luindex	lusearch	pmd	xalan	compress	db	jack	javac	jess	raytrace
Dynamic	3449	4257	657	1627	726	539	2087	2953	43	54	596	2538	13	330
Dynamic\HyPta	0	0	0	0	0	1	0	1	0	0	0	0	0	0
Dynamic\Averroes	0	0	0	0	0	1	0	1	0	0	0	0	0	0
Dynamic\Spark	0	0	0	61	4	185	3	96	0	0	0	0	0	0

the average of 414 and the median is slightly smaller than the median of 23 (min: 0, max: 2310, average: 414, median: 23) for Averroes, which suggests HyPta generates more precise call graphs than Averroes with respect to application call graph edges.

**Library Call Graph Edges.** Table 5 shows that HyPta generates precise call graphs with respect to library call graph edges for *antlr*, *luindex*, *compress* and *raytrace*. For all benchmarks, HyPta generates an average of 20 and a median of 2 extra library call graph edges (min: 0, max: 112, average: 20, median: 2) in comparison with Spark. The median is negligible as it represents 0.55% of the median number of library call graph edges generated by Spark. Since the average is smaller than the average of 22 and the median is smaller than the median of 3 (min: 0, max: 116, average: 22, median: 2.5) for Averroes, HyPta generates more precise call graph than Averroes with respect to library call graph edges.

**Library Call Back Edges.** Table 6 shows the results of comparing the library call back edges. For all benchmarks, HyPta generates an average of 85 and a median of 23 extra application call back edges (min: 1, max: 645, average: 85, median: 22.5) in comparison with Spark. The average is larger than the average of 69 and the median is larger than the median of 5 (min: 0, max: 605, average: 69, median: 4.5) for Averroes. The median represents 31.69% of the median number of application call graph edges generated by Spark. We investigate the causes of imprecise library call back edges generated by HyPta. One cause is that HyPta models the implicit calls to *<clinit>* as library call back edges under 3 conditions: a) when a static method is invoked, b) when a static field is accessed, and c) when an object or a array of objects is initialized. Whereas Spark

models the implicit calls to *<clinit>* as application call graph edges. Therefore, most of these extra library call back edges are with the static initialization blocks *<clinit>* as their targets. Another cause is that HyPta implements partial call graph construction algorithm that makes assumption about the objects referenced in the library. HyPta assumes the objects passed to the library through calls to library methods would be accessible to the library. However, some of the library methods would not leak a reference to other library code. This assumption in HyPta introduces imprecision into the points-to sets inside the library and results in spurious library call back edges. Furthermore, these objects may pollute the points-to sets of the variables in the application and result in imprecise call edges of other categories.

The results show that HyPta achieves higher precision than Averroes, especially on *bloat* and *xalan* benchmarks, with respect to the application call graph edges. The target methods of the extra call edges are declared in a relatively small number of application classes. These classes implement the interfaces or extend the abstract classes declared in the library. For instance, for *bloat*, the targets of the extra edges are declared in the classes *edu.purdue.cs.bloat.util.ImmutableIterator* and *edu.purdue.cs.bloat.codegen.LiveNews\$1*, which implement the library interface *java.util.Iterator*. This is due to the construction process of library points-to set, which determines the number of targets when resolving call sites.

## 4. Related Work

We discuss the most related work that designs efficient approaches to construct partial call graphs. The Wala framework [12] ignores the effect of the library code. This

Table 4: Comparing the precision of HyPta with respect to *Application Call Graph Edges*.

	antlr	bloat	chart	hsqldb	luindex	lusearch	pmd	xalan	compress	db	jack	javac	jess	raytrace
HyPta	6278	14639	1863	8921	901	2019	4576	10945	40	47	653	8359	6	400
Averroes	6288	15157	1921	8970	928	2105	4702	11299	40	48	653	8350	6	400
Spark	6299	13419	6732	7792	1412	2724	6893	13020	40	47	646	8519	6	400
HyPta\Spark	32	1821	12	1317	0	250	160	1159	0	0	7	45	0	0
Averroes\Spark	14	2310	32	1328	5	253	270	1536	0	1	7	45	0	0

Table 5: Comparing the precision of HyPta with respect to *Library Call Graph Edges*.

	antlr	bloat	chart	hsqldb	luindex	lusearch	pmd	xalan	compress	db	jack	javac	jess	raytrace
HyPta	648	870	554	949	241	345	430	1013	13	24	107	314	7	34
Averroes	650	880	604	960	246	348	433	1032	13	24	107	314	7	34
Spark	661	885	1060	869	336	392	797	1055	13	23	97	317	6	34
HyPta\Spark	0	10	3	112	0	29	2	108	0	1	10	2	1	0
Averroes\Spark	0	12	24	115	0	29	3	116	0	1	10	2	1	0

Table 6: Comparing the precision of HyPta with respect to *Library Call Back Edges*.

	antlr	bloat	chart	hsqldb	luindex	lusearch	pmd	xalan	compress	db	jack	javac	jess	raytrace
HyPta	62	176	136	690	50	71	118	701	1	3	12	36	12	3
Averroes	52	203	149	655	37	65	118	662	0	3	10	29	12	1
Spark	73	223	490	69	132	146	135	464	10	12	10	41	64	10
HyPta\Spark	21	56	48	645	24	35	28	310	1	1	12	11	1	3
Averroes\Spark	3	26	9	605	2	6	23	272	0	1	10	2	1	1

leads to incomplete call graphs because the pointer information is not complete both in the application and in the library. Yan et.al.’s work [13] creates procedure summaries for library methods and uses them when analyzing a specific client program. However, the creation and use of summary face a number of technical challenges in terms of analysis abstraction, algorithms and infrastructure support. Tip and Palsberg [10] associate a single points-to set  $S_E$  with the the library and incorporate refinements to make less conservative assumptions about the library. Rountev et al. [9] create a placeholder, a *main* method, to represent the unanalyzed part. The *main* method contains a variety of placeholder statements to represent all possible behaviors of the unanalyzed code. This may alter the original behaviors of the application code and result in imprecise call graphs of the application part. Averroes [2] generates a placeholder library and enables the whole-program analysis frameworks to construct partial call graphs efficiently. It builds on the separate compilation assumption proposed in Ali and Lhoták’s previous work [1]. The placeholder library implements the specific constraints derived from the separate compilation assumption that overapproximates the behaviors of the library.

## 5. Conclusions and Future Work

We propose an efficient points-to analysis approach to construct partial call graphs for Java applications. Our approach 1) uses a hybrid heap model to distinguish between the abstract memory locations in the application and those in the library; 2) models the interaction between the application and the library to infer the abstract objects created in the library. We implement the analysis based on the Spark

framework and evaluated the analysis by comparing it with Spark and Averroes. The results show that our approach provides an efficient way to construct sound and precise partial call graphs of the application part. In the future, we plan to extend the work to handle reflection and static initialization blocks more precisely.

## References

- [1] K. Ali and O. Lhoták. Application-only call graph construction. In *ECOOP*, 2012.
- [2] K. Ali and O. Lhoták. Averroes: Whole-program analysis without the whole program. In *ECOOP*, 2013.
- [3] E. Bodden, A. Sewe, J. Sinschek, H. Oueslati, and M. Mezini. Taming reflection: Aiding static analysis in the presence of reflection and custom class loaders. In *ICSE*, 2011.
- [4] B. Dufour, L. Hendren, and C. Verbrugge. \*j: a tool for dynamic analysis of java programs. In *OOPSLA*, 2003.
- [5] B. et al. The dacapo benchmarks: Java benchmarking development and analysis. In *OOPSLA*, 2006.
- [6] O. Lhoták et al. Comparing call graphs. In *PASTE*, 2007.
- [7] O. Lhoták and L. Hendren. Scaling java points-to analysis using spark. In *CC*, 2003.
- [8] V. Narayanan. Spec jvm98 benchmarks.
- [9] A. Rountev, A. Milanova, and B. G. Ryder. Fragment class analysis for testing of polymorphism in java software. *TSE*, 2004.
- [10] F. Tip and J. Palsberg. Scalable propagation-based call graph construction algorithms. In *OOPSLA*, 2000.
- [11] R. Vallée-Rai, E. Gagnon, L. Hendren, P. Lam, P. Pominville, and V. Sundaresan. Optimizing java bytecode using the soot framework: Is it feasible? In *CC*, 2000.
- [12] T. Watson. Libraries for analysis (wala).
- [13] D. Yan, G. Xu, and A. Rountev. Rethinking soot for summary-based whole-program analysis. In *SOAP*, 2012.

# Text-Based Clustering and Analysis of Intelligent Argumentation Data

Eric Barnes

Department of Computer Science  
Missouri University of Science and Technology  
Rolla, United States  
ecbyt7@mst.edu

Xiaoqing (Frank) Liu

Department of Computer Science  
Missouri University of Science and Technology  
Rolla, United States  
fliu@mst.edu

**Abstract**—Argumentation is a method by which stakeholders exchange their viewpoints and rationale in the form of arguments in an organized manner in order to conduct collaborative decision making. Many online systems have been implemented in order to provide geographically distributed stakeholders with a structured method of argumentation. However, as these systems collect large amounts of arguments; it can be difficult to readily assess the major concerns which drive the discussion. This work presents a method for clustering and classifying a set of arguments, collected through an online argumentation tool, in order to model major concerns in an argumentation process. These clusters are further analyzed to provide a qualitative understanding of the influence they have on the decision making process.

**Index Terms**—Argumentation, Collaborative Decision Making, Text Clustering,

## I. INTRODUCTION

In the Software Engineering process, there are often many points at which a selection must be made from a number of mutually exclusive alternatives. If this decision is to be made collaboratively, each stakeholder will have their own idea of which alternative is ideal with respect to their own needs and experience, and will disagree with one another about which alternative to take. When decision making proceeds without an understanding of stakeholder conflict, decisions can be made which do not adequately address the concerns of this group.

The process of argumentation in part addresses this by providing stakeholders with an organized platform for exchanging their knowledge and the rationale behind their preferences in the form of arguments. With the structured representation of stakeholder rationale, decision makers are better equipped to select alternatives which represent the collective needs and expertise of the group. Online argumentation tools further enhance this process by allowing geographically distributed stakeholders to exchange arguments asynchronously, with such arguments collected and recorded in a logical structure. Such structures reflect how arguments relate to one another and provide context for each argument made. This can be reviewed over the course of the decision making process in order to analyze the rationale behind each

viewpoint, and attempt to determine the overall consensus before making a selection.

However, as the number of arguments increases, it becomes increasingly difficult to maintain an overall understanding of the argument in progress, as each argument must be read individually for their content to be understood. Many of these arguments will refer to similar subjects or concerns. Concerns which manifest in a large amount of arguments represent influential topics of conversation and debate. If a stakeholder had some idea of what these major concerns were, they would have a quick access to the broad themes of the arguments in progress. However, arguments which relate to the same subject are not necessarily organized as such in the argument structure, as references particular concern can arise in response to any number of threads of debate. This is particularly the true when the number of arguments is large, as stakeholders themselves lose sight of the overall discussion.

It is not practical to demand that participants form logical argument structures which are perfectly organized according to strict topics. They should be free to use all relevant information and voice their concerns in the course of debate. Because computers have no issues parsing large numbers of arguments, it is more reasonable for the argumentation system itself to perform analysis on logically distributed arguments to determine what the major concepts involved in discussion are, organize arguments according to these concepts when applicable, and present this information to participants.

This work proposes the use of text clustering techniques to group arguments according to the concerns they represent. This provides stakeholders with an view of which topics are most strongly represented in the form of arguments. The system possesses an inference engine which can calculate the degree of support held by the available positions or alternatives by relating the weights of their direct and indirect responding arguments.

Once text clusters are formed to describe major argument concerns, additional analysis can be performed using the logical argument structure and the weights of each argument to determine what effect the clustered arguments have on discussion.

## II. RELATED WORKS

### A. Argumentation Systems

Philosopher Stephen Toulmin proposed a model of argumentation which has formed the basis for a wide scope of computer-based argumentation systems [1]. The first such method was gIBIS, which displayed arguments, issues, and positions in a unified graph [2]. HERMES is a decision support tool which describes arguments and evidences as nodes in a logical hierarchy [3]. Evidences in this system are specific arguments which attach themselves to conflicting argument in order to provide one with more concrete proof. The system then can use the weights of all arguments to find the most favored alternative. Chenn-Junn Huang developed an argumentation system which analyses arguments, assesses their quality, and sends feedback to users depending on their calculated skill level [4]. Collaboratorium [5] is another argumentation system which allows stakeholders to exchange rationale and ideas by posting them to a visual representation called the argument map.

In all of these systems, all arguments must be read individually in order to parse their contents. This is feasible for following individual threads of discussion, but it is not practical to attempt to read an entire argumentation structure in order to gain an overall understanding of stakeholder concern. This work is unique in that it uses text clustering to group arguments according to important discussion topics, and then using the argumentation structure to provide additional context to these argument groups.

### III. OVERVIEW OF TECHNIQUE

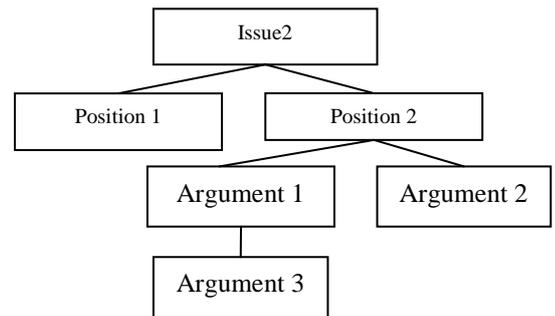
This process aims to provide both an overview of the important topics which arise during an argumentation process. Important in this sense refers to concepts which appear in a relatively large number of arguments. Arguments are collected via our online intelligent argumentation system including both their text component, weight, and logical position in the argument hierarchy.

These arguments are then clustered and classified based on their text components, the labels for these clusters denote frequently referenced concepts and terms, with labels describing the largest clusters denoting the more important terms. Text clustering is carried out using *Lingo* algorithm [6], a process developed for clustering search web search results based on the textual content of their snippets. *Lingo* was chosen because both arguments and snippets are relatively small compared to other types of documents. With the information provided by the argument clusters, and the analysis thereof, stakeholders and decision makers can have a better understanding of the broad topics of argumentation at a glance.

### IV. ONLINE INTELLIGENT ARGUMENTATION SYSTEM

An online argumentation system was developed by Liu et. al. [7] based on Toulmin's model of argumentation. The system allows stakeholders to exchange arguments in an asynchronous manner, and organizes said arguments in a logical tree structure. In this system, a set of positions are

available as a resolution to a particular issue. Argumentation is carried out to allow users to debate over which position is ideal for resolving their respective, with arguments being attached both directly to positions as well to as other as logical children. In addition, the system allows stakeholders to attach weight to their arguments, denoting their level of support to that arguments logical parent. This system has functionality to reduce weights to determine how they relate to the available positions in order to calculate their overall *favorability factor*, which expresses the level of support each has from the collected arguments. While this favorability factor can provide a quantification of stakeholder preference, decision makers may want more information to understand why the preferences are so.



**Figure 1. Sample Argumentation Tree**

An argumentation system has been implemented [1] which organizes arguments in a logical tree structure, and also allows stakeholders to attach weights to their arguments, denoting the degree of support or attack they are responding with. The argumentation tree consists of the following elements:

**Issue** - This is a design problem or complication that a particular project faces. The solution to this issue is to be found through collaborative decision making supported by the intelligent argumentation system. A project may have any number of potential issues, and the scope of such issues can include any decision that must be made within the software development process.

**Position** - This is a potential solution to a given issue. In the context of software engineering it could be a component to add to the system, a design decision, or any other business decision problem which needs to be resolved for the project.

**Argument** - This is a concise statement partially giving the rationale behind a stakeholder's preference. Arguments attached to positions contain a rationale which partially explains a stakeholder's support or lack of support for that position. Arguments attached to other arguments attempt to reinforce or contradict the statement made by the other participant with additional rationale. An argument is either supporting or attacking depending on its given weight: a number ranging from -1 to 1. Negative weights indicate attack, while positive weights indicate support. The magnitude of the weight reflects the degree of support or attack.

These elements compose a structure which will be referred to as the argument tree. A representation of a sample argument tree can be seen in Figure 1. In this figure, squares correspond to nodes in the tree, and edges indicate a parent/child relationship. In Figure 1, Argument 3 is a direct response to

Argument 1, but is also an indirect response to Position 2 at its root. The argumentation framework is capable of inferring the indirect relationship of all arguments to the corresponding position at their root. The normalized aggregate of all reduced weights is defined as the *favorability factor* of that position. This process is described fully in previous work.

## V. OPERATION OF LINGO CLUSTERING

While the complete workings of lingo are outside the bounds of this work, a brief overview of the algorithm is given as the following four steps:

**Pre-processing:** Text inputs are pre-processed in the following ways. Stemming is carried out to find the semantic representation of inflected words. Stop words are ignored to remove commonly used words which have no meaning on their own, such as conjunctions, articles, etc. Finally, text is divided into words and sentences through the use of text-segmentation heuristics.

**Phrase extraction:** A modified version of SHOC's phrase discovery algorithm is used to extract potential cluster labels from the input documents. To be a potential label, a phrase must satisfy a set of prescribed conditions such as frequency of use, etc.

**Cluster-label induction:** Matrix decomposition is used to summarize cluster labels via singular value decomposition. The relevance of different potential labels are compared for each document, and the most relevant labels for the group are selected as cluster labels

**Cluster-content allocation:** Queries are performed on each document for each cluster label, if the similarity between the document and the label exceeds a specified threshold, the document is added to the cluster..

## VI. ANALYSIS OF ARGUMENT CLUSTERS

Because the logical placement and weight of arguments in the argumentation tree is integral to their meaning, additional analysis can be performed based on this in order to properly understand concern clusters..

### A. Cluster-specific Favorability Factor

In the online argumentation system, the weight of each argument can be until it relates directly to some position. As the overall favorability of each position is calculated as the normalized summation of all reduced arguments responding to it, cluster specific favorability can be similarly derived by only considering arguments from a particular cluster.

Just as the favorability factor represents which alternatives are more strongly supported by all arguments, cluster-specific favorability factors represent the degree to which arguments pertaining to each concern support or attack the given alternatives. This can provide a more nuanced view of favorability, describing not just overall support, but relating that degree of support to specific topics, giving a potential for exploring the strengths and weakness of each position.

### B. Unique contributors

While the cluster size represents the total number of arguments in a cluster, it may be useful to know how many

stakeholders have expressed an argument relating to each concern. For instance, a cluster may contain a large number of arguments, but these arguments are only posted by a small portion of the stakeholder population. Analysis of a clusters amount of unique contributors can be used to help determine how widespread a particular concern is among stakeholders. Because each participant includes their unique id as part of their argument, it is relatively simple to look at an argument cluster and count the number of unique contributors.

### C. Response Metrics

Arguments which have a large amount of responding arguments, both direct and indirect, have a high amount of influence on the argumentation process, as their very existence provokes additional debate and discussion. Due to the structure of the argumentation tree, the direct and indirect responses to an argument can be found by isolating a sub-tree in the argumentation system which is rooted at the argument in question. The calculation of the reduced weight for all of these sub-arguments must include the weight of the influential node. Additionally, the aggregate weights of these responses are also vital information, as an argument which receives a strongly positive response indicates an opinion shared by multiple stakeholders, while a strongly negative one indicates otherwise. The process for quantifying an individual arguments *Reaction weight*, is identical to finding the favorability of a position: take the normalized sum of all responses reduced to the level of a direct response.

This principle also applies to argument clusters Their influence is not limited to their own aggregate favorability factors and size, the level and nature of the response that the cluster provokes should also be considered,. This is expressed through two additional metrics. The *response size* and *cluster reaction* are both found by tracing each argument which is not already in the cluster upwards until either a position or argument in the system is encountered. If the trace encounters an argument within the cluster, then it is a response and the response size is incremented. Additionally, the weight of the response is reduced until it is directly responding to the in-cluster argument, and this amount is added to the *cluster reaction*. The total reaction for a cluster is taken as the summation of all reduced responding argument weights normalized with respect to the total response size, giving stakeholders an idea of scope and nature of the response provoked by a cluster.

## VII. EMPIRICAL STUDY

An empirical study was carried out to simulate the use of the argumentation system for software engineering decision making. 24 participants were to take the role of stakeholders in a medium scale software development firm which had to decide on a metrics system to employ: no metrics system, represented as position 1, a lightweight metrics program represented as position 2, and a comprehensive metrics program, represented as position 3. 314 arguments in total were collected. Argument clustering was performed via Lingo with a base cluster count of 15, resulting in 26 total clusters, with 107 arguments listed as "Other Topics".

While the base cluster count could be increased to group additional arguments, it was kept at 15 to focus on the more influential clusters and obtain meaningful labels. The purpose of this method is not to necessarily categorize *all* arguments, but to locate and analyze the important topics of discussion. The top-10 most heavily populated clusters are given in Table 1, with the first column giving the cluster label as derived by lingo, column 2 giving the size of the cluster, columns 2,3,4 giving the cluster specific favorability factors in response to P1, P2, and P3, column 5 giving the total number of responses to the cluster, column 6 giving the aggregate reaction weight, and column 7 giving the number of contributors to the cluster.

Label	Sz.	P1	P2	P3	Res.	Reac.	Cont.
Software Products	31	-0.28	0.61	-0.06	10	-0.25	14
Products Developed	30	-0.42	0.65	-0.20	9	-0.16	12
Metrics can Measure	29	-0.25	0.39	-0.18	10	-0.31	12
New Technologies	28	-0.30	0.49	-0.28	11	-0.03	11
Time Money	27	-0.80	0.81	-0.85	16	-0.14	8
Reducing Requirements	22	-0.85	0.52	-0.40	16	0.15	9
Opposing Argument	18	-0.107	0.51	-0.59	3	-0.1	2
Revenue Used for Quality Assurance	17	-0.304	0.42	0.06	5	-0.12	7
Loss in Revenue	16	-0.49	0.56	-1.0	11	-0.07	10
Measure Productivity	16	-0.12	0.78	0.0	2	-0.09	8

**Table 1: Analysis of 10 largest argument clusters**

### A. Qualitative Interpretation of Results

#### 1) Labels

While the top two clusters are the most general, both relating to Software product development, subsequent clusters provide more interesting information, particularly that clusters are concerned with topics such as New Technologies, Time, Money, Reduction of Requirements etc. Each cluster could be browsed to examine their associated arguments directly for a better understanding of their meaning. For instance the Reduction of Requirements arguments were generally concerned with the business habit of reducing requirements to meet deadlines.

#### B. Cluster Specific Favorability Factors

All clusters collected favored position 2 over both 1 and 3, as is consistent with the overall favorability factor calculated by the system. However the degree of this favorability varies somewhat from cluster to cluster. For instance, the Time Money cluster came down particularly harshly on both P1 and P3 and was the most supportive of P2. Conversely, the

Revenue used for QA cluster was slightly supportive of P3 and only moderately supportive of P2.

### C. Influence Analysis through Contributors and Response

For most clusters, the collective reaction was slightly to moderately negative. One exception was Reducing Requirements, which was slightly supported by 16 responses, and was tied with Time Money for the largest amount of responses. Although it was not the most heavily populated cluster in terms of arguments total or contributors, the reaction it provokes makes it a particularly influential cluster.

Conversely, Opposing Argument, despite its high argument count, only has 2 contributors, and a negligible degree of response. Interpreting these metrics, it is clear that this is not an influential concept in relation to the others.

## VIII. CONCLUSIONS AND FUTURE WORK

Navigating a logical argumentation structure becomes increasingly difficult as the number of arguments increases, making the maintenance of the overall discussion challenging. By clustering arguments based on their textual component, it is possible to identify concerns according to clusters formed by these common terms and themes. Furthermore, due to the weight and logical organization of arguments, it is possible to perform additional analysis to determine how these concerns affect the overall argumentation process. Future work is planned to perform analysis on the argument cluster contributors to determine if it is possible to group stakeholders in terms of their frequently expressed concerns, and analyze the relationships among contributor groups.

## REFERENCES

- [1] S. E. Toulmin, **THE USES OF ARGUMENT**, Cambridge, UK, University Press, 1958.
- [2] J. Conklin, M.L. Begeman, gIBIS: A Hypertext Tool for Exploratory Policy Discussion, *ACM Transactions on Information Systems (TOIS)*, Vol.6, No. 4, (1998) pp. 303-331.
- [3] N. Papadias, , HERMES: Supporting Argumentative Discourse in Multi Agent Decision Making". *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98)* (pp. 827-832). Madison, WI: AAAI/MIT Press.
- [4] C. J. Huang et al, Implementation and Performance of an Intelligent Online Argumentation Assessment System *International Conference on Electrical and Control Engineering*, (2010) pp. 25-27.
- [5] M. Klein, "The MIT Collaboratorium: Enabling effective large scale deliberation for complex problems." MIT Sloan Research Paper No. 4679-08. MIT Sloan Research Paper No. 4679-08. 2007
- [6] S. Osinski, S. ,D. Weiss, "A concept-driven algorithm for clustering search results" *Intelligent Systems*, IEEE Volume:20, Issue: 3, 2005 pp. 48-54
- [7] X. F. Liu.,S. Raorane, M. Leu., Web Based Intelligent Collaborative System for Engineering Design, *Collaborative Product Design and Manufacturing Methodologies and Applications*, London: Springer, 2007, pp. 37-58J.

# Feature Location in a Collection of Product Variants: Combining Information Retrieval and Hierarchical Clustering

Hamzeh Eyal-Salman, Abdelhak-Djamel Seriai, Christophe Dony

UMR CNRS 5506, LIRMM, University of Montpellier 2 for Sciences and Technology, France

E-mail: {Eyalsalman, Seriai, Dony}@lirmm.fr

## Abstract

*Locating source code elements relevant to a given feature is an important step in the process of re-engineering software variants, developed by an ad-hoc reuse technique, into a Software Product Line (SPL) for systematic reuse. Existing works on using Information Retrieval (IR) techniques do not consider the abstraction gap between feature and source code levels. In our recent work, we have improved the effectiveness of IR-based feature location by introducing an intermediate level between feature and source code levels, called “code-topics”. We used Formal Concept Analysis (FCA) to identify such “code-topics”. In this paper, we investigate the results of using Agglomerative Hierarchical Clustering (AHC) algorithm to identify code-topics. In our experimental evaluation, we show that AHC significantly increases the recall of feature location with a minor decrease of precision compared to FCA.*

**Keywords:** Software product line, product variants, feature location, FCA, AHC, IR.

## 1 Introduction

Software product variants are a set of similar software products developed by ad-hoc copying with adaptations to meet new demands of customers [1]. They share some features, called *mandatory features*, and also support different, customer-specific features, called *optional features*. A feature is “a prominent or distinctive user-visible aspect, quality or characteristic of a software system” [2]. As numbers of variants and features grow, maintaining such software variants and developing new variants becomes more difficult and expensive over time. Therefore, a transition to a systematic reuse approach, such as Software Product Line Engineering (SPLE) becomes necessary.

SPLE is an engineering discipline providing methods to promote systematic software reuse for developing short time-to-market and quality products in a cost-efficient way [3]. These products are known as SPL. SPL is rarely developed from scratch. It is often built by exploiting artifacts of pre-existing software product variants. To re-

engineer software variants into a SPL, it is important to locate source code elements (e.g., classes) that implement each feature [4]. Feature location is needed to understand the source code of software variants and support product derivation from SPL core assets [3]. Manually locating feature implementations is an error-prone and time consuming task. This is because maintainers must understand several software artifacts to decide which feature is implemented by which source code elements.

Information Retrieval (IR) techniques have been widely used for feature location [5][6]. In our recent work [4], we have improved the effectiveness of IR-based feature location in a collection of product variants by bridging the abstraction gap between feature and source code levels. This bridging is performed by introducing an intermediate level, called *code-topic*. A *code-topic* is a cluster of similar classes that reveal source code intention. A *code-topic* can be a functionality implemented by the source code and provided by a feature. In our recent work [4], we have combined a technique called Formal Concept Analysis (FCA) and IR to identify such *code-topics*. In this paper, we propose to investigate the results of using Agglomerative Hierarchical Clustering (AHC) to identify “code-topics”.

The rest of this paper is organized as follows. Section 2 presents necessary background to understand our proposal. Then, Section 3 shows how AHC can be used to identify *code-topics*. In sections 4 and 5, we present experimental evaluation and discuss related work, respectively. Finally, Section 6 concludes the paper.

## 2 Background

### 2.1 IR-based Feature Location

IR-based feature location methods exploit source code information (identifier and comments) to locate a feature’s implementation. These methods work by conducting lexical matching between source code information and feature information (i.e., feature description). Different IR techniques, such as Vector Space Model (VSM) and Latent Semantic Indexing (LSI), have been proposed in the context of locating features in the source code. These techniques

share four steps: *corpus creation, preprocessing, indexing and querying*. For more details, the reader can refer to [7]. In both LSI and VSM, the textual similarity between source code documents (corpus) and features documents (queries) is measured by the cosine of the angle between their corresponding vectors. Both LSI and VSM return a ranked list of source code documents against each feature document.

The effectiveness of IR techniques is measured by their *recall, precision and F-Measure* [7]. For a given query, recall is the percentage of retrieved documents that are relevant to the total number of relevant documents, while precision is the percentage of retrieved documents that are relevant to the total number of retrieved documents. F-Measure defines a tradeoff between precision and recall so that it gives a high value only in case where both recall and precision are high. All measures have values in [0,1].

## 2.2 IR-based Feature Location in a Collection of Product Variants: An Overview

In this section, we give an overview of our recent work [4], in order to more easily understand our proposal.

### 2.2.1 Basic Assumptions

We focus on functional features that express the behavior or the way users may interact with a product. We restrict ourselves to object-oriented systems. A functional feature can be implemented by a set of packages, classes, methods and attributes. A *class* represents a main building unit in all object-oriented programming where it encapsulates functionality and data. Moreover, developers typically think of a *class* as a set of responsibilities that simulate a concept or functionality from the application domain. Consequently, we assume that a functional feature is implemented by a set of classes.

The functional feature’s implementation spans multiple classes. We assumed that classes that contribute to implement a feature have shared terms and called near to each other. By grouping these classes into a cohesive unit based on their natural language content, we can get more relevant information describing features implemented by these classes. Therefore, we proposed the *code-topic*, as a coherent cluster of similar classes that are grouped based on their textual contents to implement a functionality. The *code-topic* constitutes an intermediate level that bridges the abstraction gap between feature and source code levels. Consequently, the functional feature (i.e., its description) can be textually matched to a set of *code-topics* (i.e., their source code information) representing its functionalities. This allows us to easily map a feature to a set of classes that are similar and grouped as a *code-topic* instead of mapping each feature to each class individually.

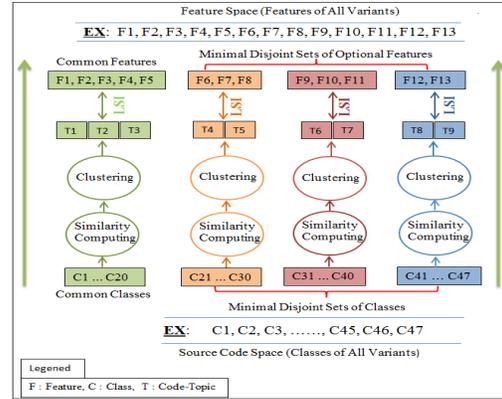


Figure 1. An Overview of the Code-Topic Identification Process.

### 2.2.2 Code-Topics Identification

In our recent work [4], we follow to strategy to improve IR-based feature location. Firstly, we determined mandatory features (resp. their classes) across software variants and grouped optional features (resp. their classes) into minimal disjoint sets at feature and source code levels. Secondly, we reduced the abstraction gap between feature and source code levels by introducing the *code-topic* as an intermediate level. In this paper, we only focus on reducing the abstraction gap.

Figure 1 gives a general overview of the two kinds of reduction. In this figure, we assume that a given collection of product variants provide a set of features  $\{F1, F2, \dots, F13\}$  and contain a set of classes  $\{C1, C2, \dots, C47\}$ . These features and classes are grouped into minimal disjoint sets at feature and source code levels. VSM is used to compute textual similarity between classes while FCA is used to cluster together similar classes. Each cluster is interpreted as a *code-topic*. LSI is used to link features and their corresponding *code-topics*. After determining the *code-topics* corresponding to each feature, we easily determine classes that implement each feature by decomposing each *code-topic* to its classes. In our approach, a *code-topic* may be associated to more than one feature.

## 3 Code-Topics Identification based on Hierarchical Clustering

Clustering, in general, is the division of objects into groups of similar objects. Each group, called a cluster, consists of objects that are similar amongst themselves and dissimilar to objects of other groups [8]. Clustering approaches are classified into hierarchical or non-hierarchical. Hierarchical clustering methods are further categorized into agglomerative (AHC for short) and divisive. Below, we present how AHC is used to identify code-topics.

### 3.1 Code-Topic Identification as a Partitioning Problem

According to our definition of the *code-topic*, the content of a *code-topic* matches a set of classes. Therefore, in order to determine a set of classes that can belong to a *code-topic*, it is important to formulate the code-topic identification as a partitioning problem. The input is a set of classes ( $C$ ). This set could be classes of the common partition or classes of any minimal disjoint set (see Figure 1). The output is a set of *code-topics* ( $T$ ) of  $C$ .  $C = \{c_1, c_2, \dots, c_n\}$  and  $T(C) = \{T_1, T_2, \dots, T_k\}$  where: (1)  $c_i$  is a class belonging to  $C$ ; (2)  $T_i$  is a subset of  $C$ ; (3)  $k$  is the number of identified *code-topics*; (4)  $T(C)$  does not contain empty elements:  $\forall T_i \in T(C), T_i \neq \Phi$ ; (5) The union of all  $T(C)$  elements is equal to  $C$ :  $\bigcup_{i=1}^k T_i = C$ .

For a given set of classes, we cluster them based on the strength of the relationship between the classes. In our case, this relationship refers to textual similarity. We use VSM to compute this similarity. We create a document for each class containing a list of terms extracted from all the identifiers of its corresponding class. VSM computes the textual similarity between two class documents by using cosine similarity between their corresponding vectors. One of these documents is treated as a query. The cosine similarity equation (refer to equation 1) represents a fitness function to decide which class documents belong to a cluster.

$$F(d_j, q) = \frac{\sum_{i=0}^n w_{i,j} w_{i,q}}{\sqrt{\sum_{i=0}^n w_{i,j}^2} \sqrt{\sum_{i=0}^n w_{i,q}^2}} \quad (1)$$

Where  $F$  is the fitness function with value in the range  $[0,1]$ .  $d_j$  and  $q$  are class and query vector documents respectively while  $w$  is a term weight. In our case, we try to maximize the fitness value for each cluster where similar classes use similar vocabulary.

### 3.2 Building a Hierarchy of Clusters

AHC groups similar classes that use similar vocabulary together and aggregate them into clusters. The basis for clustering classes is the strength of the relationship between them. The previously defined fitness function is used to measure this strength. AHC relies on a series of successive binary mergers, initially of individual class documents and later of clusters formed during the previous stages. We obtain from these binary mergers a single cluster dendrogram (*dendgr*) that contains a set of nested clusters. Figure 2 shows an example of dendrogram tree. At the lowest level, each class is in its own unique cluster. At the highest level, all classes belong to the same cluster. The internal nodes represent new clusters formed by merging the clusters that appear as their children in the tree.

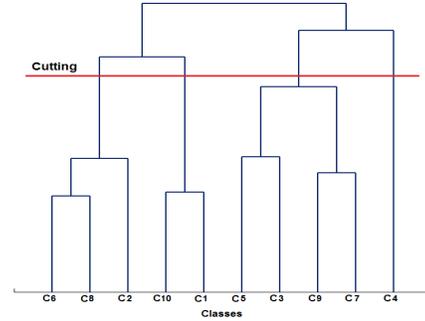


Figure 2. An example of dendrogram tree.

### 3.3 Selection Candidate Code-Topics

Breaking the dendrogram tree based on predefined criteria allows to group classes into clusters. Each cluster can be a candidate *code-topic*. Therefore, we must select the breaking point to obtain *code-topics*. This selection is performed by an algorithm based on a depth-first search (refer to algorithm 1). This algorithm takes as input the dendrogram tree and returns a set of clusters. We interpret these clusters as *code-topics*. For each node in the dendrogram (starting from the root), we compare the fitness value of the focused node and its sons ( $d_j$  and  $q$  in equation 1 become cluster nodes). If the fitness value of the focused node is less than the average of the fitness values of its two sons, then the algorithm continues on to the next son nodes. Otherwise, the focused node is identified as a *code-topic*, added to the *code-topics*( $T$ ) and the algorithm computes the next node in the stack. In this way, the most relevant *code-topics* will be identified as the traversal continues.

---

#### Algorithm 1: CodeTopicDendrogramTraversal

---

**Input:** *Dendrogram*(*dendgr*)  
**Output:** *Code-Topics*( $T$ )

- 1 *stack traversedClusters*
- 2 *push*(*traversedClusters*, *dendgr*)
- 3 **while** ( $|\textit{traversedClusters}| > 0$ ) **do**
- 4     *parent*  $\leftarrow$  *pop*(*traversedClusters*)
- 5     *son1*  $\leftarrow$  *getSon1Cluster*(*parent*, *dendgr*)
- 6     *son2*  $\leftarrow$  *getSon2Cluster*(*parent*, *dendgr*)
- 7     *avg*  $\leftarrow$  *average*( $F(\textit{son1}), F(\textit{son2})$ )
- 8     **if** ( $F(\textit{parent}) > (\textit{avg})$ ) **then**
- 9         | *add*(*parent*,  $T$ )
- 10    **else**
- 11         | *push*(*son1*, *traversedClusters*)
- 12         | *push*(*son2*, *traversedClusters*)
- 13    **end**
- 14 **end**
- 15 **return**  $T$

---

### 3.4 Code-Topics for Locating Feature Implementations

After identifying code-topics, we apply LSI (as shown in Figure 1) to link mandatory features and their possible corresponding code-topics, as well as to link each minimal disjoint set of optional features and their possible corresponding code-topics. LSI is applied by following the steps described in section 2.1, but we build LSI’s corpus and queries as follows. LSI’s corpus consists of code-topic documents, which each one corresponds to a code-topic. Each document consists of terms extracted from identifiers of corresponding code-topic’s classes. LSI takes code-topic and feature documents as input. Then, LSI measures the similarity between the code-topics and features using the cosine similarity. This returns a list of code-topics, ordered by their cosine similarity values against each feature. The retrieved code-topics should have a cosine similarity value greater than or equal to 0.70, where this value represents the most widely used threshold for the cosine similarity [5].

After linking each feature with all its corresponding code-topics, we can easily link each feature with its classes by decomposing each code-topic to its classes. For instance, if the feature *fl* is linked to two code-topics: *topic1*= *c1*, *c4* and *topic2*= *c7*, *c6*, by decomposing these topics into their classes; we can find that *fl* is implemented by five classes *c1*, *c4*, *c7*, *c6*.

## 4 Experimental Evaluation

### 4.1 Case Studies and Experiments Setting

To validate our approach, we have applied it to a collection of seven variants of a large-scale system, ArgoUML-SPL<sup>1</sup> modeling tool, and five variants of a small-scale system, MobileMedia<sup>2</sup>.

The ArgoUML-SPL is a Java open-source which supports all standard UML 1.4 diagrams. The ArgoUML-SPL’s products are generated from the same framework so that products that share the same features also share the same code. The selected products differ in terms of provided features but they support all UML diagram features. These features are implemented by source code classes. To establish ground truth links between features and their classes in order to evaluate our proposal, we compare code classes of two generated products; one of them provides all features while the other provides all features except the focused one. The obtained classes represent the real implementation of the focused feature. We repeat this process for all ArgoUML-SPL’s features. MobileMedia is a JAVA open source that manipulates multimedia on mobile devices. In our study, we have considered and analyzed variants that only implement features as classes. To establish ground

<sup>1</sup> Available at <http://argouml-spl.tigris.org/>

<sup>2</sup> Available at <http://www.ic.unicamp.br/~tizzei/mobilemedia/>

**Table 1. Precision, recall and F-measure of AHC against FCA for ArgoUML-SPL and MobileMedia.**

ArgoUML-SPL						
K	Precision		Recall		F-measure	
	AHC	FCA	AHC	FCA	AHC	FCA
0.1	52%	70%	99%	40%	68%	51%
0.2	52%	57%	99%	9%	68%	16%
0.3	52%	57%	98%	5%	68%	9%
0.4	52%	62%	98%	4%	68%	8%
0.5	52%	57%	96%	2%	67%	3%
MobileMedia						
K	Precision		Recall		F-measure	
	AHC	FCA	AHC	FCA	AHC	FCA
0.1	85%	85%	100%	100%	92%	92%
0.2	85%	85%	100%	100%	92%	92%
0.3	93%	93%	93%	93%	93%	93%
0.4	93%	93%	93%	93%	93%	93%
0.5	96%	96%	89%	89%	93%	93%

truth links between features and their source code classes, we analyze manually the source code.

The most important parameter to LSI is the number of chosen *term-topics*. A term-topic is a collection of terms that co-occur frequently in the documents of the corpus. The proper way to make such a choice is an open issue in the literature. Too many term-topics leads to the association of irrelevant terms and too few term-topics leads to loosely relevant terms. We are unable to use a fixed number of term-topics (*#term-topics*) because we have different sizes of code-topic documents. Therefore, we use a factor *K* between 0.1 and 0.5. *#term-topics* is equal to  $(K \times D_{dim})$ , where  $D_{dim}$  is the document dimensionality of the term-document matrix that is generated by LSI.

### 4.2 Results and Discussion

Table 1 summarizes precision, recall and F-measure results of locating all features of ArgoUML-SPL and MobileMedia by using AHC and FCA to identify code-topics.

On a large-scale system (ArgoUML-SPL), we notice that AHC significantly improves the recall values with a minor decrease in the precision compared to FCA. This improvement in recall is due to the fact that AHC identifies *code-topics* by determining a set of clusters so that classes of each cluster are similar amongst themselves and dissimilar to classes of other clusters. Regarding FCA, it identifies *code-topics* by determining a set of clusters in which all cluster’s members are similar to each other but it doesn’t consider similarity between clusters (refer to [4]). This means that FCA computes the textual similarity only among classes while AHC computes the similarity not only among classes but also among clusters. Therefore, the number of *code-topics* obtained by FCA is higher than AHC (423, 17 respectively). Identifying a small number of *code-topics* means that each code-topic document contain more rele-

vant source code information. This is allow to do better textual matching with feature descriptions, and hence the recall. Regarding the minor decreasing in the precision, this is due to the fact that AHC depends a lot on VSM compared with FCA. AHC uses VSM to compute similarity between classes and clusters while FCA uses VSM to compute similarity only between classes. This means that the number of false-positive links in the case of AHC is higher than FCA because VSM retrieve false-positive links which leads to impression. F-measure results refer to AHC achieve a better compromise between precision and recall than FCA.

On a small system (MobileMedia), it is observed that AHC and FCA produce the same precision, recall and F-measure results for the following reasons. Firstly, most minimal disjoint sets of optional features consist of only one feature, and hence their corresponding minimal disjoint sets of classes contain only the implementation of that feature, no more and no less). Thus, in this case we do not require LSI and *code-topics*. Secondly, MobileMedia's features are implemented by a small number of classes, sometimes by only two classes. These classes have little information that hinders building *code-topics*.

## 5 Related Work

The approach was proposed by Kuhn et al. [6] is the closet to ours. They proposed an approach to identify *linguistic topics* from object-oriented source code. Their approach relied on LSI to compute similarity among given set of methods, classes or packages. Then, AHC was used to cluster similar elements together as *linguistic topics*. The clusters retrieved by their approach are not necessarily domain concepts (i.e., features), but rather code-oriented topics. In our approach, we identify feature-oriented topics where these topics are identified from a set of classes that implement features.

Maskeri et al. [9] identify business topics from source code by using Latent Dirichlet Allocation (LDA). Their interpretation for a *topic* is a set of semantically related linguistic terms identified from identifiers names and comments. Kawaguchi et al. [10] proposed an automatic categorization method for a large collection of software systems. They use linguistic information in source code for identifying categories (topics) from open source repositories (e.g., *SourceForge*). A *category* is a cluster of related identifiers. Our approach differs from the works of *Maskeri et al.* and *Kawaguchi et al.* in two ways. Firstly, topics in their approaches are clusters of terms while code-topics in our approach are clusters of software artifacts (classes). Secondly, topics retrieved by their approach are code-oriented topics while we identify feature-oriented topics, as they are identified from source code classes that implement features (and only features).

## 6 Conclusion

In this paper, we have proposed to combine IR and AHC to improve the effectiveness of IR-based feature location in a collection of product variants. This improvement involves reducing the abstraction gap between feature and source code levels by introducing the concept of *code-topic* as an intermediate level. We have compared between two algorithms to identify code-topics: AHC and FCA. In our experimental evaluation using two different case studies, we showed that AHC significantly increases the recall of IR-based feature location with a minor decrease of precision compared to FCA.

## References

- [1] X. Yinxing, X. Zhenchang, and J. Stan, "Understanding feature evolution in a family of product variants," *Reverse Engineering, Working Conference on*, vol. 0, pp. 109–118, 2010.
- [2] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (foda) feasibility study," November 1990.
- [3] K. Pohl, G. Böckle, and F. J. v. d. Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005.
- [4] H. Eyal-Salman, A.-D. Seriai, and C. Dony, "Feature-to-code traceability in a collection of software variants: Combining formal concept analysis and information retrieval," ser. IRI'13. California, USA: IEEE, 2013, pp. 209–216.
- [5] A. Marcus and J. I. Maletic, "Recovering documentation-to-source-code traceability links using latent semantic indexing." in *ICSE*, L. A. Clarke, L. Dillon, and W. F. Tichy, Eds. IEEE Computer Society, 2003, pp. 125–137.
- [6] A. Kuhn, S. Ducasse, and T. Gırba, "Semantic clustering: Identifying topics in source code," *Inf. Softw. Technol.*, vol. 49, no. 3, pp. 230–243, Mar. 2007.
- [7] G. Salton and M. J. McGill, *Introduction to Modern Information Retrieval*. USA: McGraw-Hill, Inc., 1986.
- [8] P. Berkhin, "A survey of clustering data mining techniques," *Grouping Multidimensional Data*, pp. 25–71, 2006.
- [9] G. M. Rama, S. Sarkar, and K. Heafield, "Mining business topics in source code using latent dirichlet allocation." in *ISEC*, 2008, pp. 113–120.
- [10] S. Kawaguchi, P. K. Garg, M. Matsushita, and K. Inoue, "Mudablue: An automatic categorization system for open source repositories." in *APSEC*, 2004, pp. 184–193.

# Feature model recovery from product variants based on a cloning technique

Jihen Maâzoun  
Mir@cl Laboratory  
University of Sfax, Institut Suprieur d'Informatique et de Multimdia  
jihenmaazoun@gmail.com

Nadia Bouassida  
Mir@cl Laboratory  
University of Sfax, Institut Suprieur d'Informatique et de Multimdia  
Nadia.Bouassida@isimsf.rnu.tn

Hanêne Ben-Abdallah  
King Abdulaziz University, Jeddah, Kingdom of Saudi Arabia  
HBenAbdallah@kau.edu.sa

## Abstract

*A great number of Software Product Lines are not constructed from scratch, they are rather re-engineered from several similar product variants that have been in use. Existing methods for SPL feature model development from product variants may not be applicable when these products are developed independently and hence differ in their element names, methods code and structures. When different developers were involved in the development of the product variants, the naming assumption becomes too difficult to meet. In addition, the variants' code structures are often different when developed separately and/or when one variant is derived from another through several modifications. Furthermore, while an operation may keep its signature and name in different code variants, its internal code may be different to meet the specifics of each product variant. This paper tackles these three differences by proposing a feature model extraction method that harmonizes the names in the product variants using semantic criteria, tolerates structural differences, and identifies code variability through a clone-code detection technique. It illustrates the method applicability through an example and a CASE tool automating the method's steps.*

## 1 INTRODUCTION

Software Product Lines (SPLs) [5] have been recognized as a technique that improves productivity in software development. Using an SPL to derive a software product both

accelerates the development process and ensures the derivation of a good quality product. The *features* of an SPL can be used in different combinations to derive product variants in the application domain of the SPL. A feature is a prominent or distinctive quality or characteristic of a software system or systems [7]. An SPL is usually characterized by a feature model (FM) that indicates the features and their combination constraints (and, or, require,...).

In practice, the development of an SPL rarely starts from scratch. An SPL is often setup after several "similar" products have been developed usually via copy & paste techniques. This practical fact was used by several researchers to develop methods for the extraction of SPLs and their feature models from the source code of existing product variants, *cf.*, [1, 14]. These proposed methods are highly dependent on assumptions stemming from the use of copy & paste techniques. More specifically, they rely on three hypotheses about the product variants: they use the same vocabulary to name elements in their source code, they have very similar/identical structures, and the product variability which is encapsulated in the body of their operations is unimportant.

The "names-same structure" assumptions may not hold in the general setting where an SPL should be constructed from product variants that were produced by different developers, and/or product variants that endured too many modifications. In fact, while the names represent the application domain of the products, when different developers were involved in the development of the product variants, the naming assumption becomes too difficult to meet. In addition, the variants' code structures are often different when developed separately and/or when one variant is derived from an-

other through several modifications. For instance, a class in one product can be represented in a second product through two classes where the attributes and methods of the original class are distributed. A second example of product variability is when a class in one product was moved from one package to another package. For these simple examples of structural differences, existing feature identification methods would fail. Furthermore, while an operation may keep its signature and name in different product variants, its internal code may be different in order to meet the specifics of each product variant. If the degree of coding differences is very high, such an operation should not be identified as a core feature of the SPL because the product variants actually have different operations. Existing feature model identification methods do not treat the method's level of detail.

In our approach, we take into account the difference between the bodies of methods through code cloning. Many studies exist on code clones in object-oriented software systems. Some of them only focus on whether code clones exist or not [3] whereas others analyze code clones with respect to their effects, their removal or other peculiarities, identifying crosscutting concerns [2][11].

In this paper, we propose a bottom-up SPL construction approach that both accounts for the differences in the names and structures of the product source codes, and identifies automatically the feature constraints along with the features. The aim of this paper is to complete our approach [10] that extracts the feature model from product source codes and to present its associated tool.

More specifically, compared to our previous work [10], this paper has two contributions. First, it augments our feature model extraction method by the identification of method codes cloning. Secondly, it presents a CASE tool, named FMr-T (Feature Model recovery Tool), that implements our method.

The remainder of this paper is organized as follows. Section 2 overviews currently proposed approaches for feature identification from source code of product variants. Section 3 presents our method for feature extraction using the semantics enriched with method code clone detection. Section 4 describes the overall architecture of the tool and illustrates its functionality through an example of an SPL for banking systems. Finally, section 5 summarizes the paper and outlines our future work.

## 2 RELATED WORK

An SPL is often modeled in terms of a feature model. Kang et al [7] first proposed the use of feature models in 1990 as part of the Feature Oriented Domain Analysis (FODA). A *feature* can be seen as a system property or functionality that is relevant to some stakeholders. It is used to capture commonalities or differences among products in an

SPL. In fact, feature models are used to specify members of a product-line.

A feature model is a hierarchically arranged set of features. Relationships between a parent (or compound) feature and its child features (or subfeatures) designate the following selection strategies among features when deriving a product in the line of the SPL:

- **Mandatory:** (sub)features that must be present in every product in the line.
- **Optional:** (sub)features that may be present in some products.
- **And:** all subfeatures must be selected together.
- **Xor:** only one subfeature can be selected.
- **Or:** one or more subfeature(s) can be selected.
- **Require:** the selection of one subfeature necessitates the selection of the other.
- **Exclude:** two subfeatures cannot be part of the same product.

Note that a feature can be either simple/elementary like a package and a class, or composed of several elements like {package, Class}, {package, Class,attribute, method}...

Several works investigated feature model extraction from the source code of products in order to construct the SPL ([14], [1],[12], [9], [13]). For instance, Ziadi [14] propose an approach that first abstracts the input products in SoCPs (Sets Of Construction Primitives ) and, secondly, it identifies features by determining common and intersecting SoCPs. This approach was validated using two case studies: a banking example and the Argo-UML software product line. The obtained results show that the approach can not handle products with variable names of classes, methods and attributes. Moreover, this approach does not examine the body of the methods.

On the other hand, Al-Msie'Deen [1] propose an approach based on the definition of the mapping model between OO elements and feature model elements. This approach uses Formal Concept Analysis (FCA) to cluster similar OO elements into one feature. It uses Latent Semantic Indexing (LSI) to define a similarity measure based on which the clustering is decided. This approach improves the approach of Ziadi [14] since it extracts mandatory features and optional features along with some constraints among features like And and Require. However, it does not treat product variants with different structures or different terminologies.

In summary, the above reviewed works ignore the semantic aspect when extracting the SPL from the source code

of product variants. Besides the same terminology hypothesis, they suppose that the product variants have a similar/identical structure. However, even though products may vary in their package, class, attribute, and method names and structures, they solve semantically the same problems in their domain. These semantic relationships must be accounted for in spite of the structural differences.

### 3 SEMANTICS-BASED FEATURE MODEL EXTRACTION

In our approach, we suppose that products variants may have been constructed by different developers. Hence, the names and structures used may be different. Our approach exploits the semantics carried through the names of the classes, packages, attributes and method declarations within the product variants. In the other hand, it resolve the problems where the same method exists but with different bodies.

To account for the semantics, our feature model extraction operates in the following three steps:

- Name harmonization: in this pre-processing step, the semantic correspondences among the names of the packages, classes, methods and attribute are treated. This step relies on linguistic and typing information to harmonize the names and content of the method. It uses a cloning technique to resolve the problems where the same method exists but with different bodies.
- Features identification: In order to tolerate some structural differences among the source codes of the product variants, we adapt the FCA [8] and LSI [4] by considering the semantics.
- Feature model construction and constraints identification: In this last step, the semantic information is used first to define the hierarchy in the feature model and, secondly, to extract the types of constraints among identified features.

#### 3.1 Name harmonization

This pre-processing starts by identifying the semantic correspondences between the names of packages, classes, methods and attributes. For this step, we adapted the set of semantic criteria we defined in our previous works on constructing frameworks. The following three criteria express linguistic relationships between element names (however, the list can be extended):

- Synonyms(C1, . . . , Cn): implies that the names are either identical or synonym, e.g., Mobile-Mobile and Phone-Mobile.

- Hypernyms(C1; C2, . . . , Cn): implies the name C1 is a generalization of the specific names C2, . . . , Cn, e.g., Media-Video.
- Str.extension(C1; C2): implies that the name C1 is a string extension of the name of the class C2, e.g., Image-NameImage.

The determination of the above linguistic/semantic relationships can be handled through either a dictionary (e.g., Wordnet), or a domain ontology when available.

The above semantic criteria are insufficient when two methods have the same or synonymous names but different bodies. To resolve this problem and to extract the variability encapsulated in the body of the methods, we adapt a code cloning technique. To determine code cloning, we propose the following three rules:

<p><b>Rule1:</b> Method names are synonyms but bodies are different.  <math>(Synonyms(M_{P1}, M_{P2}))</math> and <math>(Body(M_{P1}) \neq Body(M_{P2}))</math>  <math>\rightarrow</math> Method names are kept unchanged</p>
<p><b>Rule2:</b> Method names and bodies are synonyms.  <math>(Synonyms(M_{P1}, M_{P2}))</math> and <math>(Body(M_{P1}) = Body(M_{P2}))</math>  <math>\rightarrow</math> one of the method names must be changed</p>
<p><b>Rule3:</b> Same Method names but method bodies are different.  <math>(M_{P1} = M_{P2})</math> and <math>(Body(M_{P1}) \neq Body(M_{P2}))</math>  <math>\rightarrow</math> One of the method names must be changed</p>

Note that, in example 1 of Figure 1, the methods *Correction()* and *Change()* are synonyms but they have different bodies. The difference between the bodies of these two methods exist in the loop: the method *Correction()* contains a *for* loop nested whereas the method *Change()* contains the *while* loop. Since, these methods have synonyms names and different structures, thus, there is no need to change their names. This case is treated by **Rule 1**.

In example 2 of Figure 1, the methods *PointerPressed()* and *PointerPressed(int x, int y)* have the same name but the bodies are different: the second method *PointerPressed(int x, int y)* differs from the first in the conditional structure *if*. In this case, one of the methods' names must be changed as proposed in **Rule 3**.

In example 3 of Figure 1, the methods *add image()* and *Add.Photo()* are synonyms and they have the same code. In this case, both names are transformed to a unique name as proposed in **Rule 2**.

At the end of the pre-processing step, all semantically related names would be harmonized and can then be analyzed through the FCA in the features identification step.

Method 1	Method 2
<pre> public void Correction () {     DOMParser parser = new DOMParser();     DOMParser parser1 = new DOMParser();     try {         parser.parse("fichier_xml1.xml");         parser1.parse("fichier_xml2.xml");         for (Element empElement : listEmpElement         for (Element empElement1:         listEmpElement1){             String s2=empElement.getText();             System.out.println(s2);             String s3=empElement1.getText();             System.out.println(s3);         }     } catch (Exception e) {         System.out.println(e.getMessage());     } } </pre>	<pre> public void Modification () {     DOMParser parser = new DOMParser();     DOMParser parser1 = new DOMParser();     try {         parser.parse("fichier_xml1.xml");         parser1.parse("fichier_xml2.xml");         while(c.hasNext() &amp;&amp; cl.hasNext()){             Element courantc = (Element)c.next();             String C = courantc.getText();             System.out.println(C);             Element courantcl = (Element)cl.next();             String Cl = courantcl.getText();             System.out.println(Cl);             if (C.equals(Cl)){                 System.out.println("nn");             }         }     } catch (Exception e) {         System.out.println(e.getMessage());     } } </pre>
<pre> public void PointerPressed(){     Dismiss(); } </pre>	<pre> Public void PointerPressed(int x, int y){     if(x&lt;y) {         Dismiss();     } } </pre>
<pre> Method 1 public String Add_image(){     return image.getString(); } </pre>	<pre> Method 2 public String Add_Photo(){     return image.getString(); } </pre>

Figure 1. Difference between bodies of two methods

### 3.2 Feature identification step

In this step, we use FCA and LSI to extract the common and variable parts among the harmonized product variants. Before explaining this step, let us first overview the basics of FCA and LSI.

Formal concept analysis (FCA) [6] is a method of data analysis with a growing popularity across various domains. FCA permits to analyze data described through the relationships among a particular set of data elements. In our approach, the data represent the product variants being analyzed; the data description is represented through a table where the product variants constitute the rows while source code elements (packages, classes, methods, attributes) constitute the columns of the table.

From the table, a concept lattice is derived. The concept lattice permits, in the first time, to define commonalities and variations among all products. The top element of the lattice indicates that certain objects have elements in common (i.e., common elements), while the bottom element of the lattice shows that certain attributes fit common objects (variations). The elements are grouped in blocks. First, common elements are common block which are commonly used in all products. Secondly, the blocks of variations only appear in specific products. Common blocks and block variations are composed of atomic blocks of variation representing only one feature.

Besides the blocks, the lattice also indicates the relationships among elements. The following relationships can be automatically derived from the sparse representation of the lattice and presented to the analyst:

- *Mandatory*: features that are used in every product and that appear at the top of the concept lattice.

- *Optional*: features that are used only in some products and that appear at the bottom of the concept lattice.
- *Xor*: features F1 and F2 that appear in different concepts and whose infimum is the bottom concept are only used in alternative in the product. These features are likely to be Xor features,
- *Require*: if any element (package, class, method, attribute) belonging to the feature F1 has in common attributes and methods with F2. Then they will be related with a require.
- *AND*: features F1 and F2 that appear in the same concept.

LSI measures the similarity degree between names of packages, classes, methods and attributes. Informally, LSI assumes that words that always appear together are related [4]. Consequently, we use LSI and FCA to identify features based on the textual similarity. Similarity between lines is described by a similarity matrix where the columns and rows represent lines vectors. LSI uses each line in the block of variations as a query to retrieve all lines similar to it, according to a cosine similarity. In our work, we consider the most widely used threshold for cosine similarity that equals 0.70 [4]. The similarity matrix which is the LSI result is used as input for the FCA to group the similar elements together based on the lexical similarity. Thus, any document that has similarity, only with itself will be ignored. We take the interchanged context as input for FCA which identifies the meaningful groupings of objects that have common attributes.

The application of the name harmonization step followed by the feature identification step extracts candidate features without any structure or hierarchy. The next step in our

approach determines the hierarchy and constraints among features and finalizes the feature model construction.

### 3.3 Feature Model construction and constraints identification

This phase has a threefold motivation. First, the features which are composed of many elements (package, classes, attributes, methods) are renamed based on the frequency of the names of its elements. In addition, the organization and structure of the features is also retrieved based on the semantic criteria. In fact, since the owner information was omitted, then to retrieve the organization of the features, we use the semantic criterion.

- *Hypernyms*(FeatureN1, FeatureN2)  $\rightarrow$  FeatureN1 is the parent of FeatureN2
- *str\_extension*(FeatureN1, FeatureN2)  $\rightarrow$  FeatureN1 OR FeatureN2
- *Synonyms*(FeatureN1, FeatureN2)  $\rightarrow$  FeatureN1 XOR FeatureN2

In fact, *str\_extension*(MediaListScreen, PhotoListScreen) and *str\_extension*(MediaListScreen, VideoListScreen) implies that features Photo and Video are related with OR.

As an example, the constraints between the different features that are extracted with FCA and LSI are verified and some others are added based on the semantic criteria. At the end of this last step, all the features are collected in a feature model to specify the variations between these products.

## 4 FMr-T: Feature Model recovery from source code Tool

In this section, we first present the main functionalities of the tool "FMr-T" (Feature Model recovery Tool). Secondly, we illustrate it through an example.

### 4.1 Functional architecture

The main functionalities performed by our tool are:

- Extraction of the tree containing packages, classes, methods and attributes names, and saving the information in an XML file.
- Verification of the semantic relationships with WordNet which gives the list of synonyms, hyperonyms of source code elements. Then, the detection of code clones of method bodies is performed.
- Correction of the names and modification of the XML file.
- Application of FCA and LSI to define the common blocks and atomic blocks of variation.

### 4.2 The functionalities of FMr-T

To illustrate the functionalities of the FMr tool, let us consider a set of banking software products. These source codes are developed with the JAVA language. In the first step, the user chooses the source code file, then the tree will be extracted and saved in an XML document. The XML document corresponds to the name of elements of the parsed code (package, class, method, attribute).

In the second step, to determine the correspondences between the names (package, class, method, attribute), we adapted the set of semantic criteria defined in Wordnet (see interface 1 of figure 2). The semantic relationships between the names of methods, classes, attributes, and packages of two sources of product lines codes are presented (synonymy, hypernymy, meronymy and str\_extension (defined in section 3.1)) After the semantic verification, the clone detection is performed. This allows to check the methods' bodies and detect some structural differences. In fact, a clone detection strategy is specified to verify the methods bodies using the rules presented in section 3.2. In our example, the interface 2 of Figure 2, presents the GUI to visualize variability in the body of methods. Methods *sum()* and *account()* are synonyms. In fact, we display the "clone detection" button that permits to apply rules presented in section 3.2 and display the result of clone detection as shown in part "result" of interface 2 of Figure 2. As result, the method *sum()* and *account()* will not be changed.

After applying the name harmonization, the XML file will be updated and will be an input of FCA method (see the interface 3 of figure 2) that permits to determine common blocks and blocks of variation (shown in the interface 4 of figure 2).

Common blocks and blocks of variations are composed of atomic blocks of variation representing only one feature. To define features, we apply LSI with Matlab. According to a cosine similarity that is equal to 0.70, LSI uses each line in the block of variations as a query to retrieve all lines similar to it. The similarity matrix which is the LSI result is used as input for the FCA to group the similar elements together based on the lexical similarity.

## 5 CONCLUSION

This paper first overviewed existing works for feature model extraction from product variants. Secondly, it presented a new approach based on a set of linguistic criteria to identify a feature model from different product source codes. Besides accounting for naming differences, our approach has the advantage of identifying automatically the features and their constraints in source codes with different structures and it resolves the problem of variability in methods bodies. The paper illustrated the proposed approach

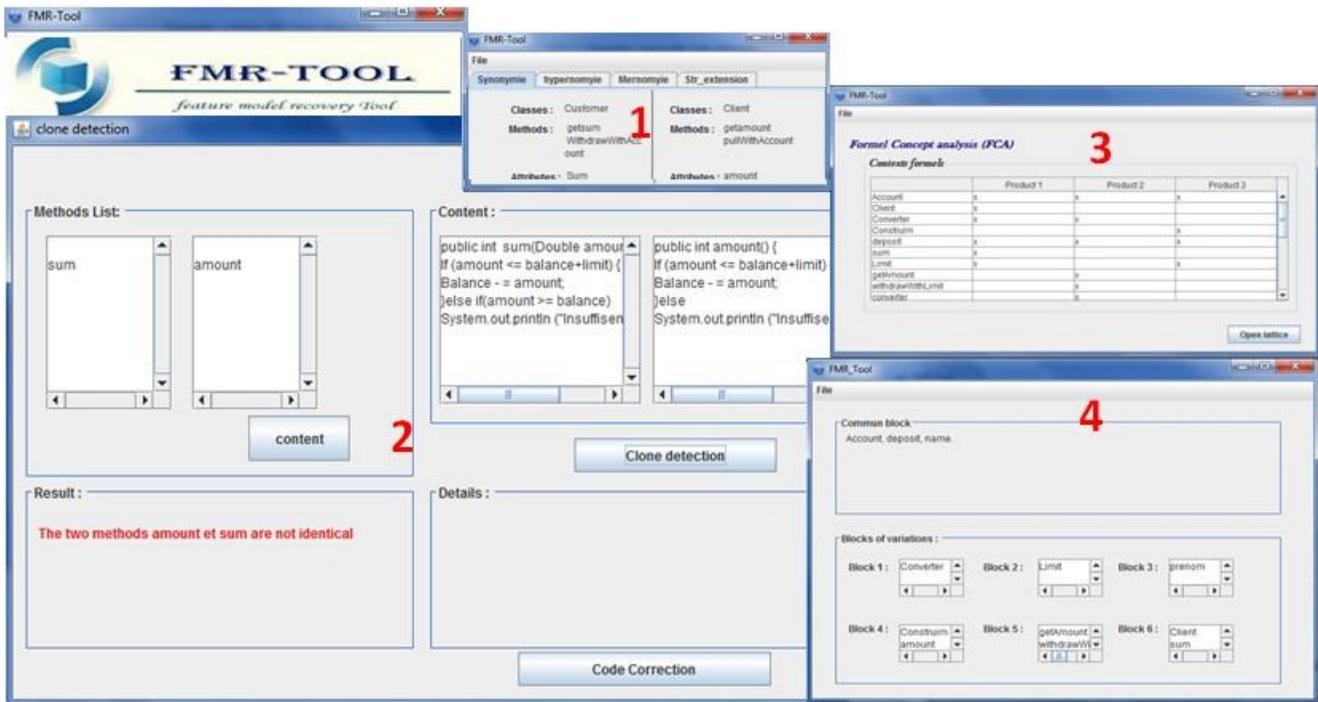


Figure 2. FMr-T: Feature Model recovery from source code Tool

through the extraction of the feature model of an SPL for banking systems. The case study was conducted through the "FMr-T" tool which implements the steps of our approach.

In our ongoing works, we are examining how to add more intelligence in the feature model extraction. We will also consider the use of semantics in the refactoring of software product lines.

## References

- [1] R. Al-Msie'Deen, A. Seriai, M. Huchard, C. Urtado, S. Vauttier, and H. Salman. An approach to recover feature models from object-oriented source code. In *Day Product Line 2012*, 2012.
- [2] M. Balazinska, E. Merlo, M. Dagenais, B. Lagüe, and K. Kontogiannis. Advanced clone-analysis to support object-oriented system refactoring. In *Proceedings of the Seventh Working Conference on Reverse Engineering (WCRE'00)*, pages 98–.
- [3] I. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier. Clone detection using abstract syntax trees. In *Proceedings of the International Conference on Software Maintenance, ICSM'98*, 1998.
- [4] D. Binkley and D. Lawrie. Information retrieval applications in software maintenance and evolution. In *Encyclopedia of Software Engineering*, 2011.
- [5] P. Clements and L. Northrop. Software product lines: Practices and patterns. *SEI Series in Software Engineering*, 2001.
- [6] B. Ganter and R. Wille. Formal concept analysis: Mathematical foundations. *Springer-Verlag*, 1996.
- [7] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson. Feature-oriented domain analysis (foda) feasibility study,. *Technical report CMU/SEI-90-TR-21*, Software Engineering Institute, Carnegie Mellon University,, 1990.
- [8] F. Loesch and E. Ploedereder. Restructuring variability in software product lines using concept analysis of product configurations. pages 159–170, 2007.
- [9] A. Lozano. An overview of techniques for detecting software variability concepts in source code. In *ER Workshops*, pages 141–150, 2011.
- [10] J. Maazoun, N. Bouassida, and H. Ben-Abdallah. Feature model extraction from product source codes based on the semantic aspect. *ICSOFT13*, pages 154–161, July 2013.
- [11] A. M. D. Nakae, T. Kamiya, S. Sato, and K. Matsumoto. Software quality analysis by code clones in industrial legacy software. 2002.
- [12] P. Paskevicius, R. Damasevicius, and V. tuikys. Quality-oriented product line modeling using feature diagrams and preference logic. In *Information and Software Technologies*, pages 241–254, 2012.
- [13] J. Rubin and M. Chechik. Combining related products into product lines. In *FASE*, pages 285–300, 2012.
- [14] T. Ziadi, L. Frias, M. A. A. da Silva, and M. Ziane. Feature identification from the source code of product variants. pages 417–422, 2012.

# RELREA - An Analytical Approach for Evaluating Release Readiness

S. M. Shahnewaz  
Department of ECE  
University of Calgary  
Calgary, Alberta, Canada  
[smsahne@ucalgary.ca](mailto:smsahne@ucalgary.ca)

Guenter Ruhe  
Department of Computer Science,  
Department of ECE  
University of Calgary  
Calgary, Alberta, Canada  
[ruhe@ucalgary.ca](mailto:ruhe@ucalgary.ca)

**Abstract**— As part of incremental and iterative software development, decisions about “Is the software product ready to be released at some given release date?” have to be made at the end of each release, sprint or iteration. While this decision is critically important, so far it is largely done either informally or in a simplistic manner, relying on a small set of isolated metrics. In this paper, we present an analytical approach combining the goal-oriented definition of the most relevant readiness metrics with their individual evaluation and their subsequent analytical integration into an aggregated evaluation measure. The applicability of the proposed approach called RELREA is demonstrated for an ongoing public project hosted on GitHub, a web-based hosting service for software development projects. Initial evidence shows that the method is supportive in evaluating release readiness at any point of the development cycle, making projections on the final release readiness and allows determination of bottleneck factors to achieve readiness.

*Keywords*-release date; release readiness; release criteria; fuzzy set; aggregation; case study

## I. INTRODUCTION

To achieve business success of products, in-time and in-quality of releasing software is a key concern of the software companies [1]. Delays in releasing software may cause substantial loss in revenue and in the worst case, failure of the complete software product. One of the main challenges of software product manager is knowing the current state of the readiness of the product based on objectivity [15]. In most cases, release readiness is evaluated at the end of the development cycles (i.e. milestones, sprints, etc.) relying on a set of isolated metrics. This has some major consequences: Firstly, there is no continuity in monitoring product readiness. Secondly, any problem related to release issues cannot be addressed proactively by the development team. Thirdly, if being below expectations in terms of readiness, it is unknown which are the limiting factors.

Evaluating release readiness at any point of time is a challenging task. One of the fundamental issues is that the definition of software release readiness is not well articulated in literature. Most of the related works [2-4, 12] in this domain mainly gauge with defect tracking in order to define release readiness. However, release readiness measures should include all important aspects of the software product [21]. In this paper, we define release readiness as an objective

measure, which is determined by aggregating the degree of satisfaction of a set of individual readiness criteria. According to this definition, the main problem of evaluating release readiness is determining the satisfaction levels of the criteria based on monitoring key product and process performance metrics.

In this paper, we propose RELREA, an analytical approach that allows evaluating release readiness at any point in time of the release cycle and making suggestions to improve the status of readiness for the final software product release. The main characteristics of RELREA are:

- Project specific release readiness criteria and metrics are selected in a systematic way.
- Unlike the existing methods (see Section III), overall release readiness is determined by aggregating and evaluating the degree of satisfaction of the criteria.
- Continuous visibility on release readiness status.
- Projection of release readiness at release time.

In the proposed approach, the concept of fuzzy set theory is employed for evaluating the degree of satisfaction of the readiness criteria based on the evaluation of the pre-defined objective measures. As an aggregation operator, Ordered Weighted Average (OWA) is applied. In order to demonstrate the applicability of the proposed approach, an illustrative case-study is presented on an ongoing public project hosted on Github. The initial evidence shows that the new method helps product managers investigating release readiness at any point of time during the development cycles.

The paper is subdivided into seven sections. In Section II, we present the background for concepts used by the proposed method. A brief discussion of related work and tools is done in Section III. The workflow of the proposed analytical approach for evaluating release readiness is presented in Section IV. In Section V, a case-study is presented to demonstrate the applicability of the proposed method. Section VI outlines the discussion on threats to validity. The final section provides conclusions and outlines future work.

## II. BACKGROUND

In this section, we provide the fundamental concepts that we have used in the proposed method.

### A. Software Release Readiness Criteria

Release readiness criteria can be defined uniformly by considering both organization goals and customer expectation of a software product. Four important dimensions of release readiness (*implementation status, testing scope and status, source code quality and documentation status*) were identified from existing literature [2-6] and tools [14, 15]. An overview of the release readiness criteria for each dimension is provided in Table I. These factors cover both important artefacts (i.e., features, source code, documents, etc.) and development activities (i.e., coding, building, testing, etc.).

TABLE I. READINESS FACTORS AND THEIR RELATED CRITERIA

Readiness dimensions	Overview of related Release Readiness Criteria
Implementation of functionality	Criteria related to feature implementation, change request implementation, coding effort, continuous integration, build trends, etc.
Testing	Criteria related to defect finding, defect fixing, reliability, test coverage, test effort, etc.
Source code quality	Criteria related code review, coding style, code smells, refactoring, code complexity, etc.
Documentation	Criteria related to user manual, design documents, test specification, test case documentation, etc.

### B. Goal Question Metric (GQM)

The Goal-Question-Metric (GQM) paradigm [8] is the de-facto standard to perform software measurement. In its essence, it guides the process of designing an effective measurement program and finding insights from the data collected. In the context of release readiness analysis, our goal is to evaluate the status of the overall readiness. The readiness dimensions corresponding to questions were refining the goal. Available data associated with every question were used to answer them in a quantitative way. An example of defining objective metrics for the testing dimension is shown in Table II.

TABLE II. EXAMPLE OF GQM APPROACH

Goal	Question (related to readiness criteria)	Metric
Assessing satisfaction of testing	Is the testing activity reducing the defects?	Defect find rate
	To what extent source code are covered by unit test?	% Line of Code (LOC) covered

### C. Evaluation of Release Readiness Criteria with the Concept of Fuzzy Set Theory

The concept of fuzzy set theory was introduced by Zadeh [9] in 1965 to represent nonstatistical uncertainty and vagueness associated with data and information. Since then, it has been widely used to solve problems in various decision making environment. In software engineering, fuzzy set theory has been applied successfully to deal with the uncertainty associated with the aspects such as cost estimation [17], reliability prediction [18], and imprecise requirement analysis [19].

**Definition 1:** A fuzzy set  $C$  in a universe of discourse  $X$  is characterized by a membership function  $\mu_C(x)$  which associates with each element in a real number in the interval  $[0, 1]$ . The function value  $\mu_C(x)$  is termed the “grade of membership”  $x$  of in  $C$ .

The concept of membership function stated in Definition 1, can be used to evaluate the degree of satisfaction of the readiness criteria discussed in II.A. For example, suppose we want to evaluate the *satisfaction of defect finding* criteria. In this regard, *defect arrival rate (DAR) (defects/day)* is an objective metric which can be used to evaluate the criteria. Let,  $X = R^+ = \{\text{positive real number that can be represented as value of DAR}\}$  and  $C = \{\text{satisfaction of defects per day}\}$  be a fuzzy set representing the criteria.

As shown in Figure 1, its membership function is defined as  $\mu_C(x)$  which means *DAR* less than 1 defects/day is considered full satisfaction of defect finding, while defect arrival rate less than 10 defects/day is considered the criteria is not satisfied at all. Values of *DAR* between, 1 and 10 are considered satisfaction to some degree.

According to this definition, if the value of *DAR* is 5 defects/day, then the “degree of membership” or “degree of satisfaction of defect finding” will be 0.55. The definition of this membership function is context specific, different projects may have different perception about the satisfaction of defect finding. Thereby, degree of satisfaction of the readiness criteria can be determined by defining the membership functions.

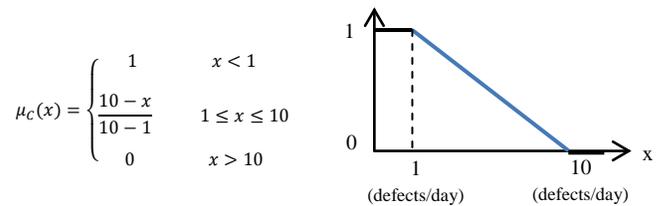


Figure 1: Membership function of the fuzzy set  $C = \{\text{satisfaction of defects per day}\}$

### D. Elicitation of Membership function

Defining membership function is one of the fundamental issues associated with the application of fuzzy set theory. Success of a solution approach largely depends on the proper definition of membership functions. In application of fuzzy set theory, heuristic based approaches are frequently applied where predefined shapes for membership functions are used.

Two broadly used categories of heuristic membership function are *piecewise linear functions* and *piecewise monotonic functions* [10, 22]. By selecting an appropriate shape and providing its specific parameters, membership functions can be elicited for a particular release readiness criterion. One limitation of this approach is that the parameters of the selected functions must be provided by the experts and need to be fine-tuned until the performance is acceptable.

### E. Aggregation of Values of Membership Functions

As discussed in section II.C and II.D, values of the membership functions represents the degree of satisfaction of

the readiness criteria. We need to aggregate the individual evaluation of the readiness criteria to obtain an overall evaluation. In literature, many aggregation operators are proposed. An overview of the properties of those operators can be found in [11]. Selection of the aggregation operator depends on the context of its use. In the context of evaluating overall release readiness, aggregation operator should have the capability of including the following special considerations along with the general mathematical properties (i.e. boundary conditions, monotonicity, continuity, associativity, symmetry, neutral element, idempotence, etc.).

- The first consideration is the relative importance or weights of the readiness criteria. This allows the decision maker to emphasize important readiness criteria, so that its influence on the overall readiness will be higher.
- Second consideration is the decision maker’s desired decision strategies. For example, a pessimistic decision maker desires that most of the criteria be satisfied. Conversely, an optimistic decision maker will be satisfied if some of the criteria are met.

In this paper, Ordered Weighted Average (OWA) [7] is applied as aggregation operator because of its capability of incorporating the above special considerations. Definition of overall release readiness with OWA operator is stated in the following:

$$OWA(\mu_{c_1}(x), \mu_{c_2}(x), \dots, \mu_{c_n}(x)) = \sum_{i=1}^n w_i \cdot b_i \quad (1)$$

Where

- i.  $\mu_{c_1}(x), \mu_{c_2}(x), \dots, \mu_{c_n}(x)$  are the values of the membership functions of the criteria  $C_1, C_2, \dots, C_n$  with  $\mu_{c_i}(x) \in [0,1]$ ,
- ii.  $w_1, w_2, \dots, w_n$  are the ordered weights satisfying  $w_i \in [0,1]$ ,  $\sum_{i=1}^n w_i = 1$ , and
- iii.  $b_i$  is the  $i^{th}$  largest element in the collection  $\mu_{c_1}(x), \mu_{c_2}(x), \dots, \mu_{c_n}(x)$ .

Yager [7] showed that OWA operators lie between maximum (“or”) and minimum (“and”) of the scores to be aggregated. Details about the properties of OWA operators and ways of including relative importance of the criteria can be found in [20].

### III. RELATED WORK

Literatures provide several techniques and tools that aim to determine release readiness before releasing software. In [5, 6], a set of product and process oriented objective measures are defined from various dimensions (i.e. functionality, quality, source code, etc.) and their values are aggregated to calculate the release readiness index. Though these approaches consider various aspects of software development life-cycle, regression model based metrics aggregation methods have narrowed down their applicability in decision making environment. For example, they do not facilitate managers to perform what-if analysis on release decisions.

Most of the other works [2-4, 12, 16] in this domain have emphasized on the trends of the metrics related to defect tracking and testing processes for evaluating release readiness. Staron et al. [13] has proposed time-to-release as a release readiness indicator for agile development by combining a set of metrics such as *# of defects, defect removal rate, test execution rate and test pass rate*. Testing phase based objective measures such as *code turmoil, test passing rate, defect find Rate and # of open defects* are used to assess the release readiness of an embedded system developed in HP lab [12]. These approaches are effective in the sense that they help the management staffs to understand the risk associated to each metrics to deliver the product on predefined release date. However, readiness analysis based on these methods largely depends on the past projects data which are not always available.

Along with the literature, there are some commercial tools which collect and present important product and process oriented metrics to the management staffs as evidences that the product is ready to deliver to the customer. Tools such as Borland TeamInspector [14] and PTC Integrity [15] extract and visualize metrics related to functionality, code analysis, test coverage, standard compliance, and budget and schedule to verify software is ready to release. However, metrics visualization does not provide effective ground for analyzing release readiness.

The discussion on the related works shows that, there is no uniform method which can be applied for evaluating release readiness. Moreover, the proposed methods do not enable managers to make many kind of predictions, assessments, and trade-offs about releasing software product. In this work, we mitigate these limitations by proposing an approach which allows continuous evaluation and prediction of product release readiness at any point of time during the development cycles.

### IV. RELEASE READINESS APPROACH

In this section, we present the workflow of the proposed RELREA release readiness approach.

#### A. Workflow of the proposed approach

The workflow of the proposed release readiness analysis method is illustrated in Figure 2. There are 8 main steps:

- 1) *Define Readiness criteria and metrics*: At the beginning of the project, based on the GQM approach discussed in Section II.B, the release manager will define the context specific readiness criteria and their corresponding objective metrics.
- 2) *Define membership function*: Based on the proposed membership function elicitation techniques described in Section II.D, product manager will define the membership functions by selecting appropriate shapes and their parameters for the readiness criteria.
- 3) *Data collection*: At any point of the current development cycle, data related to the pre-defined metrics will be collected (by automated tools).
- 4) *Computation of criteria satisfaction levels*: At this point, satisfaction levels of the readiness criteria will be

determined by computing the *degree of membership* for the current values of the objective metrics at a specific point in time  $t_i$ .

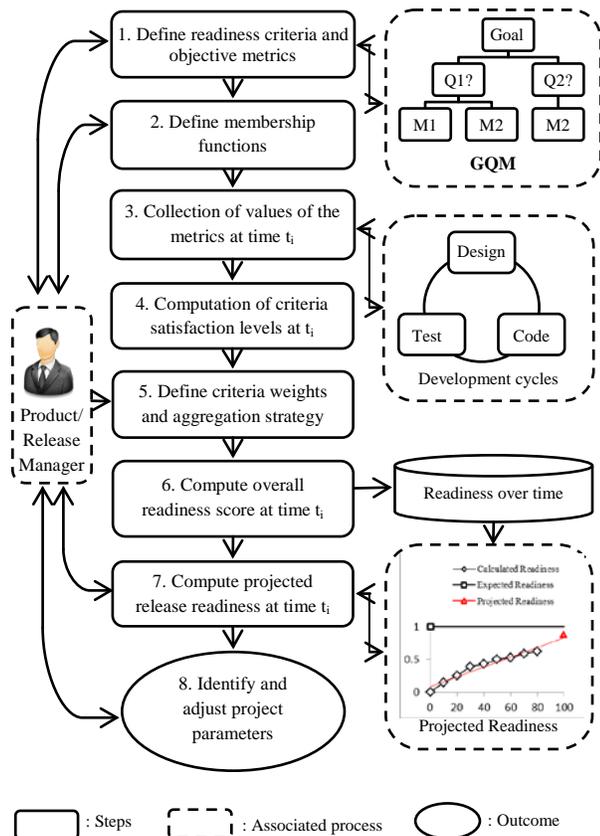


Figure 2. Workflow of the proposed release readiness analysis method.

- 5) *Define criteria weights and aggregation strategy*: By analyzing the readiness criteria, the release manager will define their relative weights. At the same time, she will specify the desired strategy (degree of optimism) for aggregating satisfaction levels of the readiness criteria.
- 6) *Computation of overall readiness*: The overall release readiness is determined by aggregating the satisfaction levels of the criteria based on (1).
- 7) *Compute projected readiness*: In this step, the projected release readiness at the given release date is made. Details about this step is discussed in IV.B.
- 8) *Identify and adjust project parameters*: This step allows the release manager to further analyze current release readiness of the product and projected readiness at the end. This includes identification of the project parameters (i.e. testing effort, coding effort, system capacity, and etc.) which need to be adjusted to achieve the target expectations (i.e. expected readiness, release window, customer satisfaction) of the product. Performing what-if analysis by varying the parameters of the membership functions helps to better understand the impact of uncertainties.

## B. Projection of Release Readiness

Based on the calculated integrated readiness scores over a time period, projected readiness  $PR(T)$  for a pre-defined release date  $t = T$  is studied. As an approximation  $PR(T)$  is calculated based on the gradient of the line between last two evaluation points  $E1(t_{i-1}, r_{i-1})$  and  $E2(t_i, r_i)$ .

$$PR(T) = \sum w'_i \times PR(t_i), \text{ for } i=1, \dots, n \quad (2)$$

In (2),  $w'_i$  denotes the weights for each evaluation point. As latest evaluation points are more reliable, more weight is given to the projected scores calculated at later stage of the development compared to the scores of the later stages. This idea is illustrated in Figure 3, where each dashed line points to the projected readiness scores based on last two evaluations.

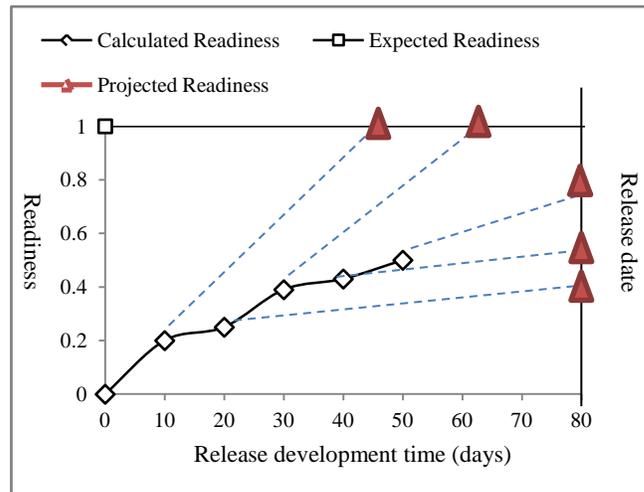


Figure 3. Projection of release readiness  $PR(T)$ .

## V. CASE STUDY

### A. Context of the Project

*Publify*<sup>1</sup> is a powerful open source blogging engine hosted on Github. It is one of the oldest Ruby on Rails project started in 2004. Since then, 52 contributors has engaged themselves in this project and contributing continuously to its development. It uses Github<sup>2</sup> for managing issues (i.e. features, bugs, tasks and etc.) and source code versioning. For the continuous integration (CI) it uses the Travis-CI<sup>3</sup> which is a cloud hosted CI platform.

The quality of the source code of the project is tracked with CodeClimate<sup>4</sup> that performs real-time static analysis on the source code repository in order to provide a comprehensive quality report. The latest version (*Publify* 8.0) of the engine was released on March 01, 2014. Though development activities of this release were kicked off on September 09, 2014, we started to monitor the project from January 26, 2014. Main goal of this case-study is to demonstrate the capability of

<sup>1</sup> <https://github.com/publify/publify/>

<sup>2</sup> <https://github.com/>

<sup>3</sup> <https://travis-ci.org/publify/publify>

<sup>4</sup> <https://codeclimate.com/github/publify/publify>

RELREA in evaluating and predicting release readiness of *Publify 8.0* at any point of time during the development.

TABLE III. CRITERIA, METRICS AND DATA USED IN THE CASE-STUDY:  $B_I$  = VALUES OF METRICS  $M_I$ .  $X_I, Y_I$  = PARAMETERS OF MEMBERSHIP FUNCTIONS FOR WHICH SATISFACTION LEVEL IS 0 AND 1 RESPECTIVELY. VALUES OF THESE PARAMETERS AND RELATIVE WEIGHTS ARE SELECTED BY REVIEWING THE PAST RELEASES OF THE PROJECT.

Readiness Dimensions	Readiness Criteria ( $C_i$ )	Weights	Metric Definition( $M_I$ )	$B_I$	Parameter of MFs		Satisfaction level (on release date)
					$X_I$	$Y_I$	
Implementation	Satisfaction of feature implementation	0.15	Feature implementation ratio	38.46	10	50	0.71
	Satisfaction of build/continuous integration trends	0.05	Percentage of successful builds/integration	62.34	30	80	0.65
	Satisfaction of implementation effort	0.05	Code Churn per contributor per day	150	50	200	0.67
Testing (Quality)	Satisfaction of defect finding	0.2	Defect find rate (per day)	0.22	1	0.1	0.86
	Satisfaction of defect fixing	0.1	Percentage of defect fixed	84.21	50	95	0.76
	Satisfaction of defect density	0.08	Defects/KLOC	7.78	12	3	0.47
	Satisfaction of unit test	0.1	Covered LOC/ total LOC	84.58	60	90	0.82
Source Code Quality	Satisfaction of codes smells	0.08	Number of code smells per class	0.6	1	0.25	0.53
	Satisfaction of code duplication	0.07	Percentage of duplicated code	22.73	40	10	0.58
	Satisfaction of method complexity	0.06	Average method complexity	8	20	5	0.80
Documentation	Satisfaction of readme file	0.06	Percentage of issues fixed	100	50	100	1.00

B. Data Collection and Modeling with RELREA

Following the proposed workflow, in step 1 important readiness criteria and metrics were defined from the four dimensions. Details of the selected criteria and metrics are presented in Table III. As the contributors of the project were not available, parameters of the membership function (piecewise linear functions) were defined by studying the values of the metrics from the previous releases.

Relative weights of the criteria were defined by two senior industry practitioners. Required data for calculating values of the metrics were collected from Github<sup>2</sup>, Travis-CI<sup>3</sup> and CodeClimate<sup>4</sup> to determine satisfaction levels of the readiness criteria according to the definition of membership functions. Finally, OWA operator was employed to determine the overall release readiness of *Publify 8.0* for three different decision strategies. Further data details can be found in [23].

C. Release Readiness Analysis of *Publify*

Using our proposed method, we computed the overall release readiness of *Publify 8.0* on six points in time. Figure 4 illustrates the impact of various decision strategies (degree of optimism) in overall release readiness. It shows that PR(170) is highest (0.81) when degree of optimism is considered to be 0.7 and lowest (0.65) when degree of optimism is 0.3. For the optimistic case, more importance is given to the three highly satisfied criteria (satisfaction of user manual(1.0), satisfaction of defect arrival(0.86), satisfaction of unit test coverage(0.82)) while less importance is given to the three low satisfied criteria (satisfaction of implementation effort (0.67), satisfaction of conditions integration(0.65), satisfaction of code duplication(0.58)). Similarly, for the pessimistic case, more importance is given to three low satisfied criteria (satisfaction of defect density (0.47), satisfaction of codes smells (0.53), and satisfaction of code duplication (0.58)) compared to the three highly satisfied criteria (satisfaction of user manual(1.0), satisfaction of defect finding(0.86), satisfaction of method complexity(0.80)).

As a result, product manager can identify the potential trade-off options among the readiness criteria. This kind of analysis of release readiness is helpful when time-to-release is important for the success of a software product. It allows identifying project parameters which need to be adjusted for improving overall release readiness.

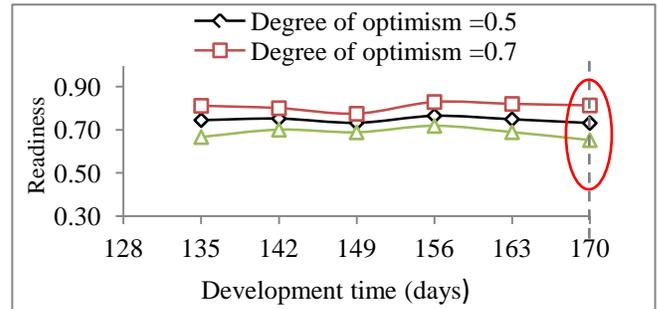


Figure 4. Impact of decision strategies on PR(170).

Figure 5 shows the readiness scores computed by RELREA at different stages of the release cycle as well as the projected readiness scores at  $t = 170$ . We concluded that achieving a targeted release readiness level of 0.9 is low and that the new version of *Publify 8.0* was potentially released too early. In support of this finding, already 15 bugs were reported within 12 days after the release date.

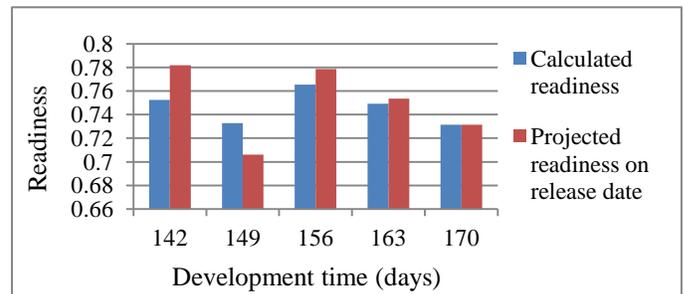


Figure 5: Calculated and projected readiness PR(170) of *Publify 8.0*.

## VI. THREATS TO VALIDITY

The key threats to the validity of the results are related to the selection of the release readiness criteria, the applied membership functions, the metrics selected, and the aggregation operator applied. As the proposed method allows defining release readiness criteria uniformly from four dimensions and metrics are defined in a goal oriented way, threats to the validity regarding selection process of readiness attributes are minimized.

One key threat to the construct validity is that the method only considers the release criteria which can be evaluated objectively. Subjective release readiness criteria (i.e. testing depth, user experience, amount of possible enhancements, etc.) are not considered in the method. However, inclusion of subjective criteria requires both human experts and more effort to support continuous evaluation of release readiness.

The accuracy of the calculated release readiness largely depends on the proper definition of the membership functions. Consideration of similar past project data and involvement of the development team will lead to more reliable membership functions. Selection of proper aggregation operator for combining the satisfaction levels of criteria is also a threat to the validity. As we choose a flexible aggregation operator which can be adjusted by the product release manager, this threat to the validity is at least reduced.

Another potential threat specific to the case-study is that the contributors of the projects were not reachable. Thus, we cannot claim that the selected readiness criteria, relative weights, metrics and parameters of the membership functions were truly relevant in the context of the project. However, to mitigate these issues, we studied the previous releases of the project and consulted with two senior software engineers for selecting the values of the required project specific parameters.

## VII. CONCLUSION AND FUTURE WORK

Release management is a decision-centric process with a number of criteria, stakeholders, and constraints involved in it. We have proposed an analytical approach to evaluate the release readiness at any point in time of the release interval and to make projection on its final status. Corrective steps can be taken to increase the current release readiness level of the software.

In future, this work will be extended to the analysis of likelihood of achieving the expected readiness before the predefined release date. The proposed method needs further analysis and evaluation of its applicability and usefulness. Stronger emphasis will be put on tuning of important project parameters. Integration of the framework with the existing project management tools such as JIRA, Team Foundation Server (TFS), and Github is intended to achieve an increase of its acceptability for industry practitioners.

## ACKNOWLEDGMENT

This research was partially supported by the Natural Sciences and Engineering Research Council of Canada, NSERC Discovery Grant 250343-12. Discussions with Trong

Tan Ho, S. M. Didar-Al- Alam, and Muhammad Rezaul Karim were helpful in conducting the project case study.

## REFERENCES

- [1] S. McConnell, "Gauging software readiness with defect tracking," *IEEE Software*, vol. 14, pp. 135-136, 1997.
- [2] J. T. S. Quah and S. W. Liew, "Gauging Software Readiness Using Metrics," in *IEEE Conference on Soft Computing in Industrial Applications*, 2008, pp. 426-431.
- [3] J. Fodeh, "Ready to Ship? – Using Release Readiness Metrics," in [http://208.254.39.65/qualtech/e\\_article000562072.cfm](http://208.254.39.65/qualtech/e_article000562072.cfm). Nov.2013.
- [4] R. Brettschneider, "Zero-Failure model -Is your software ready for release?," *Software, IEEE 6.4*, 1989.
- [5] A. Asthana and J. Olivieri, "Quantifying Software Reliability and Readiness," in *IEEE International Workshop Technical Committee on Communications Quality and Reliability*, 2009, pp. 1-6.
- [6] P. R. Satapathy, "Evaluation of Software Release Readiness Metric [0,1] across the software development life cycle " *Department of Computer Science & Engineering, University of California*, Riverside 2013.
- [7] R. R. Yager, "On ordered weighted averaging aggregation operators in multi-criteria decision making," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 18, pp. 183-190, 1988.
- [8] V. R. Basili, G. Caldiera, HD Rombach, and Rv Solingen, "The Goal Question Metric Approach". In *Encyclopedia of Software Engineering*. Marciniak J (ed.), vol. 1, pp. 578-83,2000.
- [9] L. A. Zadeh, "Fuzzy sets," *Information and Control*, vol. 8, pp. 338–353, 1965.
- [10] T. Bilgiç and I. B. Türkşen, "Measurement of membership functions: Theoretical and empirical work," *Fundamentals of fuzzy sets*, pp. 195-227, 2000.
- [11] M. Detyniecki, "Fundamentals on Aggregation Operators," in *Berkeley initiative in Soft Computing, Computer Science Division*: University of California, Berkeley, 2001.
- [12] T. Pearse, T. Freeman, and P. Oman, "Using Metrics to Manage the End-Game of a Software Project," in *Softw.Metrics Symp.*1999, pp. 207-215.
- [13] M. Staron, W. Meding, and K. Palm, "Release Readiness Indicator for Mature Agile and Lean Software Development Projects," *Agile Processes in Software Eng.and Extreme Progr.*, pp. 93-107, 2012.
- [14] "TeamInspector™," Last accessed March 14, 2014. in <http://techpubs.borland.com/bms/TeamInspector2008>.
- [15] "PTC integrity: Application Lifecycle Management (ALM)," Last accessed March 14, 2014. in <http://www.ptc.com/solutions/application-lifecycle-management/>.
- [16] O. S. Yamada, "Optimum Software-Release Time Considering an Error-Detection Phenomenon During Operation," *IEEE Trans. on Reliability*, vol. 99, pp. 596-599, 1990.
- [17] P. Musflek, W. Pedrycz, G. Succi, and M. Reformat, "Software cost estimation with fuzzy models," *ACM SIGAPP Applied Computing Review* 8, vol. 2, pp. 24-29, 2000.
- [18] W. A. Adnan and M. H. Yaacob, "An integrated neural-fuzzy system of software reliability prediction," in *Conference on Software Testing, Reliability and Quality Assurance*, 1994, pp. 154-158.
- [19] C. Temponia, J. Yenb, and W. A. Tiaob, "House of quality: A fuzzy logic-based requirements analysis," *European Journal of Operational Research* vol. 117, pp. 340-354, 1999.
- [20] R. R. Yager and A. Kelman, "An extension of the analytical hierarchy process using OWA operators," *Journal of Intelligent and Fuzzy Systems*, vol. 7, pp. 401-417, 1999.
- [21] Jrothman, "Release Criteria: Is This Software Done?," Last accessed March 14, 2014. in <http://www.jrothman.com/2002/03/release-criteria-is-this-software-done/>.
- [22] S. Medasania, J. Kimb, and R. Krishnapuram, " An overview of membership function generation techniques for pattern recognition " *Journal of Approximate Reasoning*, vol. 19, pp. 391–417, 1998.
- [23] Shahnewaz, " Analysis of Publify," Last accessed March 14, 2014. In <https://sites.google.com/site/shawniut/release-readiness>

# Change and Role as First-Class Abstractions for Realising Dynamic Evolution

Yin Chen, Xin-jun Mao

College of Computer, National University of Defense Technology, China

E-mail: chaanyean@163.com, mao.xinjun@gmail.com

## Abstract

*Today many software systems must undergo continuous dynamic evolution to adapt to variable environment while having minimal impact on their normal servicing. Programming dynamic evolution requires integrating evolution logic with business code, so that each single change as a stage of evolution can closely react to its context including objects and other concurrent changes. However, integrating evolution logic can lead to a closely coupled system which is less maintainable. In order to balance evolution logic integration and separation of concerns, so as to facilitate dynamic evolution at language level, we present a meta-model which takes change and role as first-class abstractions to model different degrees of such integration. It uses roles to model variation points and uses role enactment to realise changes in a modular and interactive manner. We extend conventional languages with primitives for describing dynamic changes and with atomic operations for manipulating changes according to the structure of target system. Our approach will hopefully facilitate the realisation of dynamic changes and well-structured, highly evolvable systems.*

## 1. Introduction

A deployed software system should evolve continuously to react to contextual and requirement changes. For many real world systems which need to be highly available, such evolution should be performed dynamically to reduce impacts on the system's normal servicing [3]. Evolution process is made up of phased changes, whose programming involves two aspects: (1) *strategy of change*, i.e., the systematic process (or its pattern) to realise a change, and (2) *target of change*, i.e., objects involved in the change. Effectively supporting these aspects at language level by explicit representation of change-related information relieves the effort of programming dynamic evolution [4]. Programs specifying targets of change should previously arrange *variation points* (as the join points of evolution logic and application

logic) which clarify the dynamic structure of target system and facilitate its manipulation. Strategies of change then exploit these variation points to realise final changes.

Many efforts are made to enable programming dynamic evolution. Architecture description languages (ADL) [1] specify dynamic software architecture with graphs, process algebras, etc. Aspect-oriented programming (AOP) encapsulates evolution concerns as *aspects* and realises fine-grained evolution actions [5]. They are basically *external* approaches [6] that enhance separation of concerns by encouraging application-independent evolution mechanisms rather than intertwining them with application code. However, due to contextual complexity, change strategies must be application-dependent and closely integrated with target systems, because: (1) instead of designing all-powerful change strategies, many changes must be temporarily designed according to runtime situation; (2) in dynamic evolution, a change strategy needs to adapt to its context, and concurrent changes need to interact for synchronisation and data sharing, so as to ensure global requirements and minimal side effects. Context-oriented programming (COP) [7] and dynamically typed languages [4] integrate evolution capability at language level, enabling dynamic activation of codes via contextual events, but at the cost of separation of concerns and global coordination.

In order to facilitate programming of dynamic changes, we propose an approach to explicitly represent evolution from both *external* and *internal* perspectives with an attempt to balance and separate evolution concerns. We introduce *role* as first-class entity to model *internal* variation points and to combine evolution concern with target system. Roles enable a separation of concerns between business logic, and coordination or evolution logic, and promote a global view of target system [2]. We introduce *change* as first-class entity to model *external* change strategies. Changes are deployed according to the role-based structure of target system and manipulate roles for their realisation. We propose explicit primitives for programming these concepts and atomic operations for bridging and manipulating *internal* and *external* perspectives of dynamic evolution.

The rest of this paper is organised as below. Section 2 presents the concepts for describing strategies and targets of changes. Section 3 presents atomic operations for specifying role-based interaction during change realisation, and the way these operations are organised into a concrete change process. Conclusion and outlook are outlined in section 4.

## 2. Meta-Model for Dynamic Evolution

We present a meta-model which provides following concepts for describing dynamically evolvable system: *organ*, *role*, *change*, and *message*, respectively modelling dynamically evolvable objects, variation points, change strategies, and information exchanged in interactive change realisation. Fig. 1 shows these concepts and their relations.

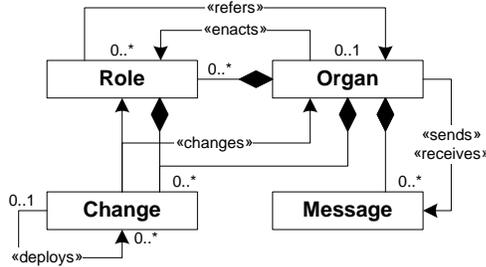


Figure 1. Meta-model for dynamic evolution

1. **Organ:** An *organ* is an evolvable object attached with changes and roles. Each organ  $\alpha$  can be represented by tuple  $\langle R, M, C, \sigma, \varsigma, R' \rangle$ .  $R$  is the set of roles enacted by  $\alpha$ .  $M$  is a queue of messages received but not yet processed by  $\alpha$ .  $C$  is the set of changes deployed to  $\alpha$ .  $\sigma$  is an assignment which determines the values of  $\alpha$ 's field.  $\varsigma$  is the sequence of statements to be executed by  $\alpha$  at the current time.  $R'$  is a set of roles owned by  $\alpha$  to be enacted by other organs.

2. **Role:** A *role* is a reference to an organ which plays a part in another organ owning the role. Each role  $\rho$  can be represented by tuple  $\langle \alpha, \kappa, C, \alpha' \rangle$ .  $\alpha$  is the unique organ which enacts  $\rho$ . Its value is  $\perp$  if no organ is currently enacting  $\rho$ . Each role is allowed to have at most one enactor, and is syntactically taken just as a reference to that enactor.  $\kappa$  is the class specifying the behaviour and properties that can be implemented by  $\alpha$ .  $C$  is the set of changes deployed to  $\rho$  and  $\alpha$ .  $\alpha'$  is the unique organ that owns  $\rho$ . We use  $\text{in}(\rho)$  (or  $\text{out}(\rho)$ ) to represent the algorithm which determines whether an organ satisfies the requirements for entering (or quitting)  $\rho$ , and then reacts correspondingly.

For an organ  $\alpha = \langle R, M, C, \sigma, \varsigma, R' \rangle$ , roles in  $R$  decompose  $\alpha$ 's tasks to the organs enacting these roles. Each role  $\rho$  specifies entrance (or quitting) constraints for its enactor  $\alpha_\rho$  using  $\text{in}(\rho)$  (or  $\text{out}(\rho)$ ) and attaches methods (defined in  $\kappa$ ) to  $\alpha_\rho$  for it to interact with other roles in  $\alpha$ . In terms of

change realisation, roles provide variation points that combines organs in different ways, regulate changes using these constraints, and specify protocols about how their enactors coordinate to realise a change.

3. **Change:** A *change* is an executing entity which is deployed to the target—an organ  $\alpha$  (or a role  $\rho$ )—to make change to  $\alpha$  (or  $\rho$ 's enactor) and which can deploy more changes. One instance of a change can occupy multiple threads. Each change  $\gamma$  can be represented by tuple  $\langle \iota, \sigma, \varsigma, C, \gamma' \rangle$ .  $\iota$  is an identifier of the target to which  $\gamma$  is deployed.  $\sigma$  is the assignment which determines parameters of  $\gamma$ .  $\varsigma$  is the statement to be executed by  $\gamma$ .  $C$  is the set of changes which are deployed by and are hence subordinate to  $\gamma$ .  $\gamma'$  is the change which has deployed  $\gamma$ . We use  $\text{init}(\gamma)$  to represent the algorithm executed by  $\iota$  to initialise  $\gamma$ . We use  $\text{chk}(\gamma)$  to represent the algorithm executed in a top-down manner after  $\text{act}(\gamma)$ 's completion to check whether the designed goal of  $\gamma$  has been achieved. The execution of  $\text{act}(\gamma)$  and  $\text{chk}(\gamma)$ , and the handling of their results, are automatically performed by the underlying framework.

4. **Message:** A *message* is an object used for asynchronous interaction among organs. Each message  $\mu$  can be represented by tuple  $\langle \alpha, \iota, \nu, \sigma \rangle$ .  $\alpha$  is the organ which has sent  $\mu$ .  $\iota$  is the identifier of the organ or role to which  $\mu$  is sent.  $\nu$  is the local name of  $\mu$  assigned by  $\alpha$ .  $\sigma$  is an assignment which determines the value of  $\mu$ 's each parameter. Both  $\nu$  and  $\sigma$  are used by  $\alpha'$  to locate the method or the statement (within a method) that handles  $\mu$ .

Fig. 2 shows how introducing *role* and *change* as first-class abstraction into object-oriented paradigm facilitates programming dynamic evolution. As shown by path **A**, in terms of change realisation, roles provide *variation points* that combines organs in different ways, so that simply by enacting a different role can an organ realise a change. These variation points can relate to business logic (which concerns how each *organ* functions or multiple *organs* coordinates, as shown by path **B**) or evolution strategy (which concerns how the *changes* exploits variation points and interact with each other to achieve their goals, as shown by path **C**). Role enactments provide managed variation points, as shown by **D**. Both concerns involves two aspects: *coordination*, that focuses on how multiple individuals work together to achieve system-level goals, and *execution*, that focuses on how each individual performs some low level operations to achieve local goals. About the coordination aspect: each role  $\rho$  specifies entrance (or quitting) constraints for its enactor  $\alpha_\rho$  using  $\text{in}(\rho)$  (or  $\text{out}(\rho)$ ) and attaches methods (defined in  $\kappa$ ) to  $\alpha_\rho$  for it to interact with other roles in  $\alpha$ . About the execution aspect: roles enable each organ to combine different services provided by other organs by delegating tasks to them through role enactment. The loop formed by pathes **A**, **C**, **D**, and also the fact that one change can deploy other changes, indicate that there are infinite possi-

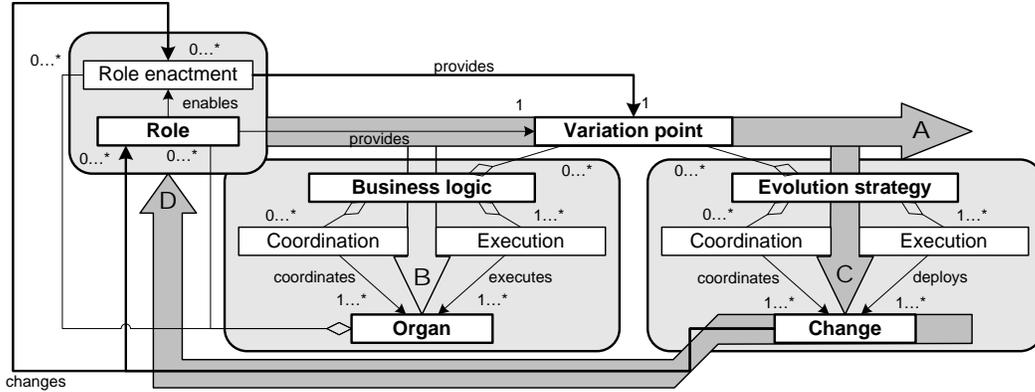


Figure 2. Applying roles and changes as first-class abstraction

bilities for changes to be used to implement *meta-changes*, that is, the changes made to the way how a part of the system is changed.

### 3. Change-Realising Atomic Operations

In a typical process for realising the changes using our meta-model, each single change needs to undergo a 4-phase process to be realised: (1) get *deployed* to the target object, (2) *change* the target object, (3) *check* whether the designed goal has been achieved, and deal with exceptions, (4) *report* the result of the change to who has deployed the change. Such process is usually iterative. For example, if (4) finds the goal unachieved, it may reconfigure and restart the change. This process can have multiple stages of preparation for completion. For example, some preparatory changes can be implemented to be used later and be dismissed once the final goal is achieved. Tasks for changing a higher level organ is decomposed and delegated to the subordinate organs through the “deploy” links. After the subordinate organs have completed and checked the changes, they report back to the higher level organ. Following syntax shows atomic operations used to realise such change-realising process.

$$\begin{array}{l}
 \zeta ::= \varepsilon \mid \zeta_{atom}; \zeta \mid \zeta \parallel \zeta \\
 \zeta_{atom} ::= \iota.\nu(\sigma) \mid \mathbf{rcv}\{\iota_i.\nu_i(\sigma_i) : \zeta_i\}_{i \in I} \mid \\
 \mathbf{dply}(\gamma, \iota) \mid \mathbf{chk}(\gamma) \mid \\
 \mathbf{in}(\rho)(\alpha) \mid \mathbf{out}(\rho)(\alpha) \mid \\
 \mathbf{hire}(\rho)(A) \mid \mathbf{fire}(\rho) \mid \\
 \zeta_{atom, ori}
 \end{array}$$

where  $\zeta$  denotes a statement which can be formed by a sequence of atomic statements or the parallel combination of multiple statements (e.g.,  $\zeta_1 \parallel \zeta_2$ ) which represents concurrent threads.  $\zeta_{atom, ori}$  denotes a statement defined by the original language, e.g., Java. Our extension provides 3 sets

of atomic operations: *message passing*, *change manipulation*, and *role manipulation*. Fig. 3 presents the operational semantic rules of atomic operations for guiding design of dynamic evolution enabling languages, in terms of effects of these operations on configurations of *organs*, *roles*, or *changes* involved.

### 4. Conclusion and Outlook

We present a meta-model for describing a dynamically evolvable system, and language-independent atomic operations for manipulating a system complying with this model. Our approach balances between *external* and *internal*. It uses roles to manipulate the structure of target system. It provides language-level primitives to describe the strategies of change (i.e., *change* classes). Compared with COP, which uses *layer* to modularise dynamically switchable behaviour, we use *role* as a medium between target system and strategies of change, to add variation points encapsulating behavioural extension of an *organ*, inter-organ relations, and even application/evolution logic. Both changes and roles can be used for dynamic evolution. Changes are *externally* built, deployed to roles and organs, forming a network alongside role enactment network, exploiting roles to achieve their goal. Roles are relatively *internal* and more closely bound to the business logic. Organising changes with roles facilitates the global coordination of distributed changes. Using explicit change manipulation rather than implicit invocation of these additional concerns (as in AOP) facilitates the anticipation of the result of change. We plan to develop a framework supporting features of our model and language-level primitives, and provides facilities for the management, serialisation, and exception handling, of distributed runtime changes.

$$\begin{array}{c}
\frac{\mu = \langle \alpha_1, \iota, \nu, \sigma \rangle, (\iota = \alpha_2 \vee \iota = \rho \in R_2 \vee \iota = \perp)}{\alpha_1 = \langle R_1, M_1, C_1, \sigma_1, \iota, \nu(\sigma); \varsigma_1, R'_1 \rangle, \alpha_2 = \langle R_2, M_2, C_2, \sigma_2, \varsigma_2, R'_2 \rangle \rightarrow \alpha_1 = \langle R_1, M_1, C_1, \sigma_1, \varsigma_1, R'_1 \rangle, \alpha_2 = \langle R_2, \mu \cdot M_2, C_2, \sigma_2, \varsigma_2, R'_2 \rangle} \text{(Send)} \\
\frac{j \in I, \mathbf{t}(\sigma_j) \prec \mathbf{t}(\sigma), \mu = \langle \alpha_0, \alpha, \nu_j, \sigma_j \rangle, (\iota_j = \alpha_0 \vee \iota_j = \rho \in R \vee \iota_j = \perp)}{\alpha = \langle R, M \cdot \mu \cdot M', C, \sigma, \mathbf{recv}\{\iota_i! \nu_i(\sigma_i) : \varsigma_i\}_{i \in I}; \varsigma, R' \rangle \rightarrow \alpha = \langle R, M \cdot M', C, \sigma, \varsigma_j; \varsigma, R' \rangle} \text{(Recv)} \\
\text{where for organ } \alpha \text{ (or role } \rho = \langle \alpha, \kappa, C, \alpha' \rangle \text{), } \mathbf{t}(\alpha) \text{ (or } \mathbf{t}(\rho) \text{) returns the type of } \alpha \text{ (or } \rho \text{).} \\
\frac{\iota = \alpha \vee \iota = \rho \in R}{\gamma = \langle \iota, \sigma_\gamma, \mathbf{dply}(\gamma_\alpha, \iota); \varsigma, C, \gamma' \rangle, \alpha = \langle R, M, C, \sigma, \varsigma, R' \rangle \rightarrow \gamma = \langle \iota, \sigma_\gamma, \varsigma, C \cup \{\gamma_\alpha\}, \gamma' \rangle, \alpha = \langle R, M, C \cup \{\gamma_\alpha\}, \sigma, \mathbf{init}(\gamma_\alpha); \varsigma, R' \rangle} \text{(Dply)} \\
\frac{\gamma \in C_0, \gamma \neq \gamma_0}{\gamma_0 = \langle \iota_0, \sigma_0, \chi_0 := \mathbf{chk}(\gamma_0); \mathbf{exc}(\gamma_0, \chi_0); \varsigma'_0 \parallel \varsigma_0, C_0, \gamma' \rangle, \gamma = \langle \iota, \sigma, \varsigma, C, \gamma_0 \rangle \rightarrow \gamma_0 = \langle \iota_0, \sigma_0, \mathbf{recv}\{\iota! \mathbf{rtn}(\chi) : \chi_0 := \chi\}; \mathbf{chk}(\gamma_0); \mathbf{exc}(\gamma_0, \chi_0); \varsigma'_0 \parallel \varsigma_0, C_0, \gamma' \rangle, \gamma = \langle \iota, \sigma, \chi := \mathbf{chk}(\gamma); \mathbf{exc}(\gamma, \chi); \iota_0. \mathbf{rtn}(\chi) \parallel \varsigma, C, \gamma_0 \rangle} \text{(Chk)} \\
\frac{\sigma'(\chi) = \mathbf{true}, \kappa \prec \mathbf{t}(\alpha)}{\rho = \langle \perp, \kappa, C, \alpha' \rangle, \alpha = \langle R, M, C, \sigma, \chi := \mathbf{in}(\rho)(\alpha); \varsigma, R' \rangle \rightarrow \rho = \langle \alpha, \kappa, C, \alpha' \rangle, \alpha = \langle R \cup \{\rho\}, M, C, \sigma', \varsigma, R' \rangle} \text{(In)} \\
\text{where } \kappa_1 \prec \kappa_2 \text{ means class } \kappa_1 \text{ is a superclass of class } \kappa_2. \\
\frac{\sigma'(\chi) = \mathbf{true}, \rho \in R}{\rho = \langle \alpha, \kappa, C, \alpha' \rangle, \alpha = \langle R, M, C, \sigma, \chi := \mathbf{out}(\rho)(\alpha); \varsigma, R' \rangle \rightarrow \rho = \langle \perp, \kappa, C, \alpha' \rangle, \alpha = \langle R - \{\rho\}, M, C, \sigma', \varsigma, R' \rangle} \text{(Out)} \\
\frac{\kappa \prec \mathbf{t}(\alpha), \alpha \in A, \rho = \langle \perp, \kappa, C, \alpha_0 \rangle \in R'_0}{\alpha_0 = \langle R_0, M_0, C_0, \sigma_0, \mathbf{hire}(\rho)(A); \varsigma_0, R'_0 \rangle, \alpha = \langle R, M, C, \sigma, \varsigma, R' \rangle \rightarrow \alpha_0 = \langle R_0, M_0, C_0, \sigma_0, \varsigma_0, R'_0 \rangle, \alpha = \langle R, M, C, \sigma, \chi := \mathbf{in}(\rho)(\alpha); \varsigma, R' \rangle} \text{(Hire)} \\
\frac{\rho = \langle \alpha, \kappa, C, \alpha_0 \rangle \in R'_0}{\alpha_0 = \langle R_0, M_0, C_0, \sigma_0, \mathbf{fire}(\rho); \varsigma_0, R'_0 \rangle, \alpha = \langle R, M, C, \sigma, \varsigma, R' \rangle \rightarrow \alpha_0 = \langle R_0, M_0, C_0, \sigma_0, \varsigma_0, R'_0 \rangle, \alpha = \langle R, M, C, \sigma, \chi := \mathbf{out}(\rho)(\alpha); \varsigma, R' \rangle} \text{(Fire)}
\end{array}$$

Figure 3. Operational semantics of change realising atomic operations

## 5 Acknowledgement

This research is funded by NSFC under granted number 61070034 and 61379051, Program for New Century Excellent Talents in University under granted number NCET-10-0898, and Open fund of State Key Laboratory of Software Development Environment under granted number SKLSDE-2012KF-0X.

## References

- [1] J. S. Bradbury, J. R. Cordy, J. Dingel, and M. Wermelinger. A survey of self-management in dynamic software architecture specifications. In *Proceedings of the 1st ACM SIGSOFT workshop on Self-managed systems*, pages 28–33. ACM, 2004.
- [2] G. Cabri, L. Leonardi, L. Ferrari, and F. Zambonelli. Role-based software agent interaction models: a survey. *The Knowledge Engineering Review*, 25(04):397–419, 2010.
- [3] M. Ghafari and M. M. P. Kallehbasti. Dynamic update of distributed systems. *Self-Adaptive Software Systems*, page 1, 2013.
- [4] C. Ghezzi, M. Pradella, and G. Salvaneschi. An evaluation of the adaptation capabilities in programming languages. In *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 50–59. ACM, 2011.
- [5] P. Greenwood and L. Blair. Using dynamic aspect-oriented programming to implement an autonomic system. In *Proceedings of the 2004 Dynamic Aspects Workshop (DAW04), Lancaster*, pages 76–88, 2004.
- [6] M. Salehie and L. Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 4(2):14, 2009.
- [7] G. Salvaneschi, C. Ghezzi, and M. Pradella. Context-oriented programming: A software engineering perspective. *Journal of Systems and Software*, 85(8):1801–1817, 2012.

# Feature-Level Change Impact Analysis Using Formal Concept Analysis

Hamzeh Eyal-Salman, Abdelhak-Djamel Seriai, Christophe Dony

UMR CNRS 5506, LIRMM, University of Montpellier 2 for Sciences and Technology, France

E-mail: {Eyalsalman, Seriai, Dony}@lirmm.fr

## Abstract

*Software Product Line Engineering (SPLE) is a systematic reuse approach to develop a short time-to-market and quality products, called Software Product Line (SPL). Usually, the SPL is not developed from scratch but it is developed by reusing features (resp. their source code elements) of existing similar systems developed by ad-hoc reuse techniques. The feature implementations may be changed for adapting SPLE context. The change may impact other features that are not interested in the change, as a feature's implementation spans multiple code elements and shares code elements with other features. Therefore, feature-level Change Impact Analysis (CIA) is needed to predict affected features for change management purpose. In this paper, we propose a feature-level CIA technique using formal concept analysis. In our experimental evaluation using three case studies of different domains and sizes, we show the effectiveness of our technique in terms of the most commonly used metrics on the subject.*

**Keywords:** *Impact analysis, feature, source code, FCA, software product line, product variants.*

## 1 Introduction

SPLE is a engineering discipline providing methods to promote systematic software reuse for developing short time-to-market and quality products in a cost-efficient way [1]. These products are referred to as SPL [1]. The whole idea behind SPLE is to build core assets consisting of all reusable software artifacts (such as source code, test cases and son) that can be leveraged to develop SPL's products. Building such core assets is driven by features that SPL products should provide. A feature is "a prominent or distinctive user-visible aspect, quality or characteristic of a software system" [2].

Building SPL's core assets from scratch is a costly task [1]. Therefore, features (resp. their source code elements) of existing similar systems developed by ad-hoc reuse techniques should be reused as much as possible to build SPL's core assets [3]. Such systems are known as *product variants*. The implementation of the obtained fea-

ture(s) may need to be changed for adapting SPLE context by adding or removing requirements (resp. their source code elements) [4]. The change may impact other features that are not interested in the change, as a feature's implementation spans multiple code elements (e.g., classes and methods) and shares code elements with other features. To avoid such a situation, feature-level Change Impact Analysis (CIA) is needed to detect introducing undesirable interactions between feature implementations. Furthermore, it is helpful to conduct change management from a SPL manager's point of view. For example, managers may most likely be interested in evaluating a given source code change in terms of affected features, in order to decide which change strategy should be executed.

Feature-level CIA is far from a trivial task when we have a large number of features. Manually tracing feature implementations to determine affected features is time-consuming, error-prone and tedious. The CIA is seldom considered at the feature level for changes made to the source code level. Most of the existing works performs CIA at the source code level with little works completed at requirement and design levels [5]. In this paper, we propose a technique to study CIA at the feature level using Formal Concept Analysis (FCA). This technique takes, as input, a change set composed of classes to be changed and computes, as output, a ranked list of affected features. Each feature in this list has a probability to be affected representing the feature priority to be checked by maintainers. Additionally, we propose two metrics to measure to what degree a given feature implementation is impacted and the changeability of the features.

The rest of this paper is organized as follows. Section 2 presents FCA. Sections 3 and 4 present the proposed approach and experimental evaluation, respectively. Next, sections 5 and 6 discuss the related works and conclude the paper respectively.

## 2 Background: Formal Concept Analysis

FCA is a technique for data analysis and knowledge representation based on lattice theory [6]. Concept lattices are core structures of FCA for extracting a set of concepts from

**Table 1. A formal context for birds.**

	Flying	Nocturnal	Feathered	Migratory	with-crest	with-membrane
Flying-squirrel	X					X
Bat	X	X				X
Ostrich			X			
Flamingo	X		X	X		
Chicken	X		X		X	

a dataset, called a formal context, composed of objects described by attributes. A formal context is defined as a triple  $K = (O, A, R)$  where  $O$  is a set of objects,  $A$  is a set of attributes and  $R$  is a binary relation between objects and attributes indicating which attributes are possessed by each object. Table 1 presents an example of a formal context for several animals described by their characteristics.

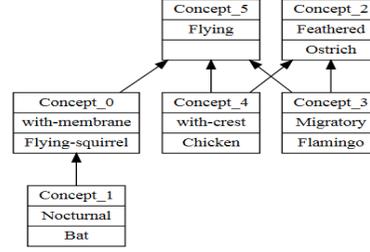
A formal concept is a pair  $(E, I)$  composed of an object set ( $E \subseteq O$ ) and an attribute set ( $I \subseteq A$ ).  $E$  is the extent of the concept (i.e., the objects covered by the concept).  $I$  is the intent of the concept (i.e., the attributes shared by the objects covered by the concept). For example,  $(\{Chicken, Flamingo\}, \{Flying, Feathered\})$  is a concept of our exam-

ple. Given a formal context  $K = (O, A, R)$ , and two formal concepts  $C_1 = (E_1, I_1)$  and  $C_2 = (E_2, I_2)$  of  $K$ , the concept specialization order ( $\leq_s$ ) is defined by  $C_1 = (E_1, I_1) \leq_s C_2 = (E_2, I_2)$  if and only if  $E_1 \subseteq E_2$  (and equivalently  $I_2 \subseteq I_1$ ).  $C_1$  is called a sub-concept of  $C_2$ .  $C_2$  is called a super-concept of  $C_1$ . Based on this specialization order definition, an important property is that a sub-concept inherits in a top-down manner the attributes (intent) of its super-concepts, while a super-concept inherits in a bottom-up manner the objects (extent) of its sub-concepts.

Let  $C_K$  be the set of all concepts of a formal context  $K$ . This set of concepts provided with the specialization order ( $C_K, \leq_s$ ) has a lattice structure, and is called the concept lattice associated with  $K$ . Figure 1 shows the concept lattice built for the formal context of Table 1.

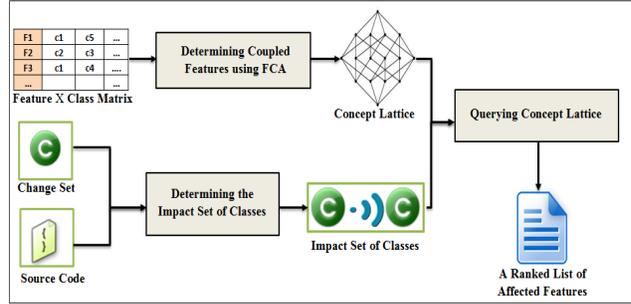
### 3 The Proposed Approach

When source code classes that implement a feature are changed, the change may be propagated to the neighbor classes which the changed classes are coupled to. As a feature's implementation may span multiple classes and features may have shared classes, the change may lead to impact the implementation of other features. Therefore, we should determine the impact set of classes and coupling relations between features to determine affected features. We rely on structural and feature couplings to support these purposes respectively. Structural coupling refers to interdependencies between classes, such as inheritance, method invocation, etc. Feature coupling is the degree to which the source code elements implementing a feature (e.g., meth-



**Figure 1. The concept lattice for the formal context of Table 1.**

ods, attributes, classes) depend on elements outside the feature [7]. Figure 2 gives an overview of our approach. This approach relies on three main steps: (i) computing the impact set of classes for source code changes, (ii) Determining coupled features using FCA, (iii) querying the generated concept lattice to compute a ranked list of affected features. The goal of the first step is to determine impact set classes due to modifying a given change set of classes. Of course, the impact set also includes the change set members. The second step aims to determine coupled features by building the concept lattice. This lattice is queried in the third step using the impact set computed in the first step to find a ranked list of affected features.



**Figure 2. Main steps of our CIA approach.**

#### 3.1 Determining the Impact Set of Classes

Analyzing the interdependencies between classes helps to determine the coupled classes, and hence determine the impact set of classes. We rely on the following interdependencies that represent coupling aspects in object-oriented applications supporting CIA: (1) *inheritance relationship*: when a class inherits attributes and methods of another class, (2) *method call*: when a method of one class calls a method of another class, (3) *attribute access*: when a class accesses an attribute of another class, (4) *shared attribute access*: when two classes access the same attribute of another class. To capture these interdependencies, the given source code is statically analyzed by building an abstract syntax tree (AST) that can be queried to extract required information.

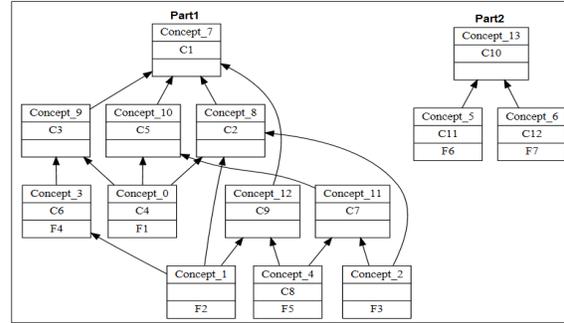
**Table 2. The formal context of features and classes.**

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12
F1	X	X	X	X	X							
F2	X	X	X			X				X		
F3	X	X			X		X					
F4	X		X			X						
F5	X				X		X	X	X			
F6										X	X	
F7										X		X

### 3.2 Determining Coupled Features Using FCA

In this step, we rely on FCA to analyze coupling relations between a given set of features. FCA allows us to determine and visualize source code classes that are shared among all features, among a subset of features and those that are specific to each feature through the hierarchical organization of the concept lattice. This step takes as input a feature-to-class traceability matrix. This matrix can be obtained by our previous work [8]. In this matrix, each feature is linked to its implementing classes. Columns and rows respectively represent features and source code classes. This matrix represents the formal context where features and classes respectively represent objects and attributes. The relation between an object and an attribute refers to which feature is implemented by which class. According to this definition of the formal context, we can obtain a concept lattice containing concepts that are composed of a set of features sharing a set of classes. Such a lattice represents dependencies between features and classes (feature coupling). Table 2 is an example of the formal context to be analyzed where objects are  $\{F1, F2, F3, F4, F5, F6, F7\}$  while attributes are  $\{C1, C2, C3, C4, C5, C6, C7, C8, C9, C10, C11, C12\}$ . Figure 3 shows the corresponding lattice of the context shown in Table 2. Based on this lattice we discover the following observations:

- The impact of changes made to classes of concepts that are located at the top of the lattice is propagated to all extents (features) of these concepts. This is because the classes of such concepts are shared among all or most lattice concepts. Such classes should not be impacted as far as possible because they may lead to the risk of changing all features. For example, if  $C1$  and  $C10$  respectively at  $Concept_7$  and  $Concept_{13}$  are modified or impacted, all features ( $F1$  to  $F7$ ) will be affected.
- The impact of changes made to classes of lattice concepts located at the bottom is local. For example, if  $C8$  is modified or impacted, only  $F5$  will be affected. Determining these classes is useful to guide the maintainers to choose from available change strategies, the one that considers only such classes to implement the change request.



**Figure 3. The concept lattice for the formal context of Table 2.**

- By descending vertically throughout the lattice, the impact of changes is gradually decreased. For example, if changes are made to  $\{C1\}$ , the set of affected features will be composed of  $\{F1, F2, F3, F4, F5\}$ , but if changes are made to  $\{C7\}$ , the set of affected features will only be composed of  $\{F3, F5\}$ .
- Lattice concepts that are downwardly reachable from jointly changed classes have a high probability of being affected. For example, assuming that impact set =  $\{C2, C3, C5\}$ , then  $F1$  has a higher probability of being affected than  $F2$ . This is because  $F1$  will be affected by three joint classes  $\{C2, C3, C5\}$ , while  $F2$  will be affected by two joint classes  $\{C2, C3\}$ .
- Based on the generated lattice, we can determine isolated sub-systems/parts and hence determine maintenance resources that are required based on the affected sub-systems/parts. For example, if the affected features are  $F6$  and  $F7$  (i.e., part2) and they are maintained by specific team, a product manager can exclude other maintenance teams and ask only the interested team to execute the change.

### 3.3 Querying Concept Lattice

#### 3.3.1 Determining Affected Features

In this step, we query the generated lattice by the impact set of classes to retrieve features that are implemented by these classes. The retrieved features represent two subsets of affected features, following the two steps below. Firstly, locating a set of lattice concepts that have as intent (excluding inherited intent) one or more of impacted classes. The extents of these concepts are a subset of affected features. Secondly, determining all downwardly reachable concepts from concepts obtained in the step 1. The extents of these concepts are another subset of affected features.

The first step is performed using a simple algorithm to explore the concept lattice for determining the required concepts, called ( $CON$ ). Due to space limitation, we are unable to present this algorithm. To perform the second step, we

propose algorithm 1. This algorithm is based on depth-first search (DFS). The algorithm takes as input a list of concepts computed in step 1 ( $CON$ ) and the concept lattice to be queried ( $CL$ ). Lines 1-10 check each concept in  $CON$  in turn so that each one becomes the root of a new tree in the DFS. The extents of concepts that constitute such a tree represent all affected features due to changes made to classes of its parent concept. The function  $getAdjacentConcepts()$  returns all concepts that are located immediately below and directly related to the current concept ( $Co$ ).  $LC_F$  is an accumulator for all traversed concepts. These traversed concepts represent all downwardly reachable concepts from  $CON$ 's concepts.  $LC_F$  may contain some concepts that do not have their own extent and  $LC_F$  also can include redundant concepts due to the overlap between DFS's trees. Therefore, lines 11-14 remove these concepts respectively using the functions ( $RemovingConHavingEmptyExt()$ ) and ( $RemovingRedundantConcepts()$ ). In line 15, we extract the extent of  $LC_F$  concepts using the function ( $ExtractingExtent()$ ) because the extent of these concepts represent the affected features.

### 3.3.2 Ranking Affected Features

Based on the observations of concept lattice mentioned earlier, we can deduce that the concept lattice organizes features hierarchically according to their probability to be impacted by a given change proposal. Therefore, we propose two metrics adapted to feature-level CIA: *Impact Probability Metric (IDM)* and *Changeability Assessment Metric (CAM)*.

IDM is used to measure the degree to which a specific feature can be affected. Features having high IDM values, the functional requirements provided by these features have a high probability to be affected. Therefore, their implementation should first be checked by maintainers. IDM values are in the range [0, 1]. Using IDM metric, we can rank the affected features from the feature that has a higher IDM value to the feature that has a lower IDM value. We propose the following equation for IDM:

$$IDM(F) = \frac{|\{I\} \cap \{Impact\ Set\}|}{|\{I\}|} \times 100\% \quad (1)$$

In Equation 1,  $F$  is an affected feature while  $I$  is the intent (classes) of a lattice concept having  $F$  as an extent and also includes intents inherited in a top-down manner. Thus, we need to compute the inherited intents of lattice concepts ( $LC_F$ ) having the affected features. This is performed using the DFS algorithm to compute all upwardly reachable concepts from each concept in  $LC_F$ .

CAM is a metric for determining the percentage of features that are affected by a given change. It describes the changeability of all features in order to help product managers to decide whether a change proposal is accepted or

---

#### Algorithm 1: LocatingAffectedFeatures

---

**Input:**  $CON, LC$   
**Output:** AF // list of affected features

```

1  $LC_F \leftarrow \phi$ 
2 foreach  $j$  from 1 to  $|CON|$  do
3    $ConceptStack \leftarrow CON[j]$ 
4   while ( $ConceptStack$  is not empty) do
5      $Co \leftarrow pop(ConceptStack)$ 
6      $AdjCos \leftarrow getAdjacentConcepts(Co, CL)$ 
7     if ( $AdjCos$  is empty) then
8        $LC_F \leftarrow Co$ 
9     else
10       $push(AdjCos), LC_F \leftarrow Co$ 
11 foreach  $i$  from 1 to  $|LC_F|$  do
12   if  $Extent(LC_F[i])$  is empty then
13      $RemovingConHavingEmptyExt(LC_F, i)$ 
14  $RemovingRedundantConcepts(LC_F)$ 
15  $AF \leftarrow ExtractingExtent(LC_F)$ 
16 return AF
```

---

to find another change plan more suitable to employ. We propose the following equation for CAM:

$$CAM = \frac{\#Affected\_Features}{\#All\_Features} \times 100\% \quad (2)$$

In equation 2,  $Affected\_Features$  represent a set of features that are potentially affected by a given impact set of classes.  $All\_Features$  represent all features. CAM values take a range [0, 1]. If the computed CAM value is high, this means that features (resp. their implementation) are more sensitive for a given change proposal and vice versa.

By referring again to Figure 3 and considering that the impact set is composed of  $\{C3, C5, C6\}$ , we find out that the IDM and CAM values of this impact set as shown in Table 3. The columns ( $Concept\_No$ ,  $Features$  and  $Rank$ ) show respectively a set of concepts having affected features, their affected features and the priority of these features to be checked. These features are ranked based on their IDM values. From Table 3, we notice that  $F4$  has the highest IDM value. Also, It shows that CAM for this impact set is equal to (71%). This means that most of features will be affected by this given impact set of classes.

## 4 Experimental Results and Analysis

### 4.1 Case Studies

For evaluation, we have applied our CIA technique respectively to core assets of three different case studies:

**Table 3. Impact results for {C3, C5, C6} changes.**

Concept_No	Features	$ \{I\} $	$ \{I\} \cap \{FCS\} $	IDM	Rank	CAM
Concept_3	F4	3	2	66%	1	71%
Concept_1	F2	5	2	40%	2	
Concept_0	F1	5	2	40%	2	
Concept_2	F3	4	1	25%	3	
Concept_4	F5	5	1	20%	4	

*ArgoUML-SPL*<sup>1</sup>, *MobileMedia*<sup>2</sup> and *BerkelyDB-SPL*<sup>3</sup>. *MobileMedia* is a small-scale system for managing multimedia files on mobile devices. The core assets of *MobileMedia* supports 6 features such as, *view photo*, *delete photo*, *sort photos*, etc. *ArgoUML-SPL* is a large-scale system for UML modeling tool. The core assets of *ArgoUML-SPL* supports 8 features such as, the *Class* diagram, the *State* diagram, the *Activity* diagram, etc. *BerkeleyDB-SPL* is a large-scale embedded database system that can be embedded in other applications as a storage engine. The core assets of *BerkeleyDB-SPL* provides 25 features, such as *transaction management*, *concurrency control*, etc.

## 4.2 Evaluation Measures

We relied on three measures inspired from information retrieval, namely *precision*, *recall* and *F-measure* to evaluate our CIA technique. *Precision* measures the accuracy of estimated impact set of features (EIS) according to the actual impact set (AIS). The EIS represents the affected feature computed by our technique while AIS is computed by manually tracing features to identify the affected features for each given change proposal. *Recall* measures the degree to which the EIS covers the AIS members. F-measure makes a trade-off between precision and recall, so that it gives a high value only in the case that both recall and precision values are high. Based on the definitions above, we can deduce that precision also quantifies elements of EIS that actually are not impacted (false-positive). Also, recall quantifies the features that are not identified but are impacted (false-negatives). Our proposed technique aims to achieve high precision, recall and F-measure. All measures have values within [0,1]. If the EIS has a high precision, this means that maintainers spend less time and effort to locate affected features. If the EIS has a high recall, this gives maintainers the confidence that all of affected features have been considered.

## 4.3 Effectiveness of Our CIA Technique

Table 4 summarizes the results obtained by our CIA technique. Columns describe respectively: change set of

<sup>1</sup><http://argouml.tigris.org/>

<sup>2</sup><http://www.ic.unicamp.br/~tizzei/mobilemedia/>

<sup>3</sup>[http://www.witi.cs.uni-magdeburg.de/iti\\_db/research/cide/](http://www.witi.cs.uni-magdeburg.de/iti_db/research/cide/)

**Table 4. Precision, Recall and F-measure of our CIA Technique.**

CSC	$ CSC $	$ EIS $	Precision	Recall	F-measure	CAM
<b>MobileMedia</b>						
CSC1	5	5	60%	75%	67%	100%
CSC2	5	6	83%	100%	90%	83%
CSC3	8	6	67%	100%	80%	100%
<b>ArgoUML-SPL</b>						
CSC1	9	5	80%	100%	88%	62%
CSC2	8	4	75%	100%	86%	50%
CSC3	18	5	80%	100%	88%	62%
<b>BerkeleyDB-SPL</b>						
CSC1	6	25	92%	100%	96%	92%
CSC2	5	25	100%	100%	100%	100%

classes (CSC), the size of CSC ( $|CSC|$ ), the size of estimated impact set of features ( $|EIS|$ ), *precision*, *recall*, *F-measure* and *CAM*. We randomly select three different CSCs for both *ArgoUML-SPL* and *MobileMedia*, and two CSCs for *BerkeleyDB-SPL*. Therefore, we have 8 CSCs to be analyzed. These selected CSCs are modified by considering different change types, including changes made to *class signature*, *class body*, *attributes*, *method signature* and *method body*.

Table 4, shows that precision values are fluctuated and take a value in the range between 60% and 100%. This fluctuation can be attributed to two reasons. Firstly, some changes made to CSC do not have any impact on feature implementations. These changes include deleting dead source code (e.g., conditional branch that logically will never be entered), adding output statements, etc. Secondly, the impact set of classes may contain some classes that, in fact, are not impacted. Such classes are called *false-positive classes*. For example, consider that C1 and C2 are two classes connected by a method invocation and C1 is proposed to be changed by adding an attribute. In this case, C2 is considered as affected class in spite of it is not be affected by this change. Such a case indicates features that are implemented by false-positive classes actually are not affected. Recall values shown in Table 4 are high where these values take a range between 75% and 100%, and in most cases they reach 100%. The reason that hinder our technique to achieve 100% for all CSCs is that we do not consider classes that are not neighbors of CSCs. These classes may contribute to implement feature(s). F-measure values confirm that our CIA technique gives high precision and recall, where these values are high taking a range between 67% and 100%.

CAM values in Table 4 shows the changeability of features of each case study against each CSC considered. We notice that all CSCs of *MobileMedia* affect more than half of its features. This is because *MobileMedia* is a small-scale system, and hence its source code classes are strongly coupled so that any change may impact many different features. For *ArgoUML-SPL*, all changes made to its features affect

almost half of its features. This is due to *ArgoUML-SPL*'s features being loosely coupled. They appear as isolated subsystems, for example, *cognitive* feature is implemented by 221 classes. For *BerkeleyDB-SPL*, changes made to its features affect most of its features. By investigating the impact set of classes and the concept lattice corresponding to this case study, we find that one of the changed classes (*EnvironmentImpl*) is located at the top of the lattice. This means that changes made to this class are propagated to the implementation of most features, which leads to a rise in the value of CAM. Based on CAM values shown in Table 4, we notice that these values quantify the changeability of the features against each change proposal. This allows SPL's manager to select the change strategy that results in the lowest possible CAM value.

## 5 Related Work

A large body of research is proposed for CIA. A comprehensive survey about CIA techniques can be found in [9]. The approach proposed by Revelle et al. [10] is the closest to ours. They proposed a feature coupling metric for supporting CIA at the feature level since features that are strongly coupled to a feature with modified implementation are the most likely to be affected. In their approach, features with a coupling value equal to or above a given threshold are considered as coupled features. However, coupled features under the specified threshold value may be affected by the change as shown in their experimental results. Hammad et al. [11] proposed an approach to determine which source code changes impact the system architecture. Diaz et al. [12] proposed an approach to perform CIA at the architectural level for changes induced at the requirement level. Chechik et al. [13] proposed a model-based approach for studying changes propagated between requirements and design models. Khan and Lock [14], utilized dependencies between requirement-level concerns and architectural components to studying the impact of requirement changes at the architecture level. All works mentioned above, except Revelle et al.'s work [10], do not support CIA at the feature level. They perform CIA at other different levels of abstraction.

## 6 Conclusion

In this paper, we have proposed a feature-level CIA approach to study the impact of changes made to source code of features obtained from product variants. This approach is useful for conducting change management from SPL manager's point of view. Our approach takes, as input, a change proposal at class level and computes a ranked list of potential affected features. Also, we propose two metrics to support this point of view. The proposed approach employed formal concept analysis, feature and structural couplings. Our experiments on three core assets of three case

studies of different domains and sizes proved the effectiveness of our approach in terms of the most used metrics on the subject (*precision, recall* and *F-measure*).

## References

- [1] P. C. Clements and L. M. Northrop, *Software product lines: practices and patterns*. Addison-Wesley, 2001.
- [2] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (foda) feasibility study," 1990.
- [3] D. Beuche, "Transforming legacy systems into software product lines." in *SPLC*. IEEE, 2011, p. 361.
- [4] J. Liu, D. Batory, and C. Lengauer, "Feature oriented refactoring of legacy applications," ser. ICSE '06. New York, NY, USA: ACM, 2006, pp. 112–121.
- [5] L. Bixin, S. Xiaobing, L. Hareton, and Z. Sai, "A survey of code-based change impact analysis techniques," in *software testing, verification and reliability*. Wiley Online Library, 2012.
- [6] B. Ganter and R. Wille, *Formal Concept Analysis: Mathematical Foundations*, 1st ed. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1997.
- [7] S. Apel and D. Beyer, "Feature cohesion in software product lines: an exploratory study," ser. ICSE '11. ACM, 2011, pp. 421–430.
- [8] H. Eyal-Salman, A.-D. Seriai, and C. Dony, "Feature-to-code traceability in a collection of software variants: Combining formal concept analysis and information retrieval," ser. IRI'13. California, USA: IEEE, 2013, pp. 209–216.
- [9] D. L. Andrea, F. Fausto, and O. Rocco, "Traceability management for impact analysis." in *ICSM*, 2008, pp. 21–30.
- [10] M. Revelle, M. Gethers, and D. Poshyvanyk, "Using structural and textual information to capture feature coupling in object-oriented software," *Empirical Softw. Engg.*, vol. 16, no. 6, pp. 773–811, 2011.
- [11] M. Hammad, M. L. Collard, and J. I. Maletic, "Automatically identifying changes that impact code-to-design traceability during evolution," *Software Quality Control*, vol. 19, no. 1, pp. 35–64, 2011.
- [12] J. Daz, J. Prez, J. Garbajosa, and A. L. Wolf, "Change impact analysis in product-line architectures." in *ECSA*, 2011, pp. 114–129.
- [13] M. Chechik, W. Lai, S. Nejati, J. Cabot, Z. Diskin, S. Easterbrook, M. Sabetzadeh, and R. Salay, "Relationship-based change propagation: A case study," ser. MISE '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 7–12.
- [14] S. S. Khan. Simon Lock, "Concern tracing and change impact analysis: An exploratory study," ser. EA '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 44–48.

# Are The Integrations Between Ontologies and Databases Really Opening the Closed World in Ubiquitous Computing ?

Vinicius Maran, José Palazzo M. de Oliveira

Institute of Informatics - Federal University of Rio Grande do Sul (UFRGS)  
Porto Alegre – RS – Brazil  
{vmaran, palazzo}@inf.ufrgs.br

Iara Augustin

Technology Center - Federal University of Santa Maria (UFSM)  
Santa Maria – RS – Brazil  
august@inf.ufsm.br

*Abstract: Currently, ontologies are widely used in ubiquitous systems for the representation of context and situation information. In addition, task ontologies are used to describe processes and daily situations that may occur in certain domains. In this way, it is necessary that there are strategies of integration between ontologies and databases, because ubiquitous systems generate large amounts of context data. In addition, it is necessary that exist ways to query and infer information on these persisted ontologies. Through the description of a motivating scenario, based on ClinicSpace ubiquitous architecture, and the study of requirements identified on recently proposed architectures, it was possible to list a set of important features to use persistence tools in ubiquitous systems. This paper presents an analysis of recently proposed solutions for the use of ontologies as data model for context representation and situation inference in ubiquitous systems. Through this analysis was possible to list a set of requirements that the actual state of the art does not attend, such as the possibility of distributed inference about information in ontologies.*

**Keywords — Ontologies; Ubicomp; Databases; Context-Awareness; Pervasive Computing.**

## I. INTRODUCTION

The idea proposed by Mark Weiser [31], known as Ubiquitous Computing, defines a world composed by intelligent environments where users and devices of all kinds are fully integrated in a way that provides the invisibility of computing. In these spaces, computing should be transparent to the user, ie, devices and systems should assist users in performing their daily tasks and it is desirable that users can not perceive the aid of computation involved in these processes. Thus, the main feature of ubiquitous systems is that they perform tasks centered on the end user and their daily activities according to the needs of these users and the context where they are inserted [14].

Context can be represented computationally in various ways. One of the most used ways is through ontologies, which allow the formal representation of a set of concepts and terms that represent domain knowledge [27]. Moreover, ontologies can be used to represent situations, and can be used for inference of new information based on the definitions of previously modeled contexts and situations. For the ubiquitous systems can complete the adaptation to the context as a result of situations that occur during their execution (context-awareness ad-

aptation) is necessary that exists strategies to interconnect context information with data persistence tools. Recent researches propose the definition of methodologies for this interconnection, but there is still no consensus on the use of a particular data model.

In this paper we present a motivation scenario based on ClinicSpace ubiquitous healthcare system [17][11], which defined a ubiquitous system for customization of tasks and clinical workflow inside hospitals. From this scenario and studies carried out on ubiquitous architectures from different domains we defined a set of features for the integration of context modeling, persistence, recovery and context inference tools. Furthermore, we found that many of these requirements are common to other ubiquitous architectures, regardless of the field in which they are used.

From the definition of these requirements, we conducted a review in relation to state-of-the-art works that propose the integration between ontologies and databases in order to verify whether these solutions meet the requirements for their use in ubiquitous systems, and if these solutions are really integrating the concept of Open World (OW – present in ontologies) with databases – which uses the concept of Closed World (CW).

OW defines that information or structures that are not modeled in an ontology exist in the world, but are not modeled. Thus, a query about a not represented knowledge in the ontology may not result in a false statement; it must return a response stating that the knowledge is unknown [21][16]. CW in turn defines that everything, which is not modeled in the conceptual model of the database, is false. Thus, if a query involves data that do not exist in the database, the response will be false, if a similar query was made in an ontology, the answer would be unknown [21][16]. Additionally, databases use UNA (Unique Name Assumption), which states that every individual has a single and unique name. In ontologies, individuals may have more than one name.

Next, Section 2 records the concepts used for modeling context and situation information in ubiquitous systems. Section 3 describes a motivation scenario based on ClinicSpace architecture. Section 4 describes the state of the art of integration between ontologies and databases. Section 5 illustrates a comparative study of methods of integration between ontologies and databases and presents a discussion of results, which

indicate a range of implementation difficulties and research possibilities in the area. Section 6 presents the conclusions of this work.

## II. CONTEXT AND SITUATION INFORMATION IN UBIQUITOUS SYSTEMS

Ubiquitous systems base their operation on information collected from the environment and on user interactions. The inputs data are collected by sensors are abstracted, forming context data. The term context can have several definitions, because it is a broad term, which encompasses several areas.

According to Dey & Abowd [10], context can be defined as "any information that might be used to characterize the situation of entities (person, place or object) that are considered relevant to the interaction between a user and an application including the user and application". Bazire & Brézillon [5] define context based on two main settings: (i) the context acts as a set of constraints that influence the behavior of a system, embedded in a given task, and (ii) definition of context depends on the area of knowledge to which it belongs. [10] and [5] are the most widespread and used context definitions in recent studies of context awareness.

Context data can be represented in various ways. Recent studies [26][23][6] made comparisons between the forms of representation of context and the requirements that they meet. These comparisons were based on six key factors, named in the comparative table as: distributed composition (*dc*), partial validation (*pv*), information quality (*qua*), incomplete and ambiguous (*inc*), level of formality (*for*) applicability in existing environments (*app*) and interoperability between systems without the necessity of adaptation (*int*) [26]. Table 1 shows the compiled result of these comparisons.

TABLE I. COMPARISON BETWEEN FORMS OF CONTEXT REPRESENTATION. ADAPTATION OF [26][23]

Ordered Approaches	<i>dc</i>	<i>pv</i>	<i>qua</i>	<i>inc</i>	<i>for</i>	<i>app</i>	<i>int</i>
Key-Value	-	-	-	-	-	✓	✓
Markup Scheme	✓	✓	✓	-	✓	✓	✓
Graphic	-	-	✓	-	✓	✓	✓
Logic Based	✓	-	-	-	✓	-	✓
Object Oriented	✓	✓	✓	✓	✓	✓	-
Ontology Based	✓	✓	✓	✓	✓	✓	✓

Thus, it was found in the comparisons that context modeling based on ontologies meets key requirements for the complete representation of contexts. Moreover, this form of representation allows interoperability between systems in different languages, a difficult task to be accomplished if we use other ways, for example, based on object oriented approach which depends of characteristics present in each programming language.

In addition to defining context, the definition and inference of situations in ubiquitous systems are necessary because they allow ubiquitous systems and services to determine what actions should be executed before a combination of situations and contexts [32]. Situation is an abstract concept that can be defined in various ways. Some authors [19][32] define it as an

external abstraction (from the point of view of applications) for a set of relevant contexts on environmental entities and their semantic relationships. There are also proposals to add time variant concept in situations [7][3]. These proposals include the concepts of situations that must have a certain time of existence, and the lifetime of the situation must be situated in global time. For example, *agitated\_patient* situation can happen during certain periods of time during a day in the field of clinicians care to elderly patients.

In literature, there are proposals to use methodologies for the representation and detection of situations. Ontologies can be used to specify situations. Moreover, logical rules can be created to perform the inference of situations and determine which operations must be performed based on the state of the environment [26]. From the modeling of context and situations based on ontologies, it is necessary to use language standards for queries and inferences in these definitions.

For queries and inferences to be made in information represented in OWL-DL ontologies, W3C [29] recommends the use of three languages: SWRL (Semantic Web Rule Language), used for the development of logical rules and consequently the inferences, SQWRL (Semantic Query-enhanced Web Rule Language), which adds functions to SWRL for queries in OWL-DL ontologies. SWRL is a language used to define rules in first order logic and is widely used in ubiquitous architectures. The code snippet shown below is an example of SWRL rule applied the definition of *Lying* action, "if an user has a tag, the tag is at 10 cm high from the floor, and it is staying for 60 seconds" [15].

```
owns(?user, ?tag) ^ hasHeight(?tag, ?height) ^
swrlb:lessThan(?height, 10) ^ stayFor(?tag, ?duration) ^
swrlb:greaterThan(?duration, 60) →
hasMovement(?user, Lying)
```

Fig. 1. SWRL definition of *Lying* situation [15]

## III. MOTIVATION SCENARIO

To identify important requirements for persistence and use of context information in ubiquitous systems, we describe a usage scenario of ClinicSpace architecture. This scenario was created based on the descriptions provided by other studies [17][18]. Besides defining the scenario and rank a set of important requirements for persistence and use of context information, we relate these requirements with other recent works of ubiquitous architectures, to determine whether these architectures meet these requirements.

ClinicSpace ubiquitous architecture was designed to be used in hospitals, allowing doctors and clinicians to model their everyday activities in computer interfaces. This modeling of activities should be aware of the context, ie, each one of the activities defined in the workflow of the user is performed according to the contexts of environments where the activity will be held. For this to occur, the ClinicSpace architecture uses the definition of contexts based on OWL-DL language, and stores these settings using the *SemantiCouch* architecture [18]. To illustrate some important requirements on the use of context, we describe a scenario where the ClinicSpace ubiquitous architecture is embedded.

*"A neurologist wants to consult the history of his patient exams on his mobile device. ClinicSpace obtains the current context, identification of the patient, location of the physician, and patient, devices and sensors. All information is modeled in OWL-DL [35]. The physician receives the results of tests made recently by the patient who is being treated, concerning their clinical specialty. Moreover, the tests are presented to the physician in a manner adapted to his device - for the viewing capability of the display, the available bandwidth of the network, location where the doctor is and the people around him (if this information requires privacy or not)".*

From the definition of this usage scenario, we identify a set of requirements for persistence and retrieval of information used by the architecture. Moreover, some of these requirements have been also identified in other recent studies of ubiquitous architectures [28][13][12]. The following are the main requirements for the persistence of context and situation information in ubiquitous architectures:

- **Vertical and Horizontal Scalability.** The persistence tools that support semantic data should provide (i) vertical scalability - possibility of increasing the capacity of the database when it is used locally; and (ii) horizontal scalability – data replication in databases on multiple nodes of a network, allowing these nodes to access data in the same way as a local database. For that it can use a distributed infrastructure to store and query data, allowing the data persisted in this architecture to be found in several places by devices of all kinds. This feature is required by the use of large amounts of information sensed from the environment in ClinicSpace [18] (generated from sensors and inferences about the environment), also, the amount of data is a problem in other ubiquitous architectures [28][13][12], and the use of heterogeneous devices is required - two key characteristics of ubiquitous systems [33][18];

- **Standardized query and inference languages for OWL.** Preferentially, the methods of integration and persistence of ontologies tools should use query and inference standardized languages for OWL. Among these languages include the SWRL and SQWRL languages, since they are subject to standards W3C [29] and permit the inference and query data in the OWL language [32][28]. Inferences and queries can also be made using SPARQL [29], but this language was defined for use in conjunction with the RDF standard. All recent architectures that uses OWL-DL as context representation (including ClinicSpace) uses SWRL to infer about context [28][13][12];

- **Persistence only of the relevant data.** The constant conversion of formats and query languages makes that occurs a steady expansion (overhead) on the overall size of the data, resulting in considerable loss of performance depending on the size of the database and the number of queries. Furthermore, the space occupied by these data greatly increases with the use of these databases [4]. [18] and [4] presents recent studies of solutions to minimize the overhead of data, but these solutions do not implement solutions for reasoning over OWL-DL;

- **Using part of the ontology in memory.** The processing of inferences and queries should be done only with relevant data, mainly because the performance and memory consumption increases considerably with the increase of individuals in

the ontology. Therefore, it is important that changes be processed on a limited data part - through a filter to use only part of the data in memory [4]. Also, the actual widely used inference engines over OWL-DL have high computational complexity, through the use of Tableau or Hypertableau algorithms. This level of complexity decreases the performance of data management, as shown in [35];

- **Support to queries, inference and modeling of temporal situations.** Situations should be modeled and should be stored according to their occurrence. The storage of situations allows ubiquitous architectures to perform processes proactively. For this to happen, it is necessary that the integration methodologies enable the modeling, query and inference of situations based on the time variation;

- **Integration with mobile platforms.** Ubiquitous systems must support heterogeneous devices in a distributed manner; this implies the use of mobile platforms in the integration of the system with the environment. In our vision, it is necessary that the strategies of integration between ontologies and databases allow architectures to use the partial replication of information and reasoning over data on mobile devices;

- **Support for OWL-DL standard.** The OWL-DL standard is commonly used to represent context information in ubiquitous systems. According to W3C, OWL-DL offers maximum expressiveness without losing computational completeness and decidability of inference engines. OWL 2 [29] offers a new set of profiles and tools, but OWL-DL still widely used to represent contexts and situations [34][22].

#### IV. INTEGRATION TOOLS BETWEEN ONTOLOGIES AND DATABASES

Strategies for integrating ontologies and models of databases have been proposed in two distinct lines: (i) integration of ontologies in relational databases [33], and (ii) integration between ontologies and NoSQL databases or distributed file systems. Semantic data are used in many areas of computing. Therefore, some solutions are defined for specific purposes, with representation languages and specific queries to the field where they will be applied. The information of techniques surveyed in this study are presented in Table 2.

##### A. Relational Databases

Many solutions related to the integration between databases and ontologies use relational model as basis for integration.

SciSPARQL [2] is an architecture that expands SPARQL language to address relevant issues to the scientific field - specifically in research groups that use information modeled in RDF format and have large amounts of generated information. This architecture allows scientific data to be stored in RDF triple format using the relational data model. To carry out the queries, the architecture allows the use of SciSPARQL, a query language that extends the functions of SPARQL query language [14] with external functions defined in Python language. The SciSPARQL does not support inferences in persisted ontologies. This feature is referred to as a topic of future work by the authors.

TABLE II. INFORMATION ABOUT INTEGRATION TOOLS BETWEEN ONTOLOGIES AND DATABASES USED IN THE ANALYSIS

Work	[2]	[24]	[8]	[25]	[20]	[4]	[18]	[30]	[9]	[1]
Data Model	RDBMS	RDBMS	RDBMS	RDBMS	XML	Object Oriented	JSON docs	Hadoop files	Files	Files
Query Language	Sci SPARQL	Sparql	Based on Datalog	ROQL	nRQL	RuleML	QBE	Not mentioned	Alc	Not mentioned
Ontology Format	RDF	RDF	RDF / OWL-DL	RDF / OWL-DL	RDF / OWL-DL	RDF / OWL-DL	RDF / OWL-DL	RDF	RDF	RDF / OWL
Inference	-	-	✓	-	✓ RacerPro	✓	-	-	✓	-
System	Not mentioned	Postgre / MySQL	Not mentioned	MySQL	eXist XML Database	Db4Owl	Couch DB	Hadoop	Not mentioned	GFS

Shan [24] presents a methodology for integrating ontologies defined in RDF format and relational database MySQL or PostgreSQL. Through a mapping table created specifically for this architecture, the algorithm C-Store [24] defines the database based on the inserted ontology model. SPARQL language is used to query information persisted in the database.

Nyaya [8] is a persistence architecture of semantic data that allows large amounts of information to be stored. To do this, each imported RDF in architecture is converted into a structure called Semantic Data Kiosks with structures and modeled data according to the imported RDF. These structures also generate some extra data that allow queries to be made using a Datalog based language. When a query is performed in the language, the architecture converts the query into subqueries in SQL, and then accesses the data persisted in relational database.

ERMOS [25] is a project that implements the integration of OWL-DL and MySQL database. It implements the Rule Based Ontology Query Language (ROQL) language. Through the use of this language, developers can query (but can not do inferences), which are converted into SQL statements.

#### B. NoSQL Databases and Distributed Filesystems

Based on the possibility of insertion of XML files without the necessity of conversion to other data model, OXDBS [20] proposed the extension of the functionalities of eXist database to the possibility of syntax checking of RDF and OWL-DL files, and allows the use of RacerPro inference engine to query and inference ontologies.

Db4OWL architecture [4] proposes the integration of OWL-DL ontologies and DB4O database, which is object-oriented. To accomplish this integration, the authors created a model in Java language that represents the structure found in OWL. From this, the structure and the information contained in it were stored in the database. Queries and inferences are written in RuleML format [4], and subsequently converted to QBE (Query by Example) format, used by DB4O database.

SemantiCouch [18] proposes an architecture of integration between OWL-DL ontologies and CouchDB document oriented-database. Thus, the information could be replicated easily if compared to other approaches. To perform the queries, the architecture provides an API based on QBE.

Other approaches propose the integration of ontologies and distributed file systems, allowing the same ontological definition to be replicated in many places. JenaPro [30] presents an architecture of integration between Jena framework - for the use of ontologies in the Java language, and Hadoop architecture - used for the distribution of information.

Swarms [9] presents an architecture for the persistence of semantic data represented in RDF format. The architecture was modeled and implemented in a way that allows the distribution and replication of information across nodes in a network. This information distribution is done intelligently, through the analysis of paths between nodes, to choose the best route information distribution. To carry out the queries, the authors use ALC language [9].

OCSS (Ontology Cloud Storage System) [1] presents an architecture and a set of algorithms of reading and writing ontologies modeled in RDF and OWL languages persisted in GFS (Google File System). This architecture allows a Cloud Computing infrastructure to be created for the storage and querying of ontologies.

## V. ANALYSIS AND DISCUSSION

Ontologies have been used as a form of contexts and situations representation in recent ubiquitous architectures, regardless of the areas where these architectures are inserted. Thus, is necessary that exist forms to integrate ontologies and databases, to allow control of the information captured by the environment and allow this information to be consulted and new information be inferred. Methodologies for integrating ontologies and databases are being proposed, but there is no consensus among studies regarding in which model database is best to perform this integration.

The more used model is the relational, mainly by extensive use by its formal conceptual model and the number of tools associated with this model. However, more recent approaches have been proposed with the use of other models in order to overcome some shortcomings of this model.

From the analysis of the characteristics of the recently proposed methods, we performed a second analysis. This analysis aims to determine whether the methods proposed thus far meet the requirements listed in Section 3, based in the scenario of ClinicSpace use, for the use of integrated databases in ubiquitous systems ontologies.

The low overhead requirement in data conversion is contemplated in more recent studies. In the works [30][9][1] there is no file conversion, because the works proposed persistence architectures using distributed file systems. The work [20] also shows no file conversion, it uses XML as a data storage model. Other recent works [4][18] proposed the use of different storage strategies to decrease the creation of new data on the persistence of ontologies process. Specifically on these two works, object-oriented and JSON language were used. As we can see, the data overhead is well worked by recent researches.

About horizontal scalability, only 4 studies in the comparative offered any solution. The works [30][9][1] provide horizontal scalability through the use of distributed file systems. However, these do not use a database management system; thus, do not offer query languages or inference to ontologies.

Regarding query languages, most jobs do not allow the systems to query and inferences using languages based on Datalog. Defined by the W3C standard for querying RDF document in databases - in most of these works, the SPARQL language is utilized. But the SPARQL language does not provide the means to achieve inferences, supported by languages SWRL and SQWRL, so little support builders and important definitions found in standard OWL-DL.

Still regarding query languages, without the possibility of inferences on the data stored in databases, some jobs do not offer the ability to query and inference of situations based on timing issues. Most recent studies support the persistence of files in OWL-DL standard. This enables contexts to be represented and stored by ubiquitous architectures. However, none of the studies reviewed presented in the form of the integration testing with mobile platforms.

This requirement is important in ubiquitous architectures, since it reduces the necessity of existence of a central server with all context representations. Instead, mobile devices can contain and manipulate context definition according to the device, subsequently; the captured context can be replicated on these devices. Table 3 shows the result of the relationship between the requirements presented in Section 3 and state of the art methods of integration between ontologies and databases.

TABLE III. RELATIONSHIP BETWEEN REQUIREMENTS OF CLINICSPACE AND OTHER UBIQUITOUS ARCHITECTURES AND RECENT UBIQUITOUS SYSTEMS INTEGRATION METHODOLOGIES

Requirements / Work	[2]	[24]	[8]	[25]	[20]	[4]	[18]	[30]	[9]	[1]
Low data overhead	-	-	-	-	✓	✓	✓	✓	✓	✓
Easily Horizontable Scalable	-	-	-	-	-	-	✓	✓	✓	✓
Support of query and inference languages to OWL (SWRL, SQWRL or based on Datalog)	-	-	✓	-	-	✓	-	-	✓	-
Support inferences and temporal modeling of situations	-	-	✓	-	-	✓	-	-	-	-
Integration with Mobile Platforms	-	-	-	-	-	-	-	-	-	-
OWL-DL support	-	-	✓	✓	✓	✓	✓	✓	-	✓

As we can see, there is still a gap between the horizontal scalability of architectures of ontologies persistence and the support of inference and query in standard languages OWL-DL. This is primarily because the majority of inference and supporting the SWRL and SQWRL engines work patterns to form separate DBMSs that store ontologies.

Architectures for context management are directly related to bond with sensors and mobile devices. The processing and storage capacity of mobile devices may limit the action of architecture. Also, perform all operations on the context in one place (client-server) is not recommended because the environment is heterogeneous and distributed [36][37]. Historic analyses are directly related to the ability of ubiquitous systems persistence and recovery of context. If modeling is done using

ontologies are generally used the SWRL language (for inferences) and SPARQL (for queries in RDF). However, these languages do not offer mechanisms to monitor the flow of information in real time, and have few resources compared to similar languages based on SQL for queries in relation to time/space [6][26]. Efficient inference and query that integrate ontologies are still a problem for ubiquitous systems [6][26][38]. Furthermore, these approaches do not meet the mobility requirements needed for ubiquitous systems.

Distributed persistence only offers the possibility to distribute and query the information, but the possibility to infer about ontologies in a distributed manner is a high complexity problem. In ubiquitous systems, the realization of these inferences should be made with portions of ontological representations. Figure 2 shows an adaptation of the scheme presented in [6].

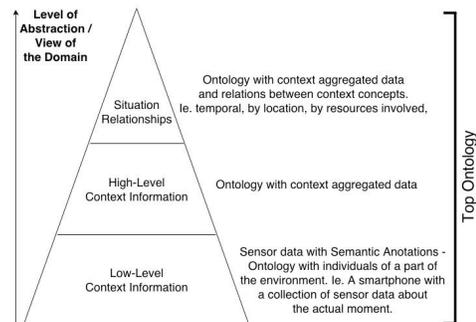


Fig. 2. Pyramid of context representation according to level of abstraction and view of the world. Adaptation of [6]

This schema of context management differs from other proposals, because it proposes the use of ontology as the top support to the whole process of representation (from the data representation to the representation of situations). Thus, architecture for context management could manage contexts and situations without constant conversions of formats, and the ability to make inferences at different levels (for there is the use of ontologies with different levels of abstraction and worldview).

For this to occur, it is necessary that exists: **(a)** A lightweight form of serialization of ontologies based on a W3C standardized language, such as JSON; **(b)** An inference engine that performs the inference of ontologies in levels, according to the worldview represented in the ontology; and **(c)** An architecture management framework that provides those allied to a persistence model that prioritizes efficiency over the generation of extra data features.

## VI. CONCLUSIONS

Ontologies have been used as a way of representation of contexts and situations in recent ubiquitous architectures, regardless of the areas where these architectures are inserted. Queries and inferences efficiently has still demonstrated a major challenge in ubiquitous systems, this is primarily because the current methodologies for integrating ontologies and databases do not include all the features of the inference engines.

The performed study shows that there are still great opportunities for research in the area, especially in relation to ubiquitous systems. In this field there are still major challenges relat-

ed to integration of inference and managers of databases on heterogeneous devices (fixed and mobile) engines, and in relation to efficiency analysis of information, due to the large amount of generation of information occurs in ubiquitous systems.

#### REFERENCES

- [1] Al Feel, H. T., & Khafagy, M. H. OCSS: Ontology Cloud Storage System. *2011 First International Symposium on Network Cloud Computing and Applications*, 9–13. doi:10.1109/NCCA.2011.9
- [2] Andrejev, A., & Risch, T. Scientific SPARQL: Semantic Web Queries over Scientific Data. *2012 IEEE 28th International Conference on Data Engineering Workshops*, 5–10. doi:10.1109/ICDEW.2012.67
- [3] Augusto, J.C., Liu, J., McCullagh, P.J., Wang, H and Jian-Bo, Yang (2008) Management of uncertainty and spatio-temporal aspects for monitoring and diagnosis in a Smart Home. *International Journal of Computational Intelligence Systems*, 1 (4). pp. 361-378.
- [4] Batzios, A., Mitkas, P.A., "db4OWL: An Alternative Approach to Organizing and Storing Semantic Data," *IEEE Internet Computing*, vol. 13, no. 6, pp. 48-55, Nov./Dec. 2009.
- [5] Bazire, M., Brézillon, P. (2005) "Understanding context before to use it". In 5th International and Interdisciplinary Conference on Modeling and Using Context, *Lectures Notes in Artificial Intelligence*, Vol 3554, pp. 29--40, Springer-Verlag.
- [6] Bettini, C., Brdiczka, O., Henricksen, K., Indulska, J., Nicklase, D., Ranaganatha, Riboni, D. (2010) *A survey of Context Modelling and Reasoning Techniques*. Per-vasive and Mobile Computing, 161-180.
- [7] Cook, D. J., Augusto, J. C., Jakkula, V. Review: Ambient intelligence: Technologies, applications, and opportunities. *Pervasive Mob. Comput.* 5, 4 (2009), 277-298. <http://dx.doi.org/10.1016/j.pmcj.2009.04.001>.
- [8] De Virgilio, R., Orsi, G., Tanca, L., & Torlone, R. NYAYA: A System Supporting the Uniform Management of Large Sets of Semantic Data. *2012 IEEE 28th International Conference on Data Engineering*, 1309–1312. doi:10.1109/ICDE.2012.133
- [9] Dentler, K., Muhleisen, H.; Dentler, K., "Large-Scale Storage and Reasoning for Semantic Data Using Swarms," *Computational Intelligence Magazine, IEEE*, vol.7, no.2, pp.32,44, May 2012.
- [10] Dey, A.; Abowd, G. The Context Toolkit: Aiding the Development of Context-Aware Applications, In: *Proceedings of Human Factors in Computing Systems*, Pittsburgh, PA: ACM Press, pp.434-441. 2006
- [11] Ferreira, G.; Librelotto, G.; Silva, L.; Yamin, A. Middleware for management of end-user programming of clinical activities in a pervasive environment In: *Workshop on Middleware for Ubiquitous and Pervasive Systems*, Vol. 389, pp.07–12. 2009
- [12] Jun Li; Shvartzshnaider, Y.; Francisco, J.; Martin, R.P.; Nagaraja, K.; Raychaudhuri, D., "Delivering Internet-of-Things services in MobilityFirst Future Internet Architecture," *Internet of Things (IOT), 3rd International Conference on the*, 2012
- [13] Kovatsch, M.; Lanter, M.; Duquenooy, S., "Actinium: A RESTful runtime container for scriptable Internet of Things applications," *Internet of Things (IOT), 2012 3rd International Conference on the*, vol., no., pp.135,142, 24-26 Oct. 2012 doi: 10.1109/IOT.2012.6402315
- [14] Kukhun, D.A.A., Sedes, F. "Step Towards Pervasive Software: Does Software Engineering Need Reengineering?" In the book: *Complex Systems Concurrent Engineering*, Ed. Springer. ISBN 978-1-84628-975-0, 2007, pp. 143-150.
- [15] Gui, Ning, et al. "A Service-oriented Infrastructure for Mutual Assistance Community." *World of Wireless, Mobile and Multimedia Networks, 2007. WoWMoM 2007. IEEE International Symposium on*.
- [16] Horrocks, I. *Ontology –v- Databases*. Available at: <http://www.cs.ox.ac.uk/ian.horrocks/Seminars/download/onto-db.ppt>.
- [17] Maran, V.; Augustin, I.; Saccol, D. B. A Service for ClinicSpace Architecture to Provide Context Data Persistence and Context-Based Selection of Documents. In: *39th Seminar on Hardware and Software*, 2012, Curitiba
- [18] Maran, V.; Machado, A. ; Saccol, D. B. ; Augustin, I. . A Software Architecture to Provide Persistence and Retrieve of Context Data Based on Ontological Models. In: *IADIS International Conference on WWW/Internet*, 2011, Rio de Janeiro, Brasil.
- [19] Najar, S., Saidani, O., Kirsch-Pinheiro, M., Souveyet, C., Nurcan, S., Semantic representation of context models: a framework for analyzing and understanding. In *Proceedings of the 1st Workshop on Context, Information and Ontologies (CIAO '09)*. ACM, New York, NY, USA.
- [20] Neumann, C. P., Fischer, T., & Lenz, R. (2010). OXDBS – Extension of a native XML Database System with Validation by Consistency Checking of OWL-DL Ontologies, 143–148.
- [21] Drummond, N., Shearer, R. The open world assumption. Or sometimes its nice to know what we don't know. The University of Manchester. Available at: <http://www.cs.man.ac.uk/~drummond/presentations/OWA.pdf>. 2006.
- [22] Martins, H. and N. Silva, "Characterization, Comparison and Systematization of Context Ontologies," *2012 Sixth Int. Conf. Complex, Intelligent, Softw. Intensive Syst.*, pp. 983–988, Jul. 2012.
- [23] Perttunen, M., Riekkki, J., Lassila, O., Context Representation and Reasoning in Pervasive Computing: a Review. In: *International Journal of Multimedia and Ubiquitous Engineering*. 2009 Science & Engineering Research Support soCieTy ISSN: 1975-0080
- [24] Shan, Y., Yu, J., & Chen, D. Implementation Method of the Semantic Data to be Stored into the Relational Database. *2012 International Conference on Computer Science and Service System*, 2075–2078.
- [25] Shoaib, M., & Basharat, A. (2010). ERMOS: An Efficient Relational Mapping for Ontology Storage. *2010 IEEE International Conference on Advanced Management Science (ICAMS 2010)*, 399–403.
- [26] Strang, T.; Popien, C. A Context modelling survey. Em: *Proceedings of the Workshop on Advanced Context Modelling, Reasoning and Management as Part of UbiComp*, pp.33-40, 2005
- [27] Swartout, W. and Tate, A. (1999). *Ontologies*. In *IEEE Intelligent Systems and their applications*, volume vl 14, n 1. IEEE.
- [28] Triantafyllidis, A. K.; Koutkias, V. G.; Chouvarda, I.; Maglaveras, N., "A Pervasive Health System Integrating Patient Monitoring, Status Logging, and Social Sharing," *Biomedical and Health Informatics, IEEE Journal of*, vol.17, no.1, pp.30,37, Jan. 2013
- [29] W3C Website. Available at: <http://www.w3c.org>. 2014.
- [30] Wang, H., Zhang, R., & Wang, Z. (2012). JenaPro: A Distributed File Storage Engine for Jena. *2012 Fifth International Joint Conference on Computational Sciences and Optimization*, 610–613.
- [31] Weiser, M. *The Computer of 21st Century*, Em: Scientific American, 1991, p 3-11.
- [32] Ye, J.; Dobson, S.; McKeever, S. Situation identification techniques in pervasive computing: A review. *Pervasive and Mobile Computing*, 8(1), 36–66. doi:10.1016/j.pmcj.2011.01.004. 2011.
- [33] Zhang, H., Wang, Z., Gao, Z., & Li, W. (2009). Design and Implementation of Mapping Rules from OWL to Relational Database. *2009 WRI World Congress on Computer Science and Information Engineering*, 71–75. doi:10.1109/CSIE.2009.290
- [34] Makris, P., Member, S., and Skoutas, D. N., "A Survey on Context-Aware Mobile and Wireless Networking: On Networking and Computing Environments ' Integration," vol. 15, no. 1, pp. 362–386, 2013.
- [35] Maran, V. ; Saccol, D. B. ; Librelotto, G. R. ; Augustin, I. Uma definição ontológica de elementos de contexto relevantes na adaptação de documentos em ambientes hospitalares pervasivos. *Ontological Definition of Relevant Context Data for Pervasive Hospitals*. Revista Brasileira de Computação Aplicada (Portuguese), v. 5, p. 26-41, 2013. <http://dx.doi.org/10.5335/rbca.2013.2586>.
- [36] Simoens, P., De Turck, F., Dhoedt, B., Demeester, P., "Remote Display Solutions for Mobile Cloud Computing", *IEEE Computer*, vol. 44, no. 8, pp. 46-53, 2011.
- [37] Makris, P., Member, S., and Skoutas, D. N., "A Survey on Context-Aware Mobile and Wireless Networking: On Networking and Computing Environments ' Integration," vol. 15, no. 1, pp. 362–386, 2013.
- [38] Krötzsch, Markus, Anees Mehdi, and Sebastian Rudolph. "Orel: Database-driven reasoning for OWL 2 profiles." *23rd International Workshop on Description Logics DL2010*. 2010.

# ONTO-ResAsset Development: An Ontology for Reusable Assets Specification and Management

Luciano Édipo Pereira da Silva<sup>1</sup> \*    Débora Maria Barroso Paiva<sup>1</sup>    Ellen Francine Barbosa<sup>2</sup>

Rosana Teresinha Vaccare Braga<sup>2</sup>    Maria Istela Cagnin<sup>1</sup>

<sup>1</sup>Facom – Federal University of Mato Grosso do Sul – Campo Grande (MS) - Brazil

<sup>2</sup>ICMC – University of São Paulo – São Carlos (SP) - Brazil

E-mail: <sup>1</sup>{lucianoedipo, dmbpaiva, istela}@gmail.com    <sup>2</sup>{francine, rtvb}@icmc.usp.br

## Abstract

*Reuse is an important mechanism to increase productivity and to reduce time and costs during software development. Although source code is the most commonly reusable asset, other types of assets can also be reused, such as requirements, business processes, analysis and design models, etc. In this context, it is important that the knowledge about reusable assets and its management are available to potential stakeholders. This work presents the development of an ontology of reusable assets specification and management, named ONTO-ResAsset. This ontology is evaluated under two points-of-view: domain experts and non-experts.*  
**Keywords**– Ontology, Software Assets, Software Asset Management, Software Reuse

## 1. Introduction

Software reuse, when conducted in a planned and systematic way, leads to better product quality, since the reusable assets have been previously used and tested. Additionally, it promotes agility to software projects execution and, consequently, cost reduction [15]. The reuse of software assets is not limited only to the implementation phase. Rather, it can also be applied in previous phases related to requirements elicitation, analysis and design [11], as well as maintenance and supporting activities (e.g. project management documentation, verification, validation, and testing). Thus, it is important to do the proper management of assets reuse [8], which is generally aided by reuse repositories [1].

The knowledge about assets and their management is scattered and spread throughout several types of documents (e.g. papers, books, standards, patterns, guides, etc.), under

different formats, and in several abstraction levels, besides being available in different ways. For example, the knowledge about reusable assets specification can be obtained specifically in books or technical reports [12, 4]; the knowledge about reusable assets managements and reuse repositories is spread in papers, books, and standards [3, 8, 1].

Under this perspective, it is relevant to notice that the knowledge about assets specification and assets reuse management should be unified, organized, and shared among those interested in it. This can be achieved through ontologies [7], which consist in an adequate technique to conceptualize knowledge.

The main goal of this paper is to present the development of a semi-formal ontology for the specification and management of reusable assets, named ONTO-ResAsset.

## 2. Related work

Works found on the literature about ontologies for the domain of interest of this paper define specific assets that can be reused during some activities of software development cycle, such as the SQA (Software Quality Assurance) ontology [2], the ontology of reusable test cases [10] and the UI<sup>2</sup>Ont ontology belongs to the user interfaces domain [13].

However, we did not find works that define ontologies to represent how a reusable asset can be specified, considering those that can be reused during the fundamental activities of software construction, and how a reusable asset can be managed. This knowledge is very important to all parts interested in software reuse and has motivated this work.

## 3. ONTO-ResAsset Development

The ONTO-ResAsset [14] was built based on the *Methontology* methodology [6], which defines a sequence

\*Financial support by Capes (Brazilian Coordination for the Improvement of Higher Education Personnel)

of steps that allow the gradual evolution of ontologies and the incorporation, at any moment, of new concepts as these steps are performed. To document the terms of the ONTO-ResAsset domain, it was used the glossary of terms and textual description in natural language. For the representation of the proposed ontology, a conceptual model was used, using UML (*Unified Modeling Language*) class diagrams.

Several information sources were used to identify the concepts of the ONTO-ResAsset [14, 3, 8, 12, 9]. The execution of the *Methontology* steps, used for the definition of the ONTO-ResAsset, is described below.

### 3.1. Step: Planning

The purpose of the ONTO-ResAsset ontology is to represent the domain of reusable assets in the context of software development, from the perspective of fundamental activities of the software construction (i.e, Requirements, Analysis, Design, Implementation, and Test), and considering the specification and the management of the reusable assets.

Regarding the level of formality, the ONTO-ResAsset is classified as a semi-formal ontology, according to Uschold [17], as it is expressed in a formally defined language - UML class diagram. Additionally, structured natural language is used in its representation.

To support the definition of the ONTO-ResAsset, the identification of domain terms was performed through four iterations of two steps of the *Methontology* (specification and knowledge capture), described in the next section. In the first iteration, the fundamental activities that are part of software development are identified and defined. In the second iteration the artifacts that can be reused during these activities were identified. In the third iteration the representation of artifacts was mapped as reusable assets. Finally, in the fourth iteration the required processes for managing reusable assets were identified.

### 3.2. Steps: Specification and Knowledge Capture

The **knowledge capture** step was performed transversely to the other steps of *Methontology*. Since the goal of this methodology is the creation of ontologies in an evolutionary manner, the knowledge capture step is carried out all the time.

In this work, the activities commonly performed for building the software were considered as fundamental activities of the software development cycle. These activities were identified from the analysis of construction activities of the software processes ontology [5] and of the development process of ISO/IEC 12207 [8]. The result of this step is a group of terms (G1) that describe fundamental activities, as for example Requirements, Analysis, Design, Implementation, and Test.

After defining the fundamental activities, we sought information about the artifacts commonly found in the literature and reused in at least one of these activities. The information sources to identify the concepts for that group (G2) were: the systematic review of Konda and Mandava [9], the domain analysis performed by Silva [14] from six reuse repositories, as well as the artifacts used during the software development according to Sommerville [16].

After the identification and selection of artifacts used by fundamental activities of software construction, it was noted in the literature the concepts that are related to reusable assets specification. Therefore, it was used as information source for that group (G3) the reusable assets specification of OMG [12]. In this paper, we considered the first two hierarchy levels of this specification for mapping the terms of the reusable assets specification, because they provide the necessary and appropriate knowledge about reusable assets within the ONTO-ResAsset scope, defined in the planning step.

The last group of terms (G4) of the ONTO-ResAsset domain refers to the management of reusable assets. For this, the following references have been used as information sources: Reuse Asset Management Process of ISO/IEC 12207 [8], and the set of functionalities of reuse repositories defined by Burégio [3], which contain functionalities commonly found in reuse repositories.

In the **specification** step, we defined 28 competence questions of the ONTO-ResAsset (for example: What activities are part of software construction? How is it possible to identify the problem that an asset can solve? What are the mechanisms that can be used to support the reuse asset management?), whose answers should be provided by the ontology. Moreover, in the **specification** step, a glossary of terms was created for each group of terms aforementioned, that due to space constraints are not presented here. Each glossary presents the name of each term, the information sources used and a textual description of each term. In the glossary of terms of group G1 we defined 6 terms (RequirementActivity, AnalysisActivity, DesignActivity, ImplementationActivity, etc). In the glossary of terms of group G2 we identified 11 terms (RequirementDocument, Model, SourceCode, Component, etc). In this group, we also identified the artifacts that are created or used in each activity of the previous group. In the glossary of terms of group G3 we defined 19 terms (Artifact, Asset, Classification, Context, etc) and in the glossary of terms of group G4 we found 18 terms (ManagementPlan, AcceptanceCriterion, InsertMechanism, Repository, etc).

### 3.3. Step: Conceptualization and Implementation

In the **conceptualization** step, the concepts corresponding to the terms obtained in the previous section and the re-

relationships and rules of association existing between them were identified.

It is worth noting that 24 new concepts were abstracted in this step. They resulted from the detailing of some relationships, but were not present in any information source used. In group G1 just a new term was defined (Person, which may take one or more roles during the fundamental activities of software construction). In group G3 we defined 16 new terms (AssetContext, SolutionArtifact, Problem, RequirementSolution, AnalysisSolution, etc). Finally, in group G4 we identified 7 new terms (Version, Feedback, Mechanism, NotificationMechanism, etc).

The concepts abstracted were needed to guide the conceptualization, aiming to answer the competence questions of the ONTO-ResAsset established in the specification step.

The relationships between the concepts were extracted from textual descriptions of domain terms, documented in the glossaries of terms (defined in the specification and knowledge capture steps). For each relationship, it has been defined a name, related to an action produced by the relation, for example, *sendsWarningAbout* (<NotificationMechanism> *sendsWarningAbout* <Asset>), and the multiplicity of the relationship between the concepts (for example, a <NotificationMechanism> *sendsWarningAbout* one or more <Asset>).

In the Conceptualization step, the meaning of each relationship was defined in a textual format. Besides, in this step a table of relationships was also created, and it presents all concepts involved in each relationship, the multiplicity and the relationship name. In some cases, rules for the existence of certain relationships were also defined and presented. Furthermore, each element of the conceptual model was described in natural language, helping to get a semi-formal definition of the domain addressed by the ontology.

In the **implementation** step, all the concepts, relationships and multiplicities were represented in a conceptual model, using the UML class diagram, where each class is a domain concept and the associations (including aggregation and composition) and inheritance relationships are the relationships established between the concepts.

In Figure 1 the high-level conceptual model of the ONTO-ResAsset is shown. Each group of concepts is represented by a package. Due to space restrictions, the conceptual model of each group could not be shown. The complete conceptual model of the ONTO-ResAsset can be obtained elsewhere [14].

#### 4. ONTO-ResAsset Evaluation

ONTO-ResAsset was evaluated by experts and non-experts in software reuse. The evaluation aimed at qualitatively analysing the ability of ONTO-ResAsset to provide

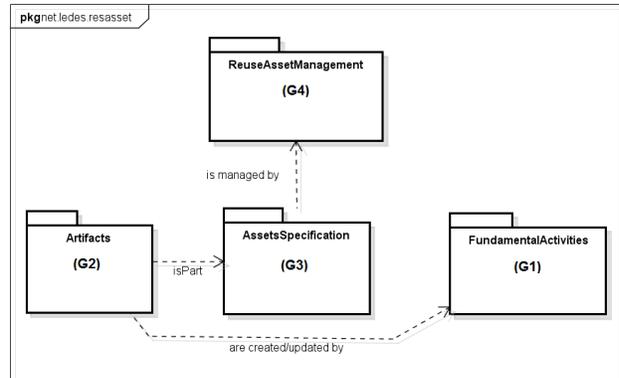


Figure 1: High-level conceptual model of ONTO-ResAsset

knowledge about the domain of reusable assets specification and management, in an attempt to support the dissemination of this domain.

The evaluation of ONTO-ResAsset was performed in two steps, considering different perspectives. In Step 1, the ontology was evaluated by experts in the field of software reuse. In Step 2, the evaluation was conducted by two groups of evaluators, representing non-experts in the field, particularly interested in the reuse of software assets.

In Step 1 the ONTO-ResAsset was sent to a group of Software Engineering researchers with interest in software reuse. The goal was to get an impression from experts regarding: (i) the consistency and the proper coverage of the ontology according to its scope; and (ii) the possible errors/inconsistencies with respect to the definition and representation of the concepts and relationships among them.

As a result of Step 1, ten suggestions for improvements in ONTO-ResAsset were identified. Of these, three refer to concepts (improvements in the textual description, specialization of concepts, removal of a concept, and so on) and seven are related to relationships between concepts (removal of redundant relationships, changing from aggregation and composition relationships to association relationships, and so on). All suggestions for improvements were addressed and implemented in ONTO-ResAsset.

Next, in Step 2, the ONTO-ResAsset was sent to a group of domain non-experts. The goal was to evaluate if the ontology was able to transfer and disseminate knowledge regarding the specification and management of reusable assets. The evaluation questionnaire of Step 2 was completely based on the competence questions established during the specification of the ONTO-ResAsset

Step 2 was performed by two groups of evaluators. Group 1 was composed of eleven non-experts, who are part of the development team for a software company. Group 2 was composed of nine students of a software development course at Facom (Graduate Program), without any expertise

in software reuse.

Each questionnaire answered by the evaluators in Groups 1 and 2 was analysed in order to verify the correctness of the answers and therefore assess the understanding of the domain expressed in the ontology by the evaluators. Each answer was scored as correct, partially correct, incorrect or unanswered. The answer was classified as partially correct when it had only some parts of the expected concepts or when it did not have all the correct items as expected.

The overall percentage of correct answers of the questions answered by Groups 1 and 2 was 68% and 73%, respectively; the percentage of partial success was 16% and 10%, respectively. Also, we noticed that 14% of the answers of Group 1 and 13% of Group 2 were completely wrong, and 2% and 4% of the questions were not answered by groups 1 and 2, respectively. From the overall analysis of the answers, it was possible to infer that the evaluators were able to understand the domain since they answered the competence questions that guided the construction of ONTO-ResAsset with a satisfactory success rate (on average 70.5%). It was also observed that the questions with lower scores in both groups refer to the concepts of reuse asset management (55% of correct answers by Group 1 and 61% by Group 2). This may indicate that the concepts of reuse asset management and their relationships require a more detailed textual description.

## 5. Conclusion and Future Work

The main contribution of this work refers to the establishment of ONTO-ResAsset – an ontology that represents and documents, in a single place and format, knowledge about the specification and management of assets that can be reused during the fundamental activities of software construction. It is important to highlight that the proposed ontology contributes to the research in the area of software reuse, gathering the existing knowledge available in several relevant sources of information in the literature.

From the evaluation performed, we also noticed that ONTO-ResAsset is able to adequately represent the knowledge about specification and management of assets, being possible to integrate the proposed ontology with other ontologies in the area of software reuse.

As future works, we highlight the need of: (i) implementing ONTO-ResAsset with the support of an ontology editor and knowledge acquisition system (such as Protégé); (ii) evolving ONTO-ResAsset to represent the taxonomic groups in more details; and (iii) evaluating ONTO-ResAsset with a greater number of non-experts in order to obtain more concrete and significant results.

## References

- [1] E. D. Almeida, A. Alvaro, V. C. Garcia, J. C. Mascena, V. A. Burégio, and L. M. Nascimento. *C.R.U.I.S.E. Component Reuse In Softw. Eng.* Cesar e-books, 2007.
- [2] N. Bajnaid and R. Benlamri. Software quality assurance ontology: from development to evaluation. In *25th Int. Conf. on Softw. Eng. and Knowledge Eng.*, pages 689–694, Boston, USA, 2013.
- [3] V. A. Burégio. Specification, design and implementation of a reuse repository. Master’s thesis, Federal University of Pernambuco, Recife (PE), Brazil, 2006. in portuguese.
- [4] M. Ezran, M. Morisio, and C. Tully. *Practical Software Reuse*. Practitioner Series. Springer London, 2002.
- [5] R. A. Falbo. *Knowledge Integration in a Software Development Environment*. PhD thesis, Federal University of Rio de Janeiro, Rio de Janeiro (RJ), Brazil, 1998. in portuguese.
- [6] M. Fernández-López, A. Gómez Pérez, and N. Juristo. Methontology: From ontological art towards ontological eng. In *Ontological Eng. Spring Symposium Series*, pages 33–40. American Association for Artif. Intelligence, 1997.
- [7] T. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *Int. Journal Human-Computer Studies*, 43(5-6):907–928, 1995.
- [8] Institute of Electrical and Electronics Engineers and Electronics Industry Association. *IEEE/Std 12207 - Systems and software engineering — Software life cycle processes ISO/IEC 12207:2008*. Standard, Feb. 2008.
- [9] B. M. Konda and K. K. Mandava. A systematic mapping study on software reuse. Master’s thesis, School of Computing, Department of System and Softw. Eng., Blekinge Inst. of Technology, Ronneby, Sweden, 2010.
- [10] X. Li and W. Zhang. Ontology-based testing platform for reusing. In *17th Int. Conf. on Internet Comp. for Eng. and Science*, pages 86–89, Los Alamitos, CA, USA, 2012.
- [11] D. Lucrédio, D. D. Brito, A. Alvaro, V. C. Garcia, R. P. Fortes, and S. L. Meira. Softw. reuse: The brazilian industry scenario. *Softw. reuse: The Brazilian industry scenario*, 81(6):996–1013, 2008.
- [12] OMG. Reusable asset specification. Technical report, Obj. Manag. Group, 2005.
- [13] H. Paulheim and F. Probst. UI2Ont—A Formal Ontology on User Interfaces and Interactions. In T. Hussein, H. Paulheim, S. Lukosch, J. Ziegler, and G. Calvary, editors, *Semantic Models for Adaptive Interactive Systems*, Human-Computer Interaction Series, pages 1–24. Springer London, 2013.
- [14] L. E. P. Silva. ONTO-ResAsset: Reusable Assets Ontology. Master’s thesis, School of Computing, Federal University of Mato Grosso do Sul, Campo Grande (MS), Brazil, 2013. in portuguese. Available at: <http://www.facom.ufms.br/gestor/titan.php?target=openFile&fileId=1210>.
- [15] S. Singh and G. Singh. Reusability of the softw. *Int. Journal of Computer Applications*, 7(14):38–41, 2010.
- [16] I. Sommerville. *Software Engineering*. Pearson Education, 9 edition, 2010.
- [17] M. Uschold. Building ontologies: Towards a unified methodology. In *16th Annual Conf. of the British Computer Society Specialist Group on Expert Systems*, pages 16–18, December 1996.

# Extending RBAC Model

## to Control Sequences of CRUD Expressions

Óscar Mortágua Pereira  
Instituto de Telecomunicações  
DETI, University of Aveiro  
Aveiro, Portugal  
omp@ua.pt

Diogo Domingues Regateiro, Rui L. Aguiar  
Instituto de Telecomunicações  
DETI, University of Aveiro  
Aveiro, Portugal  
{diogoregateiro,ruilaa}@ua.pt

*Abstract*—In database applications, access control is aimed at supervising users' requests to access sensitive data. Users' requests are mainly formalized by Create, Read, Update and Delete (CRUD) expressions. The supervision process can be formalized at a high level, such as based on the RBAC model, but in the end the relevant aspect is the data being accessed through each CRUD expression. In critical database applications access control can be enforced not on a CRUD by CRUD basis but enforced at the level of sequences of CRUD expressions (workflow). This situation can occur whenever established security policies are based on strict procedures that define step by step the actions (sequences of CRUD expressions) to be followed. Current RBAC models do not support this type of security policies. To overcome this security gap, we leverage previous researches to propose an extension to the RBAC model to control for each role which sequences of CRUD expressions are authorized. We demonstrate empirical evidence of the effectiveness of our proposal from a use case based on Java and JDBC. Our use case is based on typed security layers built from a software architectural model and also from metadata based on the proposed RBAC model extension.

*Keywords*—information security, access control, RBAC, software architecture, software engineering, components.

### I. INTRODUCTION

Access control [1][2] “is concerned with limiting the activity of legitimate users.” [3]. Four of the main strategies for regulating access control policies are: discretionary access control (DAC) [3], mandatory access control (MAC), attribute-based access control [4][5] (ABAC) and Role-based access control (RBAC) [6][7]. There are other strategies for regulating access control, such as credential-based access control (CBAC) [Li, '05; Yu, '03], content driven [Moffett, '91; Staddon, '08], location driven [Decker, '08], public key driven [Wang, '11] and certificate driven [Samarati, '01b]. Each one addresses specific security needs for the system under protection. In this paper we are focused on RBAC, which has emerged as one of the dominant access control policies [8], namely for relational database applications. RBAC policies comprise several concepts, among them: users, roles (they can be hierarchized), permissions, delegations and actions. Basically, legitimate (authenticated) users can only execute some action if he has been authorized to play the role that rules that action. At the end, actions are the four main operations on database objects (tables and views) defined by the data manipulation language

of the SQL standard: Insert, Select, Update and Delete, herein referred to as Create, Read, Update and Delete (CRUD) expressions, respectively. Depending on the granularity and the used technique, the authorization to execute these actions can be defined at the level of database objects, at the level of columns, at the level of rows and at the level of cells. Another relevant aspect that has not been addressed by current RBAC models is the sequence in which CRUD expressions are executed. Changing the order in which CRUD expressions are executed can lead to disclosing not authorized data. For example, it is very usual to use values from a previous Select expression as runtime values for subsequent Select expressions. If this sequence is not enforced, security violations can occur because the provenance of the used runtime values cannot be guaranteed [9]. To overcome this situation, in this paper we propose an extension to the RBAC model to support the definition of sequences in which CRUD expressions must be executed. We demonstrate empirical evidence of the effectiveness of our proposal from a use case based on Java and JDBC.

This paper is organized as follows: section II presents the related work; section III presents our proposal; section IV presents the proof of concept and, finally, section V presents the final conclusion.

### II. RELATED WORK

To the best of our knowledge no other researches have been conducted to provide RBAC models with the capability of controlling the sequences in which CRUD expressions are executed. Therefore, in this section we will present two main groups of aspects that are also closely related to this research: access control and service composition.

#### A. Access Control

In this sub-section we present access control in two main areas: models and techniques.

Models - Sandhu et al. [10] proposed the RBAC96, which comprises four models: RBAC0, RBAC1, RBAC2 and RBAC3. Since then, several proposals have been presented to extend these four RBAC models, among them we emphasize: credential based access control [11], temporal based access control [12], role delegation [13][14], context-aware [15],

system-to-system [16]. These and the remaining extensions are mainly focused on refining the role concept in order to adapt to particular contexts and scenarios.

Techniques - Several techniques have been proposed to protect the access to data, among them we emphasize: protection at tables and views level (vendors of RDBMS), the use of views [17], the use of parameterized views [18], the use of query rewriting techniques [19][20][17][21][22], extensions to SQL [23][24], programming languages extensions [25][26][27], security programming languages and tools [28][29][30][23][31] and, finally, semantic access control [32][33][34][35][36]. These proposals are based on a bundle of different techniques, each one with its own features. In spite of their relevance, none of them addresses the key issue of this research. They are mainly focused on controlling accesses to database objects (tables and views) on a CRUD by CRUD basis. A different technique is proposed in [37] where a security framework evaluates, at runtime, sequences of CRUD expressions in order to preserve data privacy. Beyond degrading the system performance this technique still needs to be subject to further experimentation.

Authors of this paper have also published about dynamic and distributed access control mechanisms [38][9]. Their researches were focused on techniques to implement static access control mechanisms at the level of business tiers. They were never focused on how to enforce sequences of CRUD expressions.

#### B. Service Coordination: Orchestration and Choreography

In our proposed approach, we needed to control the order in which users are authorized to use CRUD expressions. Two of the technologies that are used to control services workflow are: 1) service orchestration, which requires every service to be requested by a central control point; 2) service choreography, which allows a service to request the next service. Standard languages for each technique were proposed. Regarding orchestration, OASIS defined a standard language, which is still very active, called Web Services Business Process Execution Language [39] (WS-BPEL). WS-BPEL provides a set of functionalities that largely exceeds our needs. We will use some functionalities similar to those provided by WS-BPEL but tailored to our specific needs, such as graphs and life-cycle operations of active entities. Regarding choreography, the Web Service Choreography Description Language [40] (WS-CDL) is a language from W3C aimed at describing choreographies using the global view of the observable behavior of web services. However, the W3C Web Services Choreography working group was closed in 2009, leaving WS-CDL just as a candidate recommendation.

Several other languages exist, such as Yet Another Workflow Language [41] (YAWL) and XML Process Definition Language [42] (XPDL), but they clearly would not bring any advantage to our case.

The necessities to control the order in which sequences of CRUD expressions are executed are closer to a choreography process than an orchestration process. This way, we decided to implement our own technical approach in spite of having some

similar features with the orchestration process, as already mentioned.

### III. PROPOSED RBAC EXTENSION

This section is focused on presenting our RBAC extension. We start by presenting some of our previous work that is reutilized in this research, then we present the conceptual extension to be used in RBAC policies and, finally, a model extension is presented.

#### A. Business Schema

Before delving into the policy and its model, we start by analyzing some previous researches that we have been conducted around CRUD expressions [9][10][11][12][45]. From these researches we have defined and used the concept of Business Schema, which is basically a model from which source code can be automatically generated to handle CRUD expressions. The model, as in [9][37], can be driven by access control policies. Beyond being a model, Business Schemas have a cardinality of many to many with CRUD expressions. This means that one Business Schema can handle one or more CRUD expressions and one CRUD expression can be handled by one or more Business Schemas. Let us consider the next two Select expressions:

1) Select \* from table; 2) Select \* from table where col>10;

First we analyze the direction “one Business Schema -> many CRUD expressions”. Both expressions are Select and both have zero runtime values and the schema of the returned relations is the same. Then, the same Business Schema can be shared by both expressions. Now we analyze the direction “one CRUD expression -> many Business Schemas”. This case is simpler to explain and we can pick up any of the two Select expressions. In cases where different security policies are applied to the same Select expression, then we can use the Select expression in more than one Business Schema. For example, the same CRUD expression is managed by two Business Schemas where the runtime values are driven by different security policies.

#### B. RBAC Policy Extension

In this sub-section we present the new extension to the RBAC policy that is used to control the access requests to the data stored in relational database management systems (RDBMS). Traditionally, among other concepts, RBAC policies comprise: users, roles (they can be hierarchized), permissions, delegations and actions. Basically, legitimate (i.e. authenticated) users can only execute an action if he is authorized to play the role that controls that action. If a user is authorized to play a certain role, then he can perform all the actions controlled by that role. At the end, actions are the four main operations, provided by the Data Manipulation Language, on database objects (tables and views): read, insert, update and delete. The extension here presented aims at providing RBAC policies with the capability of controlling at the role level: 1) which Business Schemas and CRUD expressions are authorized; 2) in which sequence Business Schemas can be activated (instantiated) and, finally 3) the life-cycle of Business Schemas when the sequence moves forward one position. From this extension, security experts can now define new restrictions over the actions ruled by a role, particularly the ordered

sequences of actions (execution of CRUD expressions) users can perform.

### C. RBAC Model Extension

In this sub-section we propose a model, shown in Figure 1, to formalize the extension proposed to the RBAC policy. This model is not unique and other formalizations can be used, depending on the practical scenario at hand. The extension herein proposed leverages our previous work where we proposed Business Schemas to model the access to relational databases based on CRUD expressions. The extension must take into account two main aspects. The first aspect is the functionality to connect Business Schemas and, therefore, to build sequences of Business Schemas. The second aspect is related to the life-cycle of Business Schemas when the sequence moves forward to the next Business Schema.

We start by presenting the first aspect. We will use directed graphs (usually known as digraphs) theory to formalize our approach for sequences ( $S$ ) of Business Schemas. Basically, one vertex ( $v$ ) is one Business Schema and one directed edge ( $e$ ) connects one vertex (source vertex) to the next vertex (destination vertex). From a general digraph, there is the possibility to define several paths (sequence of vertices with each adjacent pair connected by a single direct edge). In an access control context, this type of freedom can raise some alerts when applied in systems where security is a key concern. Therefore, although our model relies on digraphs, some strict constraints need to be enforced. The main restriction lies on the impossibility of using digraphs in their widest scope. Nevertheless, we can start by designing digraphs, although, from them we have to identify the paths that are considered to be in accordance with the access control policies. A path is a sequence of vertices, each one with one edge only (except the last one, which has no edge). Only these valid paths can be used and assigned to roles. This means that users authorized to play a role: 1) can only execute the Business Schemas (vertices) defined by the associated path and 2) in the order they are defined in the path (direct edges). In order to allow the assignment of several paths to one role, our model does not enforce any restriction at that level. This can be important in situations in which a role is defined to control the use of several forms, each one controlled with its own path, this way avoiding the need to define one role for each form.

Some definitions are now introduced. A root vertex ( $r$ ) is the vertex where one path start. A leaf vertex ( $l$ ) is a vertex with no edges and, therefore, is the last vertex of a path. A front vertex ( $f$ ) is the vertex in which the sequence is running. Now we define the properties of paths in our model. Most of the concepts are shared by the theory of graphs. Even though, we present them to provide the necessary background for those who are not comfortable with graphs. The properties of paths are:

- one path comprises one and only one root vertex;
- one path comprises one and only one leaf vertex;
- self-edged vertices are not allowed in paths;
- loop-back vertices are not allowed in paths;
- any vertex of a digraph can be the root vertex of a path;
- any vertex of a digraph can be the leaf vertex of a path;

- any vertex of a digraph can appear zero, one or more times in one path;
- adjacent vertices in paths must also be adjacent vertices in the parent digraph and connected with the same direct edge.

Next we present a simple example of a digraph, see Figure 1. It comprises 5 vertices and 6 edges, one of them is a self-edge (on vertex C). From this digraph, an indeterminate number of different paths can be defined. This indeterminate number of different paths has its origin on loop-back vertices (B to A) and self-edges (on C). Three of the possible paths that can be defined from Figure 1 are shown in Figure 2.

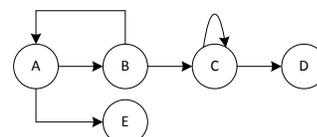


Figure 1. Example of a digraph.

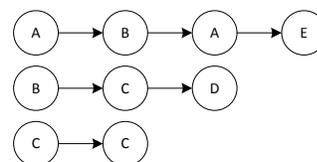


Figure 2. Examples of valid paths derived from Figure 1.

The second aspect is related to life-cycle of Business Schemas when a sequence moves to the next Business Schema. Once again, our approach provides a model in where security managers can freely choose the best option for their real scenarios. Basically, when the sequence moves to the next Business Schema, it is up to the security manager to decide which from the active Business Schemas are to remain active (their instances run normally) and which are not to remain active (their instances are running but they are disabled – methods do not execute the expected actions). The process to disable Business Schemas instances is herein referred to as the revocation process. In our model, each Business Schema has an associated list with all the previous Business Schemas to be revoked.

Finally, we present a possible and simplified model to formalize our proposal, see Figure 3. From it we can see that one role comprises one or more sequences (paths); a sequence comprises one or more Business Schemas (vertices); each Business Schema manages one or more CRUD expressions and, finally, each Business Schema (vertex - in a certain sequence and in a certain position) has a list of Business Schemas to be revoked.

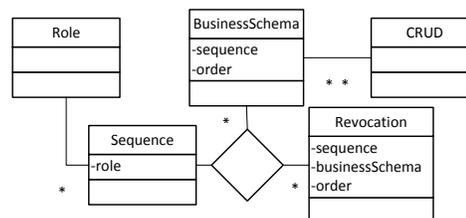


Figure 3. Example of a simplified diagram for our RBAC model extension.



each CRUD expression it supports.

Business Manager - Business Manager entity is the key entity in our architectural model. It is responsible for three main tasks: 1) to ensure that Business Schemas are instantiated in the correct order (sequences are kept in *sequences*); 2) to revoke active Business Schemas that are listed in the revocation list (*SequencyEntry - revokeList*) and, finally, 3) to validate calls to any Business Schemas' methods. This last task is required because when a Business Schema instance is revoked we need to programmatically ensure that all methods are put in a disabled state (in Java there is no possibility to explicitly destroy objects – they are destroyed by the garbage collector only when there is no reference to those objects).

The interaction between applications and our architectural model is as follows (please follow Figure 6): 1 - application selects one of the available sequences and requests the instantiation of its first Business Schema (in our example BusinessSchema\_1) to manage the execution of one of the authorized CRUD expressions (there is one method for each Business Schema – CRUD expression pair); 1.1 – BusinessManager instantiates the Business Schema; 2 – Factory returns (to the application) a reference to the instantiated Business Schema 1; 2.1 – application uses the Business Schema 1 to interact with the host database; 2.1.1 – the call to every method is validated by Business Manager (if

the Business Schema has been revoked, an exception is raised); 2.2) application requests the instantiation of BusinessSchema\_2 (this is validated by Business Manager and eventually Business Schema 1 is revoked in case it is defined in the revocation list of Business Schema 2); 3 – Business Schema 2 is instantiated (only in case it has been validated by Business Manager) and a reference is returned to the application; 3.1 – application uses Business Schema 2 instance and 3.1.1 – every call to its methods is validated by Business Manager.

#### D. Use Case

From this architectural model and from metadata based on the proposed RBAC model extension, security layers are automatically built, see Figure 4. Our use case is based on Java, JDBC and uses the Microsoft Northwind database (<http://www.microsoft.com/download/en/details.aspx?id=23654>).

Table 1 and Table 2 partially present the information used in our use case, only for one role (Role\_B1). Table 1 shows the supported Business Schemas and CRUD expressions. Table 2 shows the supported sequences. The bottom of both tables has some additional information to help with the understanding of their content. Now we present some snapshots of our concrete use case. Only some snapshots will be given in order to not overcrowd the paper. Figure 7 shows the high level data structures (classes are not shown) that were automatically created for Role\_B1 and from the metadata represented by

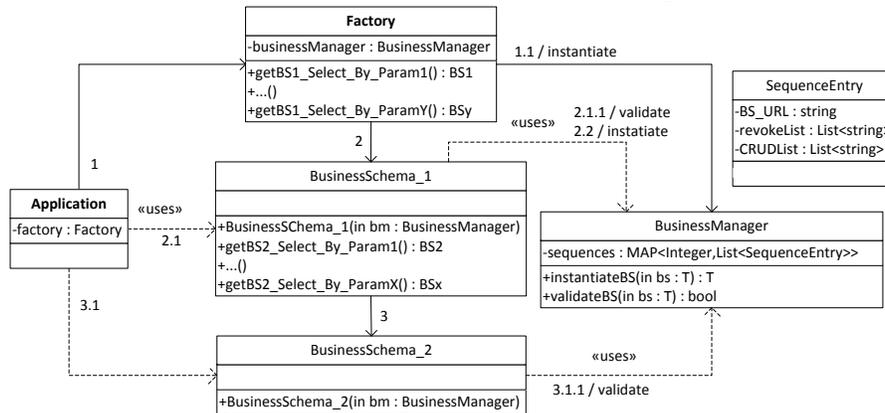


Figure 6. Software architectural model for the RBAC extension (one role).

Table 1. Roles, Business Schemas and CRUD expressions.

Role	Parent Role	BS	CRUD		
			Id	Ref	Expression
Role_A					
Role_B1	Role_A	S_Orders	1	byShipCountry	Select * From Orders Where CustomerId = ? and ShipCountry = ?
			2	byFreightLimit	Select * From Orders Where CustomerId = ? and Freight < ?
		I_Orders	3	withCustomerID	Insert Into Orders Values (?,?,?,?,?,?,?,?,?)
		S_Customers	4	all	Select * From Customers
Role:	Role Reference				
BS:	Business Schema alias				
Id:	CRUD Identification				
Ref:	CRUD reference				
Expression:	CRUD Expression				

Table 2. Roles, sequences and revocation lists.

Role	Sequence	Position	Sequence Entry		
			BS	RL	CRUDs
Role_B1	1	1	S_Customers		4
		2	S_Orders		1, 2
	2	1	I_Orders		3
		2	S_Customers	I_Orders	4
		3	S_Orders		1
Role:	Role Reference				
Sequence:	The sequence identification				
Position:	The position in the sequence				
BS:	Business Schema alias				
RL:	The revocation list				
CRUD:	The list of authorized CRUDs				

Table 1 and Table 2. They comprise the required information about the authorized Business Schemas and CRUD expressions

for Role\_B1, only. These data structures are used during instantiation process of Business Schemas as we will show. Figure 8 presents the way applications interact with Factory classes. Factory classes are also built from metadata based on the extended RBAC model and have one method for each pair *Business Schema – CRUD expression*. From Figure 8 we see that the pair (Business Schema+CRUD expression) represented by *S\_Customers\_all* is requested to be instantiated (line 60) and then the CRUD expression is executed (line 61). Instances of root Business Schemas are instantiated by the Business Manager class under Factory's requests as shown in Figure 9 for the Business Schema *S\_Customers\_all*. Here we can see that the pair represented by *S\_Customers\_all* is *s\_customers* and *s\_customers\_S\_Customers\_all* (lines 28, 29), which are internal data structures, shown in Figure 7, and, therefore, they

```

7 public abstract interface Role_IRole_B1 {
8     public static final java.lang.Class<II_Orders>
9         i_orders = II_Orders.class;
10    public static final int
11        i_orders_I_Orders_withCostumerID = 2;
12    public static final java.lang.Class<IS_Orders>
13        s_orders = IS_Orders.class;
14    public static final int
15        s_orders_S_Orders_byShipCountry = 1;
16    public static final java.lang.Class<IS_Customers>
17        s_customers = IS_Customers.class;
18    public static final int
19        s_customers_S_Customers_all = 3;
20 }

```

Figure 7. Data structure comprising information about *Role\_B1*

```

60 S_Cust = factory.get_S_Customers_all(session);
61 S_Cust.execute();

```

Figure 8. interaction between application and the Factory class.

```

25 public IS_Customers get_S_Customers_all(ISession session)
26 throws LocalTools.BTC_Exception {
27     return businessManager.instantiateBS(
28         Role_IRole_B1.s_customers,
29         Role_IRole_B1.s_customers_S_Customers_all,
30         session);
31 }

```

Figure 9. The Factory requests an instance for a Business Schema.

```

65 S_Orders = S_Cust.get_S_Orders_by_shipCountry(session);
66 S_Orders.execute(S_Cust, "Portugal");

```

Figure 10. The root Business Schema provides an edge method to step forward to next Business Schema.

```

17 @Override
18 public IS_Orders get_S_Orders_by_shipCountry(ISession session)
19 throws BTC_Exception {
20     if(!businessManager.validateExecution(
21         activeSequence, "S_Customers.S_Customers")) {
22         throw new IllegalStateException(
23             "S_Customers was used out of order!");
24     }
25
26     return businessManager.instantiateBS(
27         Role_IRole_B1.s_orders,
28         Role_IRole_B1.s_orders_S_Orders_byShipCountry,
29         session,
30         activeSequence
31     );
32 }

```

Figure 11. Validation process for the instantiation process of Business Schemas.

are not accessible from the application side. In order to access the second Business Schema, the root Business Schema provides a method to that goal, see Figure 10 (line 65). Before being instantiated, Business manager evaluates if the requested Business Schema can be instantiated, see Figure 11. If authorization is not granted, an exception is raised (line 22-23). Otherwise, an instance is returned (line 26-31).

These are only some snapshots of our use case. From these snapshots, from Table 1 and Table 2, and from Figure 6 we hope that readers can infer the remaining technical approaches on which our use case was built.

## V. CONCLUSION

This paper presents an extension of the basic RBAC policy in order to control the sequence in which CRUD expressions are executed. The model leverages previous researches where the RBAC model was extended to support the concept of Business Schema and, therefore, the execution of CRUD expressions. To formalize our proposed extension we start by resorting to the theory of graphs. From it we defined an ordered sequence of Business Schemas as being a path. Basically, the model extension allows security experts to designate the set of Business Schemas sequences to be used by users authorized to play that role. A proof of concept is also presented, which, once again, leverages previous researches. The presented proof of concept, beyond providing the empirical evidence of the effectiveness of our proposal, it also conveys the following key advantages: 1) security layers are automatically built and 2) security layers are based on typed objects, this way relieving programmers of application tiers from mastering the established security policies.

## REFERENCES

- [1] P. Samarati and S. D. C. di Vimercati, "Access Control: Policies, Models, and Mechanisms," *Found. Secur. Anal. Des.*, vol. 2171, pp. 137–196, 2001.
- [2] S. D. C. di Vimercati, S. Foresti, and P. Samarati, "Recent Advances in Access Control - Handbook of Database Security," M. Gertz and S. Jajodia, Eds. Springer US, 2008, pp. 1–26.
- [3] R. S. Sandhu and P. Samarati, "Access Control: Principle and Practice," *Commun. Mag. IEEE*, vol. 32, no. 9, pp. 40–48, 1994.
- [4] M. A. Al-Kahtani and R. Sandhu, "A Model for Attribute-Based User-Role Assignment," *Proceedings of the 18th Annual Computer Security Applications Conference*. IEEE Computer Society, pp. 353–362, 2002.
- [5] D. R. Kuhn, E. J. Coyne, and T. R. Weil, "Adding Attributes to Role-Based Access Control," *Computer (Long. Beach. Calif.)*, vol. 43, no. 6, pp. 79–81, 2010.
- [6] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli, "Proposed NIST Standard for Role-based Access Control," *ACM Trans. Inf. Syst. Secur.*, vol. 4, no. 3, pp. 224–274, 2001.
- [7] R. Sandhu, D. Ferraiolo, and R. Kuhn, "The NIST Model for Role-based Access Control: Towards a Unified Standard," *5th ACM Workshop on Role-based Access Control*. ACM, Berlin, Germany, pp. 47–63, 2000.
- [8] L. Fuchs, G. Pernul, and R. Sandhu, "Roles in information security – A survey and classification of the research area," *Comput. Secur.*, vol. 30, no. 8, pp. 748–769, 2011.
- [9] Ó. M. Pereira, R. L. Aguiar, and M. Y. Santos, "Runtime Values Driven by Access Control Policies Statically Enforced at the Level of the Relational Business Tiers," in *SEKE'13 - Intl. Conf. on Software Engineering and Knowledge Engineering*, 2013, pp. 1–7.
- [10] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-Based Access Control Models," *Computer (Long. Beach. Calif.)*, vol. 29, no. 2, pp. 38–47, 1996.

- [11] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis, "The KeyNote Trust-Management System Version 2." RFC Editor, United States, 1999.
- [12] E. Bertino, P. A. Bonatti, and E. Ferrari, "TRBAC: A Temporal Role-based Access Control Model," *ACM Trans. Inf. Syst. Secur.*, vol. 4, no. 3, pp. 191–233, 2001.
- [13] X. Zhang, S. Oh, and R. Sandhu, "PBDM: A Flexible Delegation Model in RBAC," in *Proceedings of the Eighth ACM Symposium on Access Control Models and Technologies*, 2003, pp. 149–157.
- [14] E. Barka and R. Sandhu, "Framework for Role-based Delegation Models," in *Proceedings of the 16th Annual Computer Security Applications Conference*, 2000, p. 168–.
- [15] D. Kulkarni and A. Tripathi, "Context-aware role-based access control in pervasive computing systems," *13th ACM Symposium on Access Control Models and Technologies*. ACM, Estes Park, CO, USA, pp. 113–122, 2008.
- [16] S. Haibo and H. Fan, "A Context-Aware Role-Based Access Control Model for Web Services," in *Proceedings of the IEEE International Conference on e-Business Engineering*, 2005, pp. 220–223.
- [17] S. Rizvi, A. Mendelzon, S. Sudarshan, and P. Roy, "Extending Query Rewriting Techniques for Fine-grained Access Control," *ACM SIGMOD Int. Conf. on Management of Data*. ACM, Paris, France, pp. 551–562, 2004.
- [18] A. Roichman and E. Gudes, "Fine-grained access control to web databases," *12th ACM symposium on Access Control Models and Technologies*. ACM, Sophia Antipolis, France, pp. 31–40, 2007.
- [19] Oracle, "Using Oracle Virtual Private Database to Control Data Access," 2011. [Online]. Available: [http://docs.oracle.com/cd/B28359\\_01/network.111/b28531/vpd.htm#CIHBAJGI](http://docs.oracle.com/cd/B28359_01/network.111/b28531/vpd.htm#CIHBAJGI).
- [20] K. LeFevre, R. Agrawal, V. Ercegovac, R. Ramakrishnan, Y. Xu, and D. DeWitt, "Limiting disclosure in hippocratic databases," *30th Int. Conf. on Very Large Databases*. VLDB Endowment, Toronto, Canada, pp. 108–119, 2004.
- [21] Q. Wang, T. Yu, N. Li, J. Lobo, E. Bertino, K. Irwin, and J.-W. Byun, "On the correctness criteria of fine-grained access control in relational databases," *33rd Int. Conf. on Very Large Data Bases*. VLDB Endowment, Vienna, Austria, pp. 555–566, 2007.
- [22] S. Barker, "Dynamic Meta-level Access Control in SQL," *22nd Annual IFIP WG 11.3 Working Conf. on Data and Applications Security*. Springer-Verlag, London, UK, pp. 1–16, 2008.
- [23] A. Chlipala, "Static checking of dynamically-varying security policies in database-backed applications," in *9th USENIX Conf. on Operating Systems Design and Implementation*, 2010, pp. 1–14.
- [24] S. Chaudhuri, T. Dutta, and S. Sudarshan, "Fine Grained Authorization Through Predicated Grants," *IEEE 23rd ICDE - Int. Conf. on Data Engineering*. Istanbul, Turkey, pp. 1174–1183, 2007.
- [25] B. J. Corcoran, N. Swamy, and M. Hicks, "Cross-tier, Label-based Security Enforcement for Web Applications," *35th SIGMOD Int. Conf. on Management of Data*. ACM, Providence, Rhode Island, USA, pp. 269–282, 2009.
- [26] J. Fischer, D. Marino, R. Majumdar, and T. Millstein, "Fine-Grained Access Control with Object-Sensitive Roles," *23rd ECOOP - European Conference on Object-Oriented Programming*. Springer-Verlag, Italy, pp. 173–194, 2009.
- [27] J. Zarnett, M. Tripunitara, and P. Lam, "google.pt," *Proceedings of the 15th ACM symposium on Access control models and technologies*. ACM, Pittsburgh, Pennsylvania, USA, pp. 79–88, 2010.
- [28] L. Caires, J. A. Pérez, J. C. Seco, H. T. Vieira, and L. Ferrão, "Type-based access control in data-centric systems," *20th European conference on Programming Languages and Systems: part of the joint European conferences on theory and practice of software*. Springer-Verlag, Saarbrücken, Germany, pp. 136–155, 2011.
- [29] N. Swamy, B. J. Corcoran, and M. Hicks, "Fable: A Language for Enforcing User-defined Security Policies," in *IEEE Symposium on Security and Privacy*, 2008, pp. 369–383.
- [30] D. Zhang, O. Arden, K. Vikram, S. Chong, and A. Myers, "Jif: Java + information flow (3.3)," 2012. [Online]. Available: <http://www.cs.cornell.edu/jif/>.
- [31] C. Ribeiro, A. Zúquete, P. Ferreira, and P. Guedes, "SPL: An Access Control Language for Security Policies with Complex Constraints," *Network and Distributed System Security Symposium*. San Diego, CA, USA, pp. 89–107, 2001.
- [32] Y.-J. Hu and J.-J. Yang, "A semantic privacy-preserving model for data sharing and integration," *Proceedings of the International Conference on Web Intelligence, Mining and Semantics*. ACM, Sogndal, Norway, pp. 1–12, 2011.
- [33] C.-C. Pan, P. Mitra, and P. Liu, "Semantic access control for information interoperation," *Proceedings of the eleventh ACM symposium on Access control models and technologies*. ACM, Lake Tahoe, California, USA, pp. 237–246, 2006.
- [34] J. Warner, V. Atluri, R. Mukkamala, and J. Vaidya, "Using semantics for automatic enforcement of access control policies among dynamic coalitions," *Proceedings of the 12th ACM symposium on Access control models and technologies*. ACM, Sophia Antipolis, France, pp. 235–244, 2007.
- [35] J. Lopez, A. Mana, E. Pimentel, J. M. Troya, and M. I. Y. e del Valle, "Access Control Infrastructure for Digital Objects," *Proceedings of the 4th International Conference on Information and Communications Security*. Springer-Verlag, pp. 399–410, 2002.
- [36] K. Il Kim, W. Y. Kim, J. S. Ryu, H. J. Ko, U. M. Kim, and W. J. Kang, "RBAC-based access control for privacy preserving in semantic web," *Proceedings of the 4th International Conference on Uniquitous Information Management and Communication*. ACM, Suwon, Republic of Korea, pp. 1–5, 2010.
- [37] Canfora, G.; Visaggio, C.A.; Paradiso, V., "A Test Framework for Assessing Effectiveness of the Data Privacy Policy's Implementation into Relational Databases," in *Intl. Conf. on Availability, Rliability and Security*, 2009, pp. 240–247.
- [38] Ó. M. Pereira, R. L. Aguiar, and M. Y. Santos, "ACADA - Access Control-driven Architecture with Dynamic Adaptation," in *SEKE'12 - 24th Intl. Conf. on Software Engineering and Knowledge Engineering*, 2012, pp. 387–393.
- [39] OASIS, "WS-BPEL Especification." [Online]. Available: <http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf>.
- [40] W3C, "Web Services Choreography Description Language." [Online]. Available: <http://www.w3.org/TR/ws-cdl-10/>.
- [41] "YAWL." [Online]. Available: <http://yawlfoundation.org>.
- [42] "XML Process Definition Language." [Online]. Available: <http://www.xpdl.org/>.
- [43] O. M. Pereira, R. L. Aguiar, and M. Y. Santos, "CRUD-DOM: A Model for Bridging the Gap Between the Object-Oriented and the Relational Paradigms," in *ICSEA 2010 - Int. Conf. on Software Engineering and Applications*, 2010, pp. 114–122.
- [44] O. M. Pereira, R. L. Aguiar, and M. Y. Santos, "CRUD-DOM: A Model for Bridging the Gap Between the Object-Oriented and the Relational Paradigms - an Enhanced Performance Assessment Based on a case Study," *Int. J. Adv. Softw.*, vol. 4, no. 1&2, pp. 158–180, 2011.
- [45] O. M. Pereira, R. L. Aguiar, and M. Y. Santos, "An Adaptable Business Component Based on Pre-defined Business Interfaces," in *6th ENASE: Evaluation of Novel Approaches to Software Engineering*, 2011, pp. 92–103.
- [46] ISO, "ISO/IEC 9075-3:2003," 2003. [Online]. Available: [http://www.iso.org/iso/catalogue\\_detail.htm?csnumber=34134](http://www.iso.org/iso/catalogue_detail.htm?csnumber=34134).
- [47] Microsoft, "Microsoft Open Database Connectivity," 1992. [Online]. Available: [http://msdn.microsoft.com/en-us/library/ms710252\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms710252(VS.85).aspx).
- [48] M. Parsian, *JDBC Recipes: A Problem-Solution Approach*. NY, USA: Apress, 2005.
- [49] M. David, "Representing database programs as objects," in *Advances in Database Programming Languages*, F. Bancilhon and P. Buneman, Eds. N.Y.: ACM, 1990, pp. 377–386.

# Assisting Software Projects with Bug Report Assignment Recommender Creation

John Anvik, Marshall Brooks, Henry Burton and Justin Canada

Department of Computer Science  
Central Washington University  
Ellensburg, WA 98926  
{janvik,brookmar,burtonh,canadaj}@cwu.edu

**Abstract**—Software development projects receive many change requests each day and each report must be examined to decide how the request will be handled by the project. One decision that is frequently made is to which software developer to assign the change request. Efforts have been made toward semi-automating this decision, with the most promising approaches using machine learning algorithms. However, using machine learning to create an assignment recommender is a complex process that must be tailored to each individual software development project. This paper presents the Creation Assistant for Easy Assignment (CASEA), a tool that leverages a project member’s knowledge to assist in creating an assignment recommender specific to the software development project.

**Keywords**—bug report triage; assignment recommendation; machine learning; recommender creation

## I. INTRODUCTION

Large software development projects can receive hundreds of bug reports per day [1]. Each of these bug reports needs to be analyzed to determine if the report will result in development activity. For example, a report may be examined in order to determine whether the report is a duplicate or if the problem is reproducible. In cases where the problem needs action from a developer, a decision needs to be made about to whom the work will be assigned. This decision process is called *bug triage* and must be done for all incoming reports.

Bug triage takes significant time and resources [2]. Bug report assignment recommenders have been proposed as a method for reducing this overhead. Conceptually, the creation of an assignment recommender using machine learning is straightforward. Six questions need to be answered to create an assignment recommender [1]. These questions relate to which bug reports to use and how many, determining who fixed a report, which developers to recommend, what data from the bug reports to use, and which algorithm to use.

In practice, the creation of such recommenders for a specific software development project is challenging. If a project member were to begin creating an assignment recommender for their project s/he would first have to research machine learning libraries and algorithms, as well as methods for obtaining bug reports from their repository, and that’s just for a basic recommender. Also, finding answers to these questions requires significant experimentation to construct an optimal recommender due to interdependencies between questions. For example, how the reports are labelled with

developer names affects the set of names recommended, and the choice of valid names affects which reports and how many of them will be used for creating the recommender. Finally, further research may need to be done to find ways to better format the data such as word stemming, parts of speech analysis and feature reduction. Creating an assignment recommender could take many hours to days or weeks for someone who is unfamiliar with the process.

Some answers for assignment recommender creation appear to be consistent across projects (e.g. the text from the bug report summary and description is used as the data [1] [3] [4]). Other questions are specific to the project. For example, how to determine which developer fixed a bug requires the use of heuristics, and these heuristics differ from project to project [1]. The heuristics that are used for labeling reports from one project, such as the Eclipse Platform project<sup>1</sup>, are not the same as the heuristics used for another project, such as the Firefox project<sup>2</sup>. Creating these heuristics is a time consuming process. It was observed to take between half a day to a full day depending on the complexity of the heuristics [1].

Establishing these heuristics once may not be sufficient. The recommender may need to be reconfigured if any significant changes are made in a project’s use of the issue tracking system. For example, a project may add or deprecate bug lifecycle states over time. Such changes would require a reconfiguration of the project-specific labeling heuristics and the set of reports used for creating the assignment recommender.

The initial configuration and ongoing adjustments for creating an assignment recommender represents a cost to the project. For example, the Mozilla project at one time had four different web browser projects. There existed subsets of developers that either worked exclusively on one of these projects, or overlapped with one or more these projects. If the Mozilla project had decided to incorporate an assignment recommender into its development process at that time, then a different assignment recommender would have had to be created for each subproject. At the limit, the Mozilla project, which currently has thirty-six subprojects, would have to create and maintain thirty-six different assignment recommender configurations. This could pose a substantial cost to the project.

<sup>1</sup> <http://www.eclipse.org/jdt>

<sup>2</sup> <http://www.mozilla.org/firefox>

If the use of assignment recommenders is to be made practical, the costs associated with creating and maintaining recommender configurations needs to be reduced. As reducing the number of different configurations that will need to be created and maintained by a project is unlikely, seeking to lower the costs associated with each individual configuration offers a viable solution. Also, the fact that the techniques for creating an assignment recommender extend to other types of triage recommenders [1] makes reducing such costs all the more important.

This paper presents the Creation Assistant for Easy Assignment (CASEA), an implementation of the approach proposed in [1] that leverages a project member's knowledge to assist in creating and managing bug report assignment recommender configurations. Specifically, CASEA assists in the creation of project-specific heuristics and the selection of valid developer recommendations. CASEA also allows project members to experiment with these two parameters to quickly determine if the use of an assignment recommender would benefit a project. Finally, CASEA helps project members to ensure that the assignment recommender is trained with the best data possible by considering the actual data being used to train the recommender.

This paper proceeds as follows. First, background information about bug reports and machine learning is presented. Next, details of how CASEA uses a project member's knowledge to create an assignment recommender are given. The paper then concludes with related work.

## II. BACKGROUND

This section presents background information about bug reports, their life cycles, and machine learning.

### A. Bug Reports

Bug reports, also known as change requests, provide a means for users to communicate software faults or feature requests to software developers. They also provide a means for developers to manage software development tasks.

Bug reports contain a variety of information, some of which is categorical and some of which is descriptive. Categorical information includes such items as the report's identification number (i.e. bug id), its resolution status (e.g., NEW or RESOLVED), the component the report is believed to involve, and which developer has been assigned the work. Descriptive information includes the title of the report, the description of the report, and discussions about possible approaches to resolving the report. Finally, a report may contain other information, such as attachments or links to other reports.

### B. Bug report lifecycles

All bug reports have a lifecycle, although the specific states can vary between projects. When a bug report first enters a project's issue tracking system (ITS), it is in a state such as UNCONFIRMED or NEW. The bug report then moves through different states, depending on the project's development process, and arrives at a resolution state, such as FIXED or INVALID. The lifecycle of a bug report can be used to categorize bug reports [1].

### C. Machine learning algorithms

Machine learning is the development of algorithms and techniques that allow computers to learn [5]. Machine learning algorithms fall under three categories: supervised learning, unsupervised learning, and reinforcement learning. Bug report assignment recommenders primarily use supervised learning algorithms, such as Support Vector Machines (SVM) and Naïve Bayes.

Understanding how a machine learning algorithm creates a recommender requires understanding three concepts: the feature, the instance and the class. A *feature* is a specific piece of information that is used to determine the class, such as a term that appears in one or more of a set of bug reports. An *instance* is a collection of features that have specific values, such as all of the terms in the description of a specific bug report. Finally, a *class* is the collection of instances that all belong to the same category, such as all of the bug reports fixed by a developer. In supervised machine learning, training instances are labeled with their class. A recommender is created from a set of instances and the output of the recommender is a subset of the classes to which the instances belong.

## III. ASSISTING ASSIGNMENT RECOMMENDER CREATION

This section presents CASEA, a software tool to assist a software project in creating and maintaining bug assignment triage recommenders. Although CASEA only supports the creation of assignment recommenders, it can be extended to assist with the creation and maintenance of other triage recommenders. Through the use of CASEA, a user would be able to use their project knowledge to create an assignment recommender in a short period of time.

CASEA guides a project member through the assignment recommender creation process in four steps: Data Collection, Data Formatting, Recommender Training, and Evaluation. The remainder of this section presents the details of how CASEA assists with each of these steps.

### 1) Data Collection

The first step in recommender creation is to gather the data to be used for creating the recommender. Specifically, bug reports are extracted from the project's issue tracking system. CASEA provides two methods for data collection.

The first option is to import two XML files containing the bug report data: one file containing the data for the reports to be used for training, and the other containing data to be used for evaluation. On importing the file, CASEA's Document Object Model (DOM) is populated.

The other option is for the project member to provide the URL of the project's ITS. An XML-RPC library is used to pull the data from the ITS and populate the DOM model. The user specifies the initial bug report creation date from which to start pulling reports, and the number of reports to gather. Reports that have a resolution status of RESOLVED, VERIFIED or CLOSED are chronologically gathered from the starting date. Every tenth report collected is designated as a testing report to create an unbiased set for evaluation.

## 2) Data Formatting

Having collected the data from the project's ITS, the next step is to format the data into a collection of instances to be used for training the recommender and evaluating its performance. However, before the instances are created, the data is first filtered to produce the highest quality training set. Two types of filtering are performed: automatic and assisted.

### a) Automatic Filtering

Recall that an instance contains a label and a set of features. For bug report assignment the features are the terms that occur in the summary and description of the bug report. The automatic filtering performs three actions on the textual data.

First, terms that are stop words are removed. Stop words are commonly occurring words that provide little to no information content. Examples of stop words include 'a', 'the', and 'an'. After stop words are removed, the next step is stemming. Stemming reduces all of the terms to their respective root values. Stemming is done so that words such as 'user' and 'users' are treated as the same word, ensuring a common vocabulary between the reports. Finally, punctuation and numeric values are removed, except where the punctuation is important to the term, such as class names (e.g. "org.eclipse.jdt") or URLs.

### b) Assisted Filtering

CASEA assists the user with two types of filtering: label filtering and instance filtering. In the case of assignment recommender creation, label filtering is the selection of developer names to be recommended and instance filtering is the selection of the bug reports to be used for recommender training and evaluation.

To assist with label filtering, CASEA presents the user with a label frequency graph. For an assignment recommender, this graph presents bug fixing statistics for the project developers based on all the reports in the training data set. For an assignment recommender, one would want to recommend from a core set of developers, not from the set of all developers, as a developer who has only solved one report in the past is unlikely to solve another. CASEA allows the user to select a threshold using a slider, such that only a core set of developers are recommended. By default, CASEA sets the threshold to five resolved reports. As the user moves the slider, the horizontal red cutoff line is adjusted.

Instance filtering is done using project-specific heuristics. The heuristics have two parts: a grouping rule and a label source. The grouping rule is used to categorize the data into groups for which the label source will be used for labeling the instances. For an assignment recommender, the grouping rule is a bug report lifecycle and the label source is either a field from the bug report, such as the *assigned-to* field, or other labelling information that can be extracted from the bug report, such as the user that attached a patch or marked the report as resolved. The default label source is the *assigned-to field*.

The lifecycle states occurring between the initial state and the resolution state may affect which field(s) can be used for labelling. For example, bug reports with the lifecycle NEW → FIXED would never be formally assigned to a project developer and were probably resolved by the person who

created the report. Therefore the *reporter* field is likely to be the correct label source for this group of bug reports. For bug reports that follow the lifecycle NEW → ASSIGNED → FIXED, the *assigned\_to* field is likely the best labeling choice given that the report was triaged and an appropriate developer was selected.

All of the reports in the training data set are used to determine the specific bug report life cycles for the project and the user is presented with a statistical summary of the categories. Using this information, the user can create heuristics for the most common occurring categories. For each of these categories, the user then selects the label source from a drop down, including a choice to not label reports in the group.

The user can also choose the number of heuristics to be applied, to a maximum of ten. An observation from using CASEA on a number of projects has shown that approximately 70% of reports are covered by the top five bug report lifecycles. To label the reports not covered by a configured heuristic, the user selects a default label source.

As was mentioned, dependencies exist between choices made in configuring an assignment recommender. In CASEA, there is a dependency between the two types of assisted filtering. The values in the label frequency graph come from the application of the labelling heuristics to the data, and changes to the set of heuristics cause the label filtering graph to be updated. In other words, the developer activity graph reflects the set of names from the use of the current heuristics.

### c) Formatting

After filtering the data to create the set of training and evaluation instances for the recommender, the data is then formatted for use with the machine learning algorithm. Specifically, two tables are created. First, a term frequency table is created where each column is a feature from the instances and the rows are instance ids. For an assignment recommender, the rows are bug report ids and the columns are terms. The content of the table is the term-frequency/inter-document frequency (tf-idf) values for each term in each report of the data set. Tf-idf provides a measure of the importance of a term within a document. This table is only created for the training instances.

The second table is an index of instances to labels. In this case, the table is an index of bug report ids to developer name given by the heuristics. This table is created for both the training and evaluation data sets.

## 3) Recommender Training

Once the user has filtered the data and the data is formatted, the recommender is created using a machine learning algorithm. The two most commonly used machine learning algorithms for assignment recommenders are Support Vector Machines and Naïve Bayes [1] [4]. CASEA uses a multi-class SVM algorithm with a Gaussian kernel for creating a recommender. The time to train a recommender was found to be between a few seconds and a few minutes, depending on the size of the training dataset, the configuration parameters, and the computer used.

#### 4) Evaluation

Once the user starts the recommender creation process, the user is moved to an Analysis tab that presents progress information, such as the time taken to train the recommender and evaluation results. The user can then return to the Configuration tab, adjust the values for label and instance filtering, and create a new recommender. This process continues until the user is either satisfied with the created recommender, or the user has determined that an assignment recommender cannot be created with a high enough accuracy to benefit the project. At any time the user can save the recommender and return at a later date.

To evaluate the performance of the recommender, the measures of precision and recall are used. Precision and recall are defined as follows:

$$\text{Precision} = \frac{\# \text{correct recommendations}}{\# \text{recommendations}}$$

$$\text{Recall} = \frac{\# \text{correct recommendations}}{\# \text{with correct expertise}}$$

To accurately compute these values, the set of developers that had the expertise to resolve each bug report is needed. However, this information can be difficult to determine and approximations are necessary [1]. CASEA uses a list of developers that resolved reports in the same component as an approximation of bug fixing expertise. Note that this approximation will overestimate developer expertise, trading off a more accurate precision value for a less accurate recall value. As high precision is favored over high recall this tradeoff is acceptable. CASEA presents results for the top one, three and five ranked recommendations.

### IV. RELATED WORK

#### A. Developer Assignment Recommendation

Many prior researchers have proposed using assignment recommenders to reduce the amount of resources to be put towards triage, such as [1] [3] [4] [6] [7]. All previous work has sought to find specific answers to the assignment recommender creation questions; however none of these works, except for [1], have addressed the problem of making the use of such recommenders practical.

#### B. Assisting with Recommender Creation

Making the use of machine learning recommenders practical has been done in other areas, such as bioinformatics. However, these tools typically focus on explaining the results of the recommendations [8] or helping to visualize how machine learning works [9].

There are two commercial products that could be considered similar to CASEA. Skytree Server [10] is similar in intent to CASEA; however to use Skytree Server still requires advanced knowledge of machine learning and statistics [11]. Big ML [12] also provides assistance with using machine learning, but is restricted to a single machine learning algorithm shown to be not effective for assignment recommendation [1].

### V. CONCLUSION

This paper presented CASEA, a tool to assist in the creation of bug report assignment recommenders. CASEA assists a user in labelling and filtering the bug reports used for creating a project-specific assignment recommender. Although there have been many efforts towards determining a good approach to assignment recommender creation, these approaches do not address how to leverage a project member's knowledge in the creation of the recommender. CASEA seeks to remedy this situation.

### REFERENCES

- [1] J. Anvik and G. C. Murphy, "Reducing the Effort of Bug Report Triage: Recommenders for Development-oriented Decisions," *ACM Trans. on SE and Methodology*, vol. 20, no. 3, 2011.
- [2] Y. Cavalcanti, P. Neto, I. Machado, E. de Almeida and S. de Lemos Meira, "Towards understanding software change request assignment: a survey with practitioners," in 17th Inter. Conf. on Eval. and Assess. in SE, 2013.
- [3] G. Jeong, S. Kim and T. Zimmermann, "Improving bug triage with bug tossing graphs," in ACM SIGSOFT Symposium on the Foundations of SE, 2009.
- [4] P. Bhattacharya and I. Neamtii, "Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging," in IEEE Inter. Conf. on Software Maintenance, 2010.
- [5] T. Mitchel, *Machine Learning*, McGraw-Hill, 1997.
- [6] R. Shokripour, J. Anvik, Z. M. Kasirun and S. Zamani, "Why so complicated? simple term filtering and weighting for location-based bug report assignment recommendation," in 10th IEEE Inter. Working Conference on MSR, 2013.
- [7] H. Naguib, N. Narayan, B. Brügge and D. Helal, "Bug report assignee recommendation using activity profiles," in 10th IEEE Inter. Working Conference on MSR, 2013.
- [8] B. Poulin et al, "Visual Explanation of Evidence in Additive Classifiers," in 18th Conf. on Innovative Applications of AI, 2006.
- [9] B. Noris, "ML Demos," [Online]. Available: <http://mldemos.epfl.ch/>. [Accessed 28 February 2014].
- [10] Skytree Inc., "Skytree Server," [Online]. Available: <http://www.skytree.net/products-services/skytree-server>. [Accessed 28 February 2014].
- [11] S. Charrington, "Three New Tools Bring Machine Learning Insights to the Masses," [Online]. Available: <http://readwrite.com/2012/02/27/three-new-tools-bring-machine>. [Accessed 3 Feb 2014].
- [12] BigML, "BigML is Machine Learning," [Online]. Available: <https://bigml.com/>. [Accessed 28 February 2014].

# Recovering Valuable Information Behaviour from OSS Contributors: An Exploratory Study

Mário André de F. Farias<sup>\*§</sup>, Paulo Ortins<sup>\*</sup>, Renato Novais<sup>†</sup>, Methanias Colaço Júnior<sup>‡</sup>, Manoel Mendonça<sup>\*¶</sup>

<sup>\*</sup>Federal University of Bahia <sup>†</sup>Federal Institute of Bahia <sup>‡</sup>Federal University of Sergipe <sup>§</sup>Federal Institute of Sergipe <sup>¶</sup>Fraunhofer Project Center for Software and Systems Engineering, Bahia, Brazil

Email: mario.andre@ifs.edu.br, paulo.ortins@gmail.com, renato@ifba.edu.br, mjrse@hotmail.com, manoel.mendonca@ufba.br

## Abstract

**Context.** Distributed software development is currently a modern practice in software industry. This is especially true in Open Source Software (OSS) development community. Understanding how developers' practices are on those projects may guide communities to successfully manage their projects. **Goal.** We mined two repositories of the Apache Httpd project in order to gather information about its developers' behavior. **Methods.** We developed an approach to cross data gathered from mail list and source code repository through mining techniques. The approach uses software visualization to analyze the mined data. We conducted an experimental evaluation of the approach to assess the behavioral patterns from OSS development community. **Results.** The collected data built a rich set of information. The results highlight Apache developers' behavior patterns. We perceived that there is a correlation over time between emails and commits of the Apache Project developers. **Conclusion.** The use of data mining and software visualization to analyze crossing data sources can spot important properties of the development process.

## I. INTRODUCTION

Software repositories have been used to discover useful knowledge about the development, maintenance and evolution of software. However, these data sources are not built in a structured and organized way. So, it is necessary a considerable effort to gather evidence from those repositories. To this end, researchers have been developing different approaches [1, 3, 5, 10, 13, 14]. They use data mining, software visualization (SoftVis), text mining and mining software repository. Some of them analyze each repository separately, even that combining different techniques (e.g. metrics, SoftVis) is a promising approach [11].

According to Kagdi et al. [6], the mining software repositories activity combines different areas, such as: knowledge discovery [7] and information retrieving [8]. Combine approaches from different areas is a promise strategy. For example, enrich those techniques with information visualization may produce solutions that aim to reveal software properties that were initially hidden in these environments.

Based on this premise, this paper presents an exploratory study that uses data mining and SoftVis techniques to analyze

OSS developers' behavior. The study is particularly interested in analyze discussion mailing lists and source code repositories through the use of SoftVis. The main characteristic of the used approach is the integration between the repositories data. It aims to integrate and analyze data originated from Apache Httpd mailing list and source code data.

This paper is organized as follows. Section II presents related works. Section III describes our experimental evaluation. Section IV reports and discusses our findings. Finally, Section V concludes the paper with a discussion of future work.

## II. RELATED WORK

This section discusses some related work concerned with identifying patterns in development community through mining repository software and software visualization. Heller et al. [5] proposed a strategy that mined a GitHub repository metadata and used visualization techniques to identify patterns in development community. The study focused on specific patterns, such as the effect of geographic distance on developer relationships, social connectivity and influence among cities, and variation in project specific contribution styles.

Some works have already considered email specific analysis to study OSS development process and behavior of people [4, 12]. In [4], the authors investigated the impact of computer-mediated interaction on person perception. In particular, they studied how important traits for socialization and collaboration may be detected from the text of an email communication. Pattison et al. Finally, [12] studied the relation between several software entities mentioned in emails and the number of times these entities were included in the changes made.

Two works are closest to the research presented here. First, Canfora et al. [1] mined explicitly documented cross-system bug fixings from versioning repository and data from two project mailing lists. It aimed to identify Cross-System-Bug-Fixings activities between FreeBSD and OpenBSD, and to understand the social role of developers performing such activities by means of social network analysis. We based our cross-system mailing list in this work. Second, in [3], the authors performed an experimental evaluation of the approach. They conducted an experiment to assess the PRS of top developers at Apache server mailing lists.

### III. EXPERIMENTAL EVALUATION

The experimental process follows the guidelines by [15]. Here, we present the experimental definition and the planning process. The next section presents the gathered evidence and the results.

The main goal of our study is to reveal interesting behavioral patterns in OSS contributions, such as the crossing analysis of mailing lists contribution, commits in the projects, and geographic location (geo-location) of contributions in OSS.

#### A. Planning

1) *Context Selection*: The experiment context was open source project repositories. These repositories have large amount of e-mails and commits. Commonly, the data is not ready to use. It is necessary to clean the data to avoid misleading understanding. For example, there is an arduous manual work of searching for valid emails and commits data. For that, we developed powerful computational procedures following [2, 3]. On top of that, we did a detailed manual analysis of the committers' profiles in order to gather geographic information. The approach study followed three steps: First, we extracted data from: a) Apache's commits repository; b) Apache developer's mailing list; and c) geographic information from geo-location services; Second, we crossed the data collected in the previous step in order to associate the data to the developer that produced it; and finally, we built interactive visualizations that help users to discover relevant information. Next, we present each of these steps in detail.

Three modules were developed by us for this study in order to provide an able environment to integrate and analyze two different source repositories. The first is a module for Extraction, Transformation and Load (ETL) of emails. The second module is for mining source code repositories. Finally, the third is a visualization module. These visualizations are publicly available at (<http://goo.gl/RSS4VR>).

2) *Research Questions*: This work aims investigate OSS Developers' behavior. To do this, we mined two software repositories in order to analyze the research questions addressing the distribution of email and commits over time in the project. Our research questions are described as follows:

- i. How commits and emails are distributed over time among the Apache Project developers?
- ii. How commits and emails are distributed over time by Apache Project top committers?

3) *Participant and Artifact Selection*: In this study, we used as object of analysis the Apache Httpd Project (<http://httpd.apache.org/>). Over its 17 years of development, the project received more than 60,000 commits, totalling more than two millions of lines of code written by more than 100 developers around the world. These developers use a mailing list to communicate with each other.

To answer our research questions, we extracted and analyzed the body of 100,479 email messages and 33,586 commits from the Apache repositories between 1995 and 2005. We selected the four developers who had the greatest number of commits. We refer to these developers as "Dev A", "Dev B",

"Dev C" and "Dev D". We also grouped all the other developers, and refer to them as "Cluster", these developers represent the rest of population. We analyzed the same developers and same period used in the related study [3].

4) *Preparation*: The experimental study preparation took two months. One month and three weeks for understanding and develop the modules and one week for the preparation and execution of a pilot. The pilot study was carried using a small sample of emails and commits which was chosen at random. Thus, the pilot helped us to calibrate some specific characteristics of our modules and to find improvement point, such as performance of the crossing data and geographic information.

#### B. Experiment execution

We retrieved the geographic information (latitude/longitude and the time offset) for each committer aiming to know the origin of commits and emails. Apache Httpd project does not have this data for all developers. They provide this information only for the core committers (the ones who contributes more to the project).

In these cases, Apache Httpd project provides a page with complementary information about them. Unfortunately, core committers represent only 63 from the total developers (110). For the others, we needed to perform a manual task to retrieve their geographic information. They also have different time offsets. This brings out another issue, since it is necessary to consider the time zone when collecting the weekday and time for each commit. In this case, we needed to get each developer's time zone and adjust the times for the Apache server time.

After retrieving geographic information, we found more 27 profiles, totalling 90 from the 110 available developers. We decided remove the commits from those developers which we could not find geographic information. So, we reduced the amount of analyzed commits from 33,586 to 31,611. Even considering all crossing data process, it was not possible to find a match between some emails and commits. In those cases, the data was ignored. This corresponded to 29,698 emails. So, we considered in our analysis 70,781 emails.

To establish a link between extracted data from email and commits we needed that at least one property was shared by both data sources. For example, either an ID or an email in both data sources should be equal. However, this was not possible, since, in Apache Http project, the data sources have different users' profiles. This was another challenge in our approach. It was necessary to match different kinds of data, e.g. email address with nickname and name with nickname. In order to perform this task, we adapted the approach proposed by [1].

### IV. RESULTS AND DATA ANALYSIS

The collected data built a rich set of information. Nonetheless, in general, data extracted from software repositories are too difficult to be analyzed in the same state that they were stored [9]. Thus, we decide to use visualizations to reorganize them in such manner that users can easily understand the

whole database. We discuss now the results of this study. To answer the research questions, we analyzed the data taking into consideration (i) the relation between emails and commits of the Apache Project developers (ii) the relation between emails and commits of the Apache Project top committers (iii) The beginning of Apache Project.

*i. Relation between emails and commits of the Apache Project developers:* Through the interaction with the period filter to generate heat maps over the time, we perceive that the heat zones (regions where contributions were made) used to appear first in the emails' map and after in the commits' map, it's evidence that in this project developers first interact in the email list and after commit code to the repository.

We could confirm this behaviors in the Apache Httpd Web Site. According with the site, changes to the code are proposed and voted on the mailing list and only after they are approved, they are committed in the repository. On top of that, we could also identify that the regions that have more participation in the emails list are also the regions with more participation in the code repository (see Fig. 1). An exception is the Japanese developers' behavior. In this case, there is a considerable amount of commits (bottom of Fig.1) but a low participation in the discussion mailing list (top of Fig.1 ). It may suggest an introverted behavior due to cultural factors.

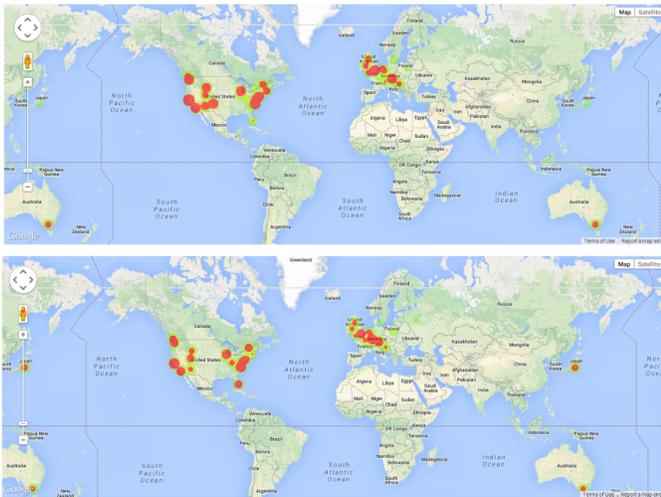


Fig. 1: Heat maps showing amount of (a) emails and (b) commits

We also perceived that there is a correlation between emails and commits timestamps. Developers normally commit code and discuss in the list in the same time as well in the same weekday. Figure 2.a shows the interactions over time between emails and commits of the Apache Project developers. If we analyze the evolution, we may confirm that commits and emails follow the same pattern distribution.

*ii. Relation between emails and commits of the Apache Project top committers:* For a more detailed analysis, our crossed data approach allows us to explore individually data from every developer in the project. Due the restriction of this paper's size, we choose to analyze the relation between emails sent and commits made only by the top committers. The Fig.

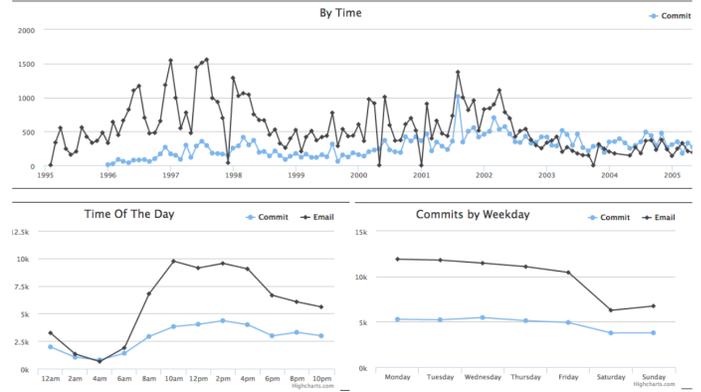


Fig. 2: Interactions over time between emails and commits of developer population

3 shows these distributions for each one over the time. Through these charts is possible to identify when a developer starts to contribute and when he leaves the project. According the Httpd's Site, both developers A and B, are considered emeriti members due their contributions, however, they left the project, this happening was also perceived in our visualization. The other two developers remained active, at least, until 2005.

Other finding that may be perceived is the work pattern; there are two evident work patterns in the charts, one pattern is the developer that the major part of his contribution was made between Monday and Friday and between 8 am to 6 pm, this pattern can be observed for the developers A, B, C. While the other pattern is the developer that the major part of his contributions was made during the weekend and at night, this pattern can be perceived for the developer D. The first pattern can be evidence that the developer was working in the project as a daily job or as part of his research; it's not uncommon for companies to pay open source developers to maintain tools that they use or students that contribute to open source projects as part of their research. The other pattern can be evidence that the developer is an open source enthusiast and has a daily job during the day and like to contribute in his spare time.

*iii. The Apache Project beginning:* Analyzing carefully the data evolution in Fig. 2, we could see that the discussion in the email list started a year before the first commits. This is an interesting behavior, since they had time to discuss before to start the implementation. However, it is common to observe OSS projects starting on the other way around. First, a project is created with few developers. They start to commit and create the first release. After that, users start to use it. Using the software, bug fix and request for new features will rise. So, users use a bug tracking system to report the issues. At the end, the developers start the discussions in the email list.

We decided to investigate why the Apache http evolved in this way. We looked the website and discovered that the Apache Httpd was a continuation of NCSA Httpd, which stopped to be developed when Rob McCool left NCSA in 1994. A group of webmasters then started to develop their own extensions and bug fixes, in 1995, they solved to join all

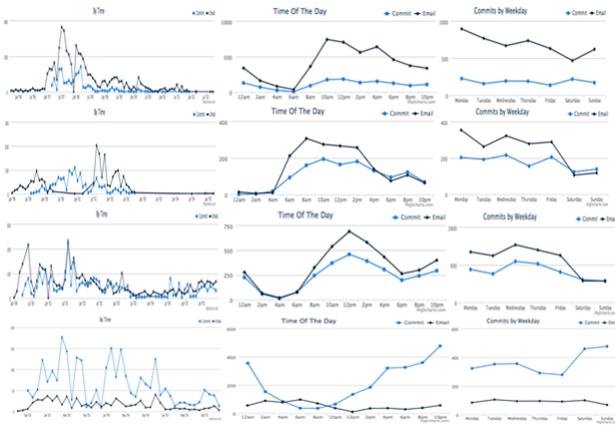


Fig. 3: Distribution of emails and commits of the Apache Project top committers. (a) Dev A, (b) Dev B, (c) Dev C, (d) Dev D.

this features and bug fixes in a unique distribution and then the Apache Group was created.

## V. CONCLUSION AND FURTHER WORK

In this paper, we presented an useful and innovative approach that extract information from two important data sources in a software project. We mined and try to match emails list and source code repository data. This approach can be used to discover hidden behavioral patterns in unstructured databases. We also believe that our approach can be used by OSS leaders to increase developers' contributions or to keep contributors in their projects. OSS managers can also use our approach to split tasks accordingly to each developers' profile or to tracking team's contributions over time considering weekdays and day periods.

We have evidences that discussion lists and repositories can be used to measure project activity or to predict each other. We now draw answers to our research questions stated in the section III. Regarding RQ1, we may confirm that commits and emails follow the same pattern distribution in the Apache evolution. Considering RQ2, we could see that this relation is very similar to rest of developers. Besides, we identified from our visualizations is possible to find some work pattern.

Our future work will address three key issues: (1) Improve our approach by extracting other relevant data from other OSS. This work is in process; and (2) Extend this study to mine data from Postgree, emails and commits, aiming to compare to findings performed by [2, 3]; and (3) Develop new interactive visualizations.

## REFERENCES

[1] G. Canfora, L. Cerulo, M. Cimitile, and M. Di Penta. Social interactions around cross-system bug fixings: The

case of freebsd and openbsd. In *MSR*, pages 143–152, 2011.

[2] M. Colaço, M. Mendonça, M. Farias, P. Henrique, and D. Corumba. A neurolinguistic method for identifying oss developers' context-specific preferred representational systems. *ICSEA*, page 112 to 121, Nov 2012.

[3] M. Colaço, M. Mendonca, M. Farias, and P. Henrique. Oss developers context-specific preferred representational systems: A initial neurolinguistic text analysis of the apache mailing list. *MSR*, pages 126–129, May 2010.

[4] A. J. Gill and et al. Perception of e-mail personality at zero-acquaintance: Extraversion takes care of . . .

[5] B. Heller, E. Marschner, E. Rosenfeld, and J. Heer. Visualizing collaboration and influence in the open-source software community. In *MSR*, pages 223–226, 2011.

[6] 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. *JSME*, 19(2):77–131, Mar. 2007.

[7] O. Maimon and L. Rokach. *Data Mining and Knowledge Discovery Handbook*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[8] C. D. Manning, P. Raghavan, and H. Schütze. *Introd. to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[9] R. Mazza. *Introduction to Information Visualization*. Springer, 1 edition, 2009.

[10] R. Novais, C. Nunes, A. Garcia, and M. Mendonca. Sourceminer evolution: A tool for supporting feature evolution comprehension. In *ICSM*, pages 508–511, Sept 2013.

[11] R. L. Novais, A. Torres, T. S. Mendes, M. Mendonça, and N. Zazworka. Software evolution visualization: A systematic mapping study. *IST*, 55(11):1860 – 1883, 2013.

[12] D. S. Pattison, C. A. Bird, and P. T. Devanbu. Talk and work: A preliminary report. In *MSR*, pages 113–116, New York, NY, USA, 2008. ACM.

[13] P. C. Rigby and A. E. Hassan. What can oss mailing lists tell us? a preliminary psychometric text analysis of the apache developer mailing list. In *MSR*, pages 23–, Washington, DC, USA, 2007. IEEE Computer Society.

[14] R. Witte, Q. Li, Y. Zhang, and J. Rilling. Text mining and software engineering: an integrated source code and document analysis approach. *Soft. IET*, 2(1):3–16, February 2008.

[15] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering: An Introduction*. Springer, Norwell, MA, USA, 2012.

# Measurement of the Non-Technical Skills of Software Professionals: An Empirical Investigation

Lisa L. Bender<sup>1</sup>, Gursimran S. Walia<sup>2</sup>, Fabian Fagerholm<sup>3</sup>, Max Pagels<sup>4</sup>, Kendall E. Nygard<sup>5</sup>, Jürgen Münch<sup>6</sup>

Department of Computer Science

University of Houston - Clear Lake<sup>1</sup>; North Dakota State University<sup>2,5</sup>; University of Helsinki<sup>3,4,6</sup>

[lbender@uhcl.edu](mailto:lbender@uhcl.edu); {[gursimran.walia](mailto:gursimran.walia)<sup>2</sup>, [Kendall.Nygaard](mailto:Kendall.Nygaard)<sup>5</sup>}@ndsu.edu; {[fabian.fagerholm](mailto:fabian.fagerholm)<sup>3</sup>, [max.pagels](mailto:max.pagels)<sup>4</sup>, [juergen.muench](mailto:juergen.muench)<sup>6</sup>}@cs.helsinki.fi

**Abstract**— Software development managers recognize that project teams need to be developed and managed. Although technical skills are necessary, non-technical (NT) skills are equally necessary for project success. There are several tools that assist in measuring the effectiveness of the technical skills that teams use to perform projects, but there are no proven tools to measure the NT skills of software developers. Behavioral markers (BM), observable behaviors that have positive or negative impacts on individual or team performance) are beginning to be successfully used by airline and medical industries to assist managers in assessing NT skills of project teams and individuals. The purpose of this research is to develop and validate a NT skills taxonomy for software developers. This paper presents an empirical investigation to develop and validate a NT skills taxonomy which was in turn used to construct a BM system tool for said developers and software development teams.

**Keywords**—Non-technical Skills; behavior marker; performance.

## I. INTRODUCTION

The software development process is a team activity and the success of a software project depends on the effective performance of the software project team. West [1] notes that working in a team is not automatically beneficial. Simply bringing people together does not ensure that they will function as an effective team. The Project Management Institute recognizes the need to develop project teams. The most recent PMBOK Guide [2] states “*teamwork is a critical factor for project success, and is one of the primary responsibilities of the project manager*”. PMBOK also acknowledges that non-technical (NT) skills, in comparison to technical skills, are equally important for project success and team development. Multiple authors agree that the NT skills are critical to project success [3, 4]. Other authors assert that NT skills can have the largest impact on software development [5, 6].

The cognitive and interpersonal skills which underpin software professionals and technical proficiency are being recognized as requirements for a competent software developer [7]. One major factor that is driving the demand for NT skills is the requirement for an agile workforce to support agile organizations [8]. Agile software development methodology is based on incremental and interactive development. This development is carried out through the collaboration between self-organizing, cross-functional teams. Agile teams depend greatly on efficient communication, taking responsibility, initiative, time management, diplomacy, and leadership.

While the performance of individuals is very important to creating an effective team, there are no established guidelines

for measuring team effectiveness. Different authors have identified different criteria for assessing team effectiveness [9, 10]. These criteria generally include measurements of task performance as well as the interpersonal skills of the team members, which include attitudes and behaviors. While there is extensive literature with respect to different ways to measure task performance (e.g., lines of code) for software development [11], little research has been performed on measuring NT skills. On that note, the aviation and health care industries have already recognized the importance of NT skills to the success of their teams, and have been using behavioral marker systems to structure individual and team assessments of these NT skills. We believe that software teams can also draw upon these models to improve teamwork in software development.

As software project development managers and educators, this is one of the factors that motivated our research. How can managers objectively measure the NT skills of their employees to determine if their NT skills are adequate or if they need improvement? How would feedback be provided to the team members so that they could improve their performance? The research reported here is an attempt to answer these kinds of questions. Thus, the purpose of this research is to identify the NT skills required by effective software professionals, and to develop a behavioral marker system for evaluating these skills.

## II. BACKGROUND

This section provides background on the NT skills (Section II.A) and the behavior marker systems (Section II.B).

### A. Non Technical (NT) Skills

NT skills are the cognitive, social, and personal resource skills that complement technical skills and contribute to overall task performance [12]. Classic examples of NT skills are leadership, patience, cooperation, communication, decision making, conflict management, stress and workload management, attention to detail, empathy, and confidence. In short, NT skills cover both the social and cognitive side of a person. In a survey released in 2013 by the Association of American Colleges and Universities [14], it was found that employers feel that NT skills, both cognitive and interpersonal, are more important than a particular major. Even professional organizations such as Professional Engineering Competence (UKSPEC), IEEE Computer Society etc. state that professional engineers have an obligation to possess NT skills [15].

Universities and colleges have strived to create curricula to prepare students to be Software Engineers. Some researchers have defined competencies (both technical and NT) for Undergraduate Software Engineering students, however these

do not encompass all of the competencies, such as many necessary interpersonal skills, needed for a Software Engineering professional [16, 17]. Other researchers have developed expert profiles (tools that communicate the technical and NT skills required) for engineering professionals that include input from both academia and industry; however, they do not define specific competencies required for a Software Engineer [18]. Educators summarize important course knowledge and skills that the student's should develop in course syllabi. Employers list minimum requirements for new hires in job advertisements. With so many different sources and kinds of information available, it is difficult to synthesize what competencies and in particular, NT skills, are required in the software profession. Therefore, this research attempts to develop a NT skills taxonomy for software professionals.

### B. Behavior Markers(BM) and Behavior Marker System

Behavioral markers (BM) are defined [19] as “*observable, non-technical behaviors that contribute to superior or substandard performance within a work environment*”. They are derived by analyzing data regarding performance that contributes to successful and unsuccessful outcomes. These markers are often structured into categories (e.g. communication, situational awareness, and decision making). The overall purpose of a BM system is to provide a method to assess team and or individual behaviors using markers. In general, BM systems have a taxonomy or listing of NT skills that are associated with effective job performance. This listing is combined with a rating scale to allow the skills (which are demonstrated through behaviors) to be assessed by trained observers. These BM systems are part of an observation-based method to capture and assess individual and team performance on data rather than on gut feelings. Observers use the BM tool, designed in the form of a structured list of skills, to rate skill and behavior performance. This allows an individual's or team's skills to be rated in their real context. BM systems can provide feedback on performance to individual/ teams as well as a common language for discussing and teaching NT skills.

BM systems have demonstrated value for assessing NT skills, for providing feedback on these skills, for improving training programs for NT skills, and in the use of building databases to identify norms and prioritize training needs. Given the prevalence and success of BM systems, we believe that they can be effective at improving NT skills in software development teams. However, a BM system developed for one domain cannot simply be transferred to another domain. It is important to recognize that BM systems need to be specific to the domain and culture. O'Conner et al. [20] noted that the Human Factors Analysis and Classification System developed for aviation was not appropriate for assessing the NT skills for U.S. Navy divers.

A brief description of the domains in which BM systems have been successfully used (i.e., airline, medicine) follows:

1) *Airline Industry*: The first BM system developed for the airline industry had two primary purposes: to evaluate the effectiveness of crew resource management (CRM) by measuring observable behaviors and to aid the development of future CRM programs [21]. The Line Operation Safety Audit

(LOSA) is a very successful BM system, and many of the BM systems in other industries were adapted from this audit tool. It focuses on interpersonal communication, leadership, and decision making in the cockpit. Trained observers (pilots and human factors experts) ride along in the cockpit and observe the flight crews during normal flight operations. They score the behaviors of the crew using LOSA. This tool has been very successful in measuring the strengths and weakness of the flight crews' interpersonal skills and is endorsed by the International Civil Aviation Organization [22].

2) *Medical Industry*: To help improve teamwork in healthcare, BM systems are being adopted. Two predominate tools to date include the Anesthetists' NT Skills (ANTS) System and the Observational Teamwork Assessment of Surgery (OTAS). ANTS provides a taxonomy for structured observations of anesthetists [23]. This system has proven very useful in assessing the NT skills of anesthetists in simulation training and has provided important performance feedback for the individuals. OTAS was developed to evaluate the technical and interpersonal skills in surgery teams [24]. Empirical studies have shown that the underlying cause of many adverse events in surgery were the result of poor communication, coordination, and other aspects of teamwork rather than technical failures. OTAS has been found to be a valid measure the technical and NT performance of surgical teams.

Our goal is to develop a BM system that can improve software professional team member performance by providing feedback in the form of an objective and documented assessment of the NT skills of the team members.

## III. RESEARCH APPROACH

The research approach included activities with a major focus on a) developing a taxonomy of NT skills of software professionals, b) validating and refining the taxonomy, and c) developing a BM system for software developers. More details on the study design and results appear here.

As a first step, a systematic literature review was performed to identify and analyze the NT skills of software professionals. A *systematic literature review* is a systematic search process that focuses on particular research question and provides an exhaustive summary of literature relevant to that question. By performing a systematic review, researchers can be more confident that they have found background information relevant to their study. The more common *ad hoc* approach does not provide this same level of assurance [25]. Next, a focus survey was developed using the process recommended by Davis et al. [26]. The focus groups consisted of employers, SE and CS industrial professionals and instructors. The software professional NT skills profile survey used the NT skills information gathered from the systematic literature review and was developed with the assistance of a focus group. The results of the survey were used to develop the software professional NT skills taxonomy. Finally, a review of related BM system literature was carried out in order to develop a BM system for software developers, and a BM rating tool was created.

## IV. RESEARCH DESIGN

This section describes the research design. Section IV.A describes the systematic review protocol. Section IV.B presents the focus group design, survey procedure, and the process of developing the BM system and the behavior examples.

### A. Systematic Literature Review

In accordance with systematic review guidelines [27], the review protocol was developed that specified the questions to be addressed, the databases to be searched and the methods to be used to identify, assemble, and assess the evidence. To properly focus the review, a set of research questions (RQ's) were developed. With the underlying goal to develop a software professional NT skills profile, the high-level question addressed by this review was:

*“What are the NT skills required of software professionals performing well in their field and how can we discover what NT skills are valued by employers?”*

This high-level question was then decomposed into the more specific RQ's. **RQ#1** identified the existing empirical studies reported on desired competencies in software professionals. **RQ#2** focused on efforts, methods or tools that are used to identify or can be used to identify a comprehensive list of NT skills. If any of these methods or tools has been implemented, we also analyzed their level of success and lessons learned. **RQ#3** combined the results of the first two RQ's in an attempt to develop a software professional NT skills profile. Details on the review protocol (sources searched, search execution, inclusion and exclusion criteria, quality assessment, data extraction) can be referred to in a report [28].

### B. Focus Group

After an initial list of NT skills was identified from the literature review, the skills were clustered into major categories (details in Section V). Synthesis of the literature review of software developer NT skills was then incorporated into a survey. This survey was sent to a diverse group of individuals from SE academia and industry for review. We employed two online surveys to assist in gathering this input. The focus group members who participated in these surveys were located in three different states, thus using a survey questionnaire was an efficient way of collecting the NT skills input. Both surveys used a cross-sectional survey design in which we gathered information about the NT skills important to a professional software developer at a specific point in time. First, the initial list of NT skills was compiled and then the first electronic survey was created. The first survey used an initial draft of NT skills gathered from the literature review as a basic guideline and then gathered NT skills priorities, missing NT skills, description clarifications, and comments to produce a more robust NT skills inventory. Once this survey was complete, an updated NT skills profile was created. The purpose of the second survey was to gather examples of good and poor behaviors for the top rated NT skills from the first survey. The details of the survey design appears in the subsections

#### 1) Survey Methodology

The surveys intend to evaluate the NT skills of a software developer performing well in professional practice, and identify the observable actions of the NT skills of a software developer.

**Survey Participants:** A group of 20 individuals (SE professors, and industry managers representing both publically and privately held companies from small to large software development departments) was asked to provide input on the list of NT skills. Because cultural differences have been found to have a significant impact on individuals [29], we decided to only seek input from educators and employers along the I-29 corridor of Minnesota, North Dakota, and South Dakota. Three universities (Dakota State University, North Dakota State University, University of Minnesota Crookston), along the I-29 corridor, were identified to have programs that would produce graduates suitable to being employed as Software Engineers and individuals were selected from each university. Each of the industry collaborators were selected because the companies they were associated with were located along the I-29 corridor; they all employed many new graduates that work in SE and software development related jobs; and they all have well developed human resource departments with sufficient resources to have created comprehensive competency expectations for company's employees and thus would have clearly defined expectations. The industry collaborators included managers of software professionals from each of the companies.

**Survey Procedure:** The survey included following 3 steps.

**Focus Group Survey #1:** The focus group was emailed an electronic survey and asked to rank the importance of each NT skill to software professionals. The skills were listed in the categories (discussed in the results section). The survey also included the descriptions for each skill (can be found in [28]). The ranking that we asked the focus group to produce provides prioritization of NT skills that most reflect expert activities. The focus group was also asked to provide inputs (suggested revisions to the NT skills, clarifications of the NT skill descriptions, missing elements, assess quality, and any further comments) to the NT skills. The quality of the NT skills was assessed per the guidelines provided by Davis and Beyerlein [18] by asking the focus group to provide feedback. The focus group helped create a more robust NT skill list.

**Compile High Priority List of NT Skills:** The results of this first survey were compiled into an improved NT skills taxonomy. Some competencies were re-grouped, and the list trimmed of the competencies that did not meet the quality standards. This more robust NT skill list only include the most highly prioritized NT skills, which was intended to make it easier for the focus group to complete the second survey.

**Focus Group Survey #2:** The second electronic survey sent to the focus group posed open-ended questions. The participants were asked to provide examples of observable actions that indicate good performance and behavior of each NT skill as well as examples of observable actions that indicate poor performance and behavior of each NT skill. They were asked to provide as many examples as they wished for each skill. The examples of good and poor behaviors were collected. Based on the inputs from the second survey, we developed a behavior-based software engineer NT skills taxonomy and used it as the basis for the behavioral marker system. The resultant examples of good and poor behavior for a subset of skills appear in the results section

## V. RESEARCH RESULTS

This section provides results and findings organized around the activities described in Section IV.

### A. Non Technical (NT) Skills based on Literature Review

After an initial list of NT skills was identified from the literature review, we clustered the skills into four major categories: *communication*, *interpersonal*, *problem solving*, and *work ethic*. These categories are illustrated in Fig. 1 and were reviewed by the involved researchers. We also had performed research to find meaningful descriptions for each skill. In many instances, it was felt that an identified skill overlapped with another NT skill, thus a list of synonyms was created to help provide clarity. Details of this information for one of the NT skills category (i.e., “*Communication*”) can be seen in Table I and more details appear in [28].

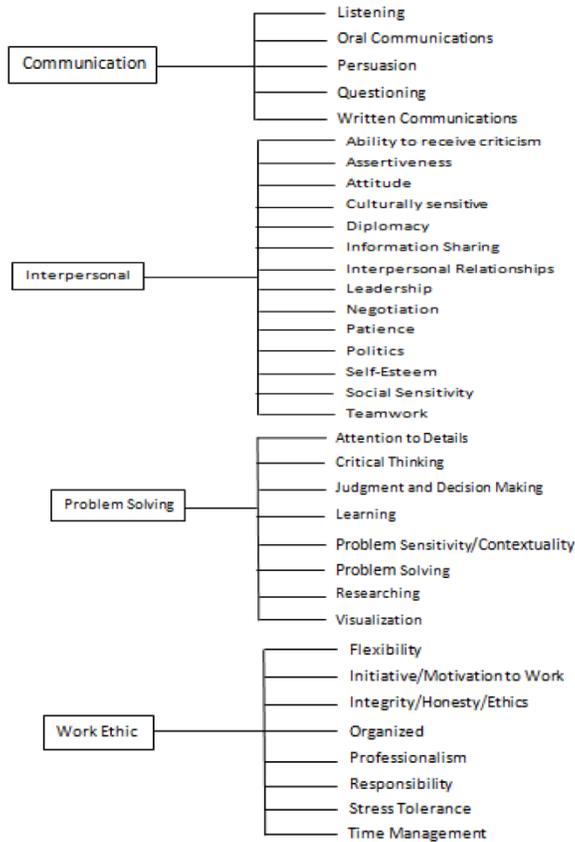


Fig. 1: Desired NT skills of Software Professionals

### B. Results from the Focus Group# 1 and # 2

As mentioned earlier, individuals during the focus group# 1 ranked the importance of each NT skill to software professionals. The skills were listed in the categories and in the same order (as seen for one of the categories in Table I). The survey also included the descriptions listed (as in Table I).

We looked at different ways to analyze the Likert data from the first survey. One method was to look at the NT skills that received the highest percentage of essential ratings. In that vein, the list of the top skills and the percentage of respondents who thought this skill was most essential (rank = 1) can be found in Table II. A second, and very common method considered, often used in analyzing the Likert data, was to simply summarize the Likert values for each NT skill. Based on the summaries, the most essential NT skills, in order, are 1) *teamwork*, 2) *attitude*, 3) *listening*, 4) *initiative/motivation to work*, 5) *critical thinking*, 6) *problem solving*, 7) *attention to detail*, 8) *flexibility*, 9) *integrity/honesty/ethics*, 10) *time management*, 11) *interpersonal relationships*, 12) *oral communications*, 13) *questioning*, 14) *learning*, 15) *leadership*, and 16) *responsibility*. These two lists were very similar; however, after discussing these results with focus group, it was decided to combine the two lists to comprise the second draft of the NT skills that should be considered in the focus group#2.

TABLE II. ESSENTIAL NT SKILLS RATINGS

Non-Technical Skill	% of respondents who rated skill as essential
Teamwork	91%
Initiative/Motivation to work	73%
Listening	73%
Attitude	64%
Critical Thinking	64%
Oral Communications	64%
Leadership	64%
Problem Solving	64%
Attention to Detail	55%
Flexibility	55%
Integrity/Honesty/Ethics	55%
Time Management	55%

The output of focus group # 2 were examples of observable actions that indicated good performance and behavior of each NT skill as well as examples of observable actions that indicate poor performance and behavior of each NT skill. They were asked to provide as many examples as they wished for each skill. The skills under consideration were: *teamwork*, *initiative/motivation to work*, *listening*, *attitude*, *critical thinking*, *oral communication*, *leadership*, *problem solving*, *attention to detail*, *flexibility*, *integrity/honesty/ethics*, *time*

TABLE I. DETAILED NT SKILL OF SW DEVELOPERS: COMMUNICATION

Category	Skill	Synonyms	Description
Communication	Listening	Listen and Understand	Paying attention to and concentrating on what is being said, and asking questions that refine points about which one is uncertain.
	Oral Communications	Communication; Verbal Communication; Communication Skills; Presentation Skills	Presenting your ideas in a manner easily understood by your audience, both in group meetings and person to person. Reinforcing the message to others through gestures and facial expressions.
	Persuasion	Change Agent; Salesman; Influence; Influence and Control; Ability to Influence; Sales; Managing Power/Expectations	Promoting the system you advocate; persuading others to accept your viewpoint.
	Questioning	Interviewing	Asking the right questions in order to obtain the information needed.
	Written Communications		Preparing written documents that accurately communicate ideas in a manner that is easily understood by intended readers.

Listening	
Paying attention to and concentrating on what is being said, repeating points of discussions to ensure mutual understanding and asking questions that refine points about which one is uncertain.	
Examples of Good Behaviors	Examples of Bad Behaviors
Restates, rephrase (paraphrase), or summarize the message to provide feedback if the message was clear and understood. Questions of confirm understanding of the message (e.g. Listening carefully to a teammates presentation, asking questions, and providing constructive feedback in a supportive way).	Passively participating and not paying full attention. (e.g. Looking around the room, checking emails, or other activities on a laptop or phone that show they are not paying full attention to the speaker).
Pauses before restatement or question to show that the message was carefully considered.	Creates distractions while someone else is talking such as whispering to others while someone else is speaking.
Asks clarifying questions. E.g. asks questions to ensure design met the requirements.	Cutting in to the conversation, not letting others complete thoughts. Talking at the same time as someone else.
Lets others talk; does not interrupt.	Combative listening mode - not receptive to other points of view and immediately want to promote their point of view.
Looks at the person speaking. Maintains good eye contact with the speaker.	Exhibits poor body language: crossing arms and legs, stands too close or too far away to the speaker and causes the speaker noticeable discomfort.
Exhibits receptive body language such as leaning forward, nodding or shaking head, etc. (note that nonverbal clues vary from culture to culture. e.g. in some cultures vertically nodding head usually means agreement, in other cultures shaking head means agreement)	

Fig. 2: Example behavioral markers for listening

management, and questioning. A total of 408 examples of good and poor behaviors were collected.

These examples of good and poor behavior provided by the focus group were analyzed and, using an adaptation of the consensual qualitative methodology [30], reviewed and redundant examples were eliminated. The researchers then reviewed the remaining behaviors and evaluated their clarity and how observable they were. Some behavioral examples, such as “being a good team player” and “body language and persona emitting that you do not enjoy your work”, were too ambiguous and removed. It was also felt that the “Leadership” skill did not have enough observable behaviors that would be able to be clearly identified, so that NT skill was removed. The result of the second survey was a behavior-based software engineer NT skills taxonomy. Fig. 2 shows the resultant examples of good and poor behavior for the “Listening” skill. The same process was used to create examples of good and poor behavior for each NT skill, and can be referred in [28].

### C. Creating a Behavior Marker System

The literature review on existing behavioral marker systems showed that there are no BM systems currently being used in the software industry, but did identify existing BM systems in aviation, health care, nuclear power, rail transport and maritime transport. Each system’s structure was examined. The Communication and Teamwork Skills (CATS) Assessment showed the most potential for use in software development because it was devised to measure communication and teamwork providers in a variety of medical environments rather than focusing on a specialization. It also provided an easy to use scoring method [31]. Our results have shown that the NT skills important to good software development team practice include communication, interpersonal, problem solving, and work ethic. Our NT skills taxonomy also includes examples of good and poor behaviors for each skill.

Using the results, we developed a BM marker audit tool (see Fig. 3) that we refer to as the NT Skill Assessment for Software Developers (NTSA). The NTSA is designed to be used by an observer (i.e. manager, team leader, coach) during routine team interactions or meetings. It is intended that each time a behavior is observed, a mark is placed in the appropriate column by clicking on the column: observed and good,

Non-Technical Skills (NTSA) Assessment Instrument, Final Version				
Date: _____		Observer: _____		Segment: _____
Event (place check mark): Stand up Meeting: ___ Customer Demo: ___ Other (describe): _____				
Category	Skill	Good	Poor	Comments
Communication	Listening			
	Oral Communication			
	Questioning			
Interpersonal	Attitude			
	Teamwork			
Problem Solving	Critical Thinking			
	Problem Solving			
Work Ethic	Flexibility			
	Initiative/Motivation to Work			

\* Place one mark in the appropriate column each time behavior is observed.

Fig. 3: NT skills assessment instrument

variation in quality or expected but not observed. Observations can be clarified by placing explanations in the comments section. The observer can see skill definitions and examples of good and poor behavior for a particular behavioral marker by viewing the second page. A manager is allowed to list as many or as few skills as desired in the behavioral marker column. The reason for this flexibility is that different organizations and different managers may wish to focus on a certain subset of NT skills. The observer will score the behaviors base on how well the behavior meets the behavioral examples and its definition. Our NTSA behavioral marker tool will be very usable for practitioners. Empirical validation of our NTSA tool is under progress and our aim is to provide a tool that requires minimal training to use.

## VI. DISCUSSION OF RESULTS

An underlying goal of this research was to develop a useful taxonomy of the NT skills required for software engineers. NT skills are important to the success of projects, but a complete and relevant list approved by both academics and industry has never been developed until now. To accomplish this goal, this research used information found during a literature review and further refined by a focus group of experts in the field to develop NT skills taxonomy. This taxonomy can be used by software developers, educators, and industry to identify the NT skills required by software engineers and software developers that are necessary to have in order to build high-quality

software. Based on evidence gathered from the systematic literature review and the results of two rounds of focus group of experts in the software industry, an NT skills taxonomy was created and validated.

The management of software developers' NT skills is particularly important to today's teams because more and more industries are using agile methodologies which rely less on documentation and more on informal interactions between people. Professional software organizations feel that these skills need to be tracked and feedback provided so that software development team project members can improve. However, there are currently no tools available to assist with this task. To accomplish this goal, we developed a behavioral marker system for software developers based on the NT skills taxonomy. Individuals responsible for measurement and development of the NT skills of their software development teams can use the marker system as a tool across projects to determine areas of strength and areas that need improvement, providing objective feedback to teams and managers. A tool such as the NTSA provides a mechanism to improve a team and by extension the software that they produce.

## VII. CONCLUSION AND FUTURE WORK

This work will benefit researchers, educators, and industry professionals in identifying relevant NT skills to research, and to provide focus on improving the NT skills in software professionals that are so important to software project success. **For researchers**, this work can serve as a starting point for future research into improving the relevant NT skills of software professionals. **For industry**, this work provides a method for managers to measure and manage the NT skills of their software professionals. Industry can use the NT skills taxonomy to identify the NT skills they feel are most relevant to their organizations. The NTSA provides a common language with which to understand and communicate about NT skills important to software professionals. Our future work will include performing studies to refine the NTSA tool and ultimately validate it for the eight NT skills identified as most important to software developers. We expect that future work can further deepen the understanding of which skills are important specifically for software development in contrast to skills that are relevant for teamwork in general. Further validation is also needed in different cultural contexts and development domains. We have begun collaborating with other researchers to use the BM system to investigate industry and student teams in different development environments. Ultimately, the NT skills ratings should be used as independent variables in studies examining the impact of non-technical skill performance on project outcomes.

## REFERENCES

- [1] M.A. West. *Effective teamwork: practical lessons from organizational research*. Malden, MA: Wiley-Blackwell, 2004, pp. 9-14.
- [2] Project Management Institute. *A Guide to the Project Management Body of Knowledge (PMBOK Guide)*. Newton Square, PA: Project Management Institute, 2008, pp. 215.
- [3] S. Acuna, N. Juristo, and A.M. Moreno, "Emphasizing Human Capabilities in Software Development", *IEEE Software*, vol. 23, 2006, pp. 94-101.
- [4] E. Amengual, and A. Mas, "Software Process Improvement through Teamwork Management," in *Proceedings of the 8th International Conference on Product-Focused Software Process Improvement*, 2007, pp. 108-117.
- [5] A. Cockburn, and J. Highsmith, "Agile software development: The people factor", *Computer*, vol. 34, 2001, pp. 131-133.

- [6] N. Gorla, and Y. Wah Lam, "Who Should Work With Whom?" *Communications of the ACM*, vol. 47 No. 6, pp. 79-82, Jun. 2004.
- [7] Ahmed, F. Capretz, L.F., Bouktif, Salah, and Campbell, P., "Soft Skills and Software Development: A Reflection from Software Industry", *International Journal of Information Processing and Management(IJIPM)*, vol. 4, number 3, May 2013, pp. 171-191.
- [8] Abell, Angela, *Information World Review*; Dec 2002; 186; ABI/INFORM Complete pg. 56
- [9] S.G. Cohen, and D.E. Bailey, "What Makes Teams Work: Group Effectiveness Research from the Shop Floor to the Executive Suite", *Journal of Management*, vol. 23, 1997, pp. 239-290.
- [10] J.J. Jiang, J. Motwani, and S.T. Margulis, "IS team projects: IS professionals rate six criteria for assessing effectiveness", *Team Performance Management*, vol. 3, 1997, pp. 236-242.
- [11] O. Hazzan and I. Hadar, "Why and how can human-related measures support software development processes?" *The Journal of Systems and Software* 81, 2008. Pp/ 1248-1252.
- [12] R. Flin, P. O'Connor, and M. Crichton, "Safety at the sharp end: A guide to non-technical skills", 2008, Burlington, VT: Ashgate Publishing Company. Pg. 264
- [13] D. Carnegie, *How to Win Friends and Influence People*. New York, NY: Pocket Books, 1998, pp. xvi.
- [14] Higher Ed News, "Survey Finds Business Executives Aren't Focused on Majors They Hire" accessed Mar. 14, 2014,
- [15] UKSPEC, "UK-SPEC UK Standard for Professional Engineering Competence," accessed Mar. 14, 2014, [www.engc.org.uk/ecukdocuments/internet/documentlibrary/UK-SPEC third edition.pdf](http://www.engc.org.uk/ecukdocuments/internet/documentlibrary/UK-SPEC%20third%20edition.pdf)
- [16] A. Begel and B. Simon, "Novice software developers, all over again," in *Proceedings of the Fourth International Workshop on Computing Education Research (ICER08)*, 2008, pp. 3-14.
- [17] M. Devlin and C. Phillips, "Assessing Competency in Undergraduate Software Engineering Teams", *IEEE Engineering Education*, Universidad Politecnica de Madrid, Apr. 2010, pp. 271-277.
- [18] D.C. Davis, S.W. Beyerlein, and I.T. Davis, "Development and use of an engineer profile", in *Proceedings American Society for Engineering Education Conf., American Society for Engineering Education*, Jun. 2005 .
- [19] B. F. Klampfer, R. L. Helmreich, B. Hausler, B. Sexton, G. Fletcher, P. Field, S. Staender, K. Lauche, P. Dieckmann, and A. Amacher. "Enhancing performance in high risk environments: Recommendations for the use of behavioral markers." *Behavioral Markers Workshop*, 2001, pp. 10.
- [20] P. O'Connor and A. O'Dea, "The U.S. Navy's aviation safety program: a critical review," *International Journal of Applied Aviation Studies*, 2007, vol. 7, p. 312-328.
- [21] B. F. Klampfer, R. L. Helmreich, B. Hausler, B. Sexton, G. Fletcher, P. Field, S. Staender, K. Lauche, P. Dieckmann, and A. Amacher. "Enhancing performance in high risk environments: Recommendations for the use of behavioral markers." *Behavioral Markers Workshop*, 2001, pp. 10.
- [22] J.R. Klinect, P. Murray, A. Merritt, and R. Helmreich. "Line Operations Safety Audit (LOSA): Definition and operating characteristics," in *Proceedings of the 12th International Symposium on Aviation Psychology*, 2003, pp. 663-668.
- [23] G. Fletcher, R. Flin, P. McGeorge, R. Glavin, N. Maran and R. Patey, "Development of a Prototype Behavioural marker System for Anaesthetists' Non-Technical Skills (ANTS)," *Workpackage 5 Report*, Version 1.1. (2003)
- [24] N. Sevdalis, M. Lyons, A.N. Healey, S. Undre, A. Darzi, and C.A. Vincent. "Observational Teamwork Assessment for Surgery: Construct Validation with Expert Versus Novice Raters." *Annals of Surgery*, vol. 249, pp. 1047-1051, 2009.
- [25] G.S. Walia and J.C.Carver, "A systematic literature review to identify and classify software requirement errors". *Information and Software Technology*, vol. 51, pp. 1087-1109, 2009.
- [26] D.C. Davis, S.W. Beyerlein, and I.T. Davis, "Development and use of an engineer profile", in *Proceedings American Society for Engineering Education Conf., American Society for Engineering Education*, Jun. 2005 .
- [27] H. Kandeel, K. Wahbe, *Competency models for human resource development: case of Egyptian software industry. Managing Information Technology in a Global Environment*. 2001 Information Resources Management Association International Conference. Idea Group Publishing. 2001, pp. 117-121
- [28] L.L. Bender and G.S. Walia, "Measurement of Non-Technical Skills of Software Development Teams", *Department of Computer Science*, North Dakota State University, Fargo, ND, Tech. Rep. NDSU-CS-TR-14-001, Mar. 2014.
- [29] G. Fletcher, R. Flin, P. McGeorge, R. Glavin, and R. Patey. "Anaesthetists' Non-Technical Skills (ANTS): evaluation of a behavioural marker system." *British Journal of Anaesthesia*, vol. 90, pp. 580-588, 2003.
- [30] C.E.Hill, S. Know, B.J Thompson, E.N. Williams, S.A. Hess and N. Landany 2005. *Consensual qualitative research: an update. Journal of Counseling Psychology* 52 (2), 196-205.
- [31] F. Robert, A. Abran, and P. Bourque, "A Technical Review of the Software Construction Knowledge Area in the SWEBOK Guide," *STEP* 2002, pp. 9

# Understanding the popularity of reporters and assignees in the Github

Joicy Xavier, Autran Macedo, Marcelo de A. Maia

Computer Science Department

Federal University of Uberlândia

Uberlândia, Minas Gerais, Brasil

joicyxavier@mestrado.ufu.br, autran@fc.ufu.br, marcmaia@facom.ufu.br

**Abstract**—Github has evolved from traditional version control systems to incorporate the wave of the Web 2.0. Intensive collaboration among developers is one of the main goals of Github beyond traditional version control. Understanding how those developers collaborate is a key issue to enhance the outcomes of individuals and of the ecosystem as a whole, as well. Developers activity during the collaboration may be partially registered in the Github database. The analysis of this database can help to answer important questions about different facets of collaboration. In this work, our interest is to understand which factors can influence developers' popularity and provide insights for individuals to enhance their own popularity. We measure popularity with the number of developer followers. We have analyzed a subset of the Github database in order to explain the high popularity phenomenon. Although, we have found that commit activity is an important factor for high popularity, we also have observed developers with low activity (reports and assigns) but with a high number of followers. We present external factors that can explain this dichotomy and they should be considered as key factors in the ecosystem of open-source development.

**Keywords**—Github, popularity, profiling

## I. INTRODUCTION

The data availability in an unprecedented scale imposes a major challenge on our capacity to extract relevant information from data-intensive repositories. In the software development scenario, the situation is similar, for instance, *Ohloh.net* accounts for near 30 billion stored lines of code in February, 2014. The Web 2.0 phenomenon has influenced how software engineers manage projects. Collaboration has been considered a key success factor for the software lifecycle and service providers are increasingly offering more support for software development collaboration. The service providers for software development sharing has been moving from the classical repository supporting version control to more sophisticated services such as, issues trackers and collaborative code review. They also offer some support to extract open data from the repository. In this scenario, mining software repositories plays a key role to shed light on the intricate relationships that manifest in the different artifacts produced during the development process.

A major service provider for project hosting is Github, which hosts not only source code and respective commits, but also the life cycle of issues and the events raised from the collaboration among their users. Github users may be viewed according to their role during the software life cycle. The user

roles which we are interested are based on the activities around the life cycle of an issue. A user is not necessarily a developer that commits to the repository to fix known bugs. Users can also be those that report the issues without necessarily having to fix them, although it is possible that the user can assume both roles. It is important to be aware of the specific role of the user in the software life cycle because different user profiles may induce different collaboration patterns and consequently different popularity profile. It is also important to understand different patterns and profiles in this collaboration process in order to improve our behaviour as a community [5]. Popularity is already an important issue for studies in social networks [2], [4], [9]. However, social networks of developers may have their own specific issues and should be understood on their own.

In this work, we aim at understanding the behaviour of these kind of roles using a partial dataset from selected projects of Github. Our goal was the establishment of users that typically report bugs and of users that typically fix bugs. We assessed variables such as registration time, number of commits, number of participating projects, and number of followers. In order to discover the possible patterns concerning that data, specially which is typical profile of popular users in the Github, we used descriptive statistics and a data mining algorithm [1].

The remaining of the paper is organized as follows: in Section 2, we propose the used methodology. In Section 3, we have shown the results and finally in Section IV, the concluding remarks are presented.

## II. METHODOLOGY

We used a subset of the dataset provided by the GHTorrent group [6]<sup>1</sup>. This dataset contains data from 90 projects, including the top-10 starred software projects for the top programming languages on Github. The dataset accounted for more than 150,000 issues in 13 different programming languages. Our subset considered only issues that were associated to bugs, then the number of issues was drop to 56,959. (From now on, every time we mention issue or dataset, we mean the issue of our subset or our subset, respectively). Both reporters and assignees can be associated to the issues. A reporter is a Github user, which creates an issue (record) relating to a bug information. Whenever a bug is fixed or a bug is allocated to some developer, the respective issue record is related to

<sup>1</sup><http://2014.msconf.org/challenge.php>

TABLE I. RANGES USED IN THE DATA ANALYSIS

Time	Followers	Commits
0-1 years	0-50	0-100
1-2 years	50-100	100-300
2-3 years	100-150	300-500
3-4 years	150-200	500-1000
+4 years	+200	+1000

that user (a.k.a assignee). For each user (reporter or assignee) associated to issues, we computed how long time the user has been registered in the Github (*Time*), the number of followers (*Followers*), the total number of commits in all participating projects (*Commits*), and the number of collaborating projects (*Projects*). These variables were computed in order to understand the user profile concerning the relationship between activity and popularity.

Firstly, this data was analyzed separately for reporters and assignees in order to understand their profiles concerning issues. For this was done a data analysis to identify the numeric ranges of each attribute and were created scales in order to highlight the most significant intervals. The data was classified into different ranges for the variables *Time*, *Followers*, and *Commits*, which are described in Table I. We defined the ranges according to the mean values to identify the skewness in the data more clearly. A more common approach would be defining ranges according quartiles, but in this case the ranges would have the same number of elements, and we were interested in showing which range had possibly lower or higher number of elements.

The data was analyzed using Weka<sup>2</sup> in order to find association rules. We defined the minimum support equals to 0.1 and the minimum confidence greater equals to 0.4. These values were combined to obtain a reasonable number of rules, as we are working with few variables. A larger confidence and support would generate obvious rules, and this would not be interesting for our study. For instance, if we consider support=0.5 we would have only 52 rules, which has little to say about relationships on the followers. Example of rules with support = 0.5 is given below. The rule concerning followers do not provide significant value saying that reporters has 50 to 100 followers with confidence 0.54. Using support = 0.1 and confidence = 0.4, we obtained 326 rules that were more interesting to be analyzed and will be shown in the sequel.

1. Commits=0-100, Projects=0 ==>  
Time=4+ Usertype=reporter      conf:(0.75)
2. Projects=0 ==>  
Time=4+ Commits=0-100      conf:(0.72)
3. Usertype=reporter ==>  
Time=4+ Commits=0-100 Projects=0      conf:(0.64)
4. Usertype=reporter ==>  
Followers=50-100      conf:(0.54)

With support = 0.5 the confidence values were also larger, however, only a few rules that say something about the popularity are found, so it was decided to consider a support = 0.1.

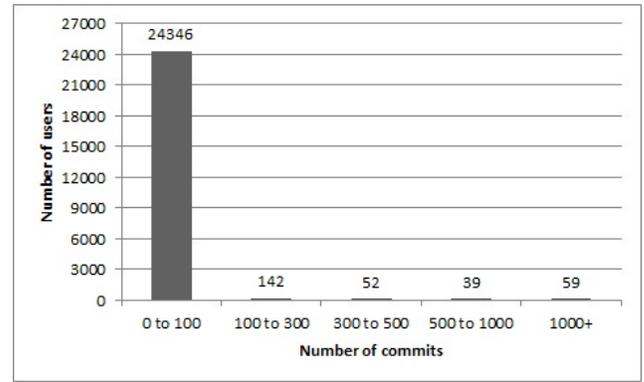


Fig. 1. Number of reporter commits

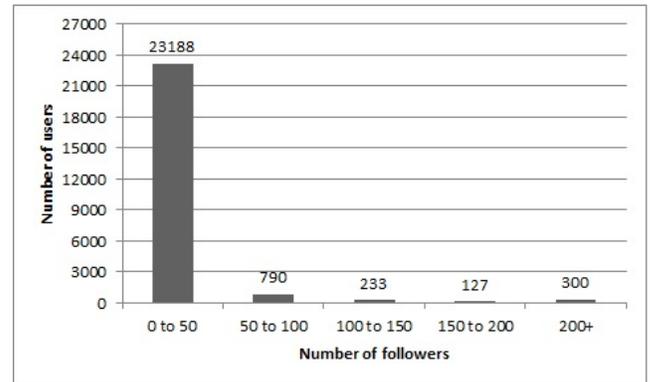


Fig. 2. Number of reporter followers

### III. RESULTS

In this section, we present the data analysis for reporters associated with the filtered bugs and for the users that worked on those bugs (assignees).

#### A. Reporters

Users that create issue records in the Github are called reporters. These users contribute to bug detection and publish them to the community in order to get them solved. The data analysis was driven to establish the typical profile of reporters. We found 24,638 different users that reported the 56,959 bugs, that is, a mean of 2.31 bugs reported per user. For each of these users, we computed the previously described variables *Commits*, *Followers*, *Projects*, and *Time*.

Because reports do not necessarily need to commit, the minimum number of commits was zero and the maximum number was 7,903 commits. As expected, the large majority of the reporters (24,346) have small number of commits (0-100), as shown in Figure 1.

The number of followers for reporters ranged from zero to 12,340. We can observe that most of reporters (94.11%) have small number of followers (0-50). We can observe that there are some reporters that are intensively followed, but only 5.88% have more than 50 followers, as presented in Figure 2.

Github users may collaborate with more than one project. In Figure 3, we have shown the number of projects that each

<sup>2</sup>www.cs.waikato.ac.nz/ml/weka/index.html

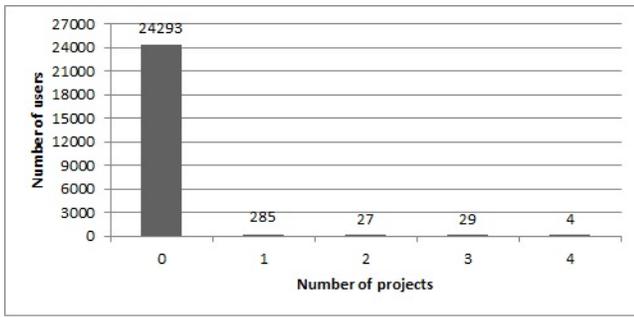


Fig. 3. Number of projects reporters participate

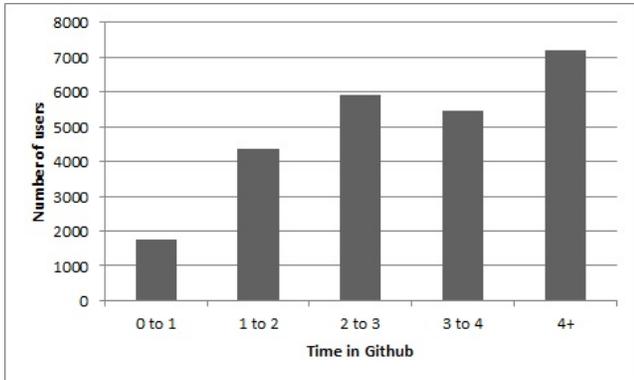


Fig. 4. Registration time for reporters

of the reporters participate. Interestingly, we can observe that 24,293 reporters (98.6%) participate in zero project, once it is possible to report a bug without being attached to the respective project.

The period of time that the reporters are registered in Github varied from 72 days to six years. A significant part of the users (7,181 – ~30%) have been registered in Github for more than four years. Nonetheless, the other ranges are quite uniformly distributed, as can be observed in Figure 4, except for the range from zero to one year which is the least frequent.

### B. Assignees

Assignees are the Github users who fix bugs. Once an assignee fixes a bug the correspondent issue is assigned to him. From the 56,959 bugs, only 4,425 had an associated assignee. For those assigned bugs, 191 different users were involved in their fixing, accounting for a mean of 23 bugs per user. In the same way as we did with reporters, we evaluated the variables *Commits*, *Followers*, *Projects*, and *Time*.

The predominant range for assignee commits was zero to 100. However, the difference was not as high as was for reporters. Moreover, the remaining ranges were quite uniformly distributed, as shown in Figure 5. This was somewhat expected since assignees are responsible for committing the fixes for bugs. One interesting point is that there is a considerable number of very active assignees (those with more than 1000 commits). This result reinforces the fact that the different assignees' profiles are more uniform among themselves concerning commit activity.

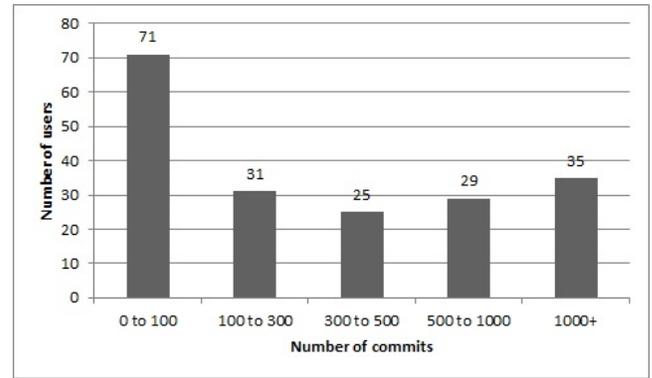


Fig. 5. Number of assignee commits

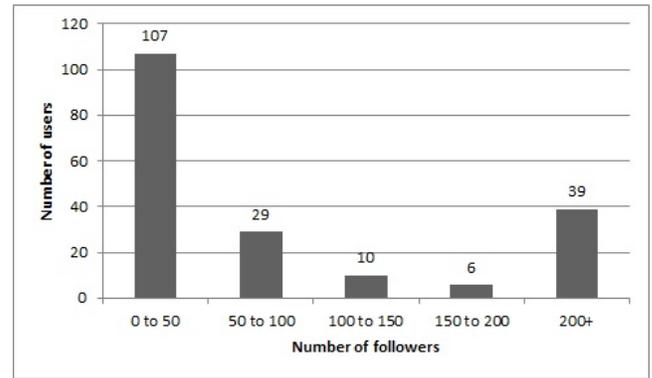


Fig. 6. Number of assignee followers

With respect to the number of followers 56% of users (107) have from zero to 50 followers and 20% (39 users) have more than 200 followers, as shown in Figure 6. The analysis of the number of followers of both reporters and assignees have shown that the 0-50 range is predominant. However, the ranges with higher number of followers have much more higher relative frequency for assignees than for reporters, specially in the range with more than 200 followers. This result shows that although popularity is for very few reporters, this is not exactly the case for assignees. It suggests that assignees tend to be more popular than reporters in Github, which is quite expected because Github is mainly focused on developers.

The group of assignees is distinct from reporters also in the number of projects they participate. While most of reporters do not participate in any project, most of assignees (135 assignees = 70.6%) participate in at least one project, as shown in Figure 7. Only 1% of reporters participates in at least one project.

With respect to the registration time, 51.8% of the assignees are registered in the Github for more than four years and only 2.6% are registered for less than one year, as shown in Figure 8. This could be explained by the sample that considered the top-10 starred projects, which presumably are assumed to have a established team. We can also observe that the larger the registration time, the large is the user contribution in the fixing of bugs.

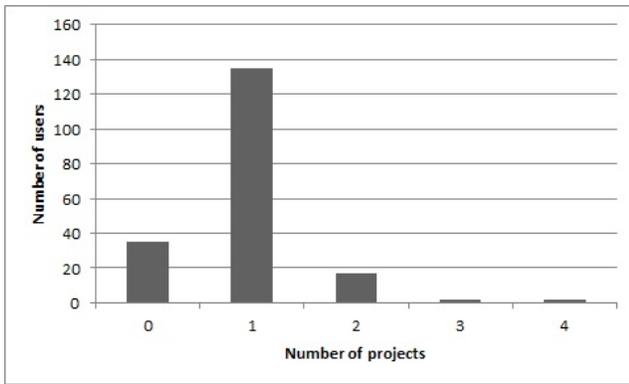


Fig. 7. Number of projects assignees participate

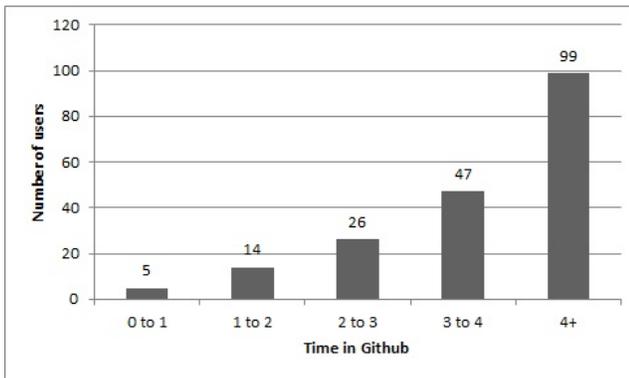


Fig. 8. Registration time for assignees

### C. Mining relationships on user variables

We have opted to use association rules to find possible relationships between the activity of users in the repository in order to unveil possibly non-intuitive relationships. The activities can be measured with the attributes time of register, number of commits, number of participating projects, number of followers and that if the user is reporter or assignee of the projects. Thus, we chose the Apriori algorithm because it is a widely used algorithm to find association rules.

We have used the Apriori algorithm to mine association rules using the variables *Commits*, *Followers*, *Projects*, and *Time*.

With respect to reporters, we found rules, shown below, indicating that reporters are typically users with low activity, independently of the registration time. In other words, the productivity of reporters little popular are not likely to improve over time.

1. Time=1-2 Followers=0-50 Projects=0 ==> Commits=0-100 conf:(1)
2. Time=2-3 Followers=0-50 Projects=0 ==> Commits=0-100 conf:(1)
3. Time=3-4 Followers=0-50 Projects=0 ==> Commits=0-100 conf:(1)
4. Time=4+ Followers=0-50 Projects=0 ==> Commits=0-100 conf:(1)

With respect to assignees, the encountered rules have shown that not only assignees tend to have a larger registration

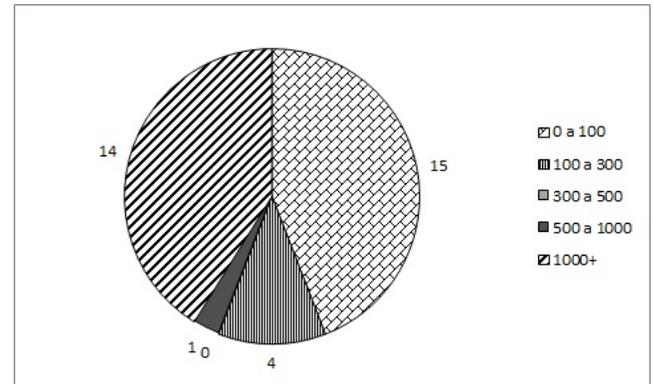


Fig. 9. Number of commits for users with +1000 followers

time, but also they tend to be more active and popular. In this case, the registration time is also related to the number of followers and the number of commits. As we can observe in the following rules, users with registration time greater than four years have higher number of commits and followers, whereas users with registration time less than four years have lower number of followers. We can observe in *rule 6* that users with low number of commits tend to have low number of followers.

1. Commits=1000+ ==> Time=4+ conf:(0.83)
2. Followers=200+ ==> Time=4+ conf:(0.82)
3. Commits=500-1000 ==> Time=4+ conf:(0.66)
4. Time=2-3 ==> Followers=0-50 conf:(0.81)
5. Time=3-4 ==> Followers=0-50 conf:(0.49)
6. Commits=0-a-100 ==> Followers=0-50 conf:(0.76)

So, we could observe that assignees are generally users with higher registration time, have higher commit activity, and so, have higher visibility among their peers. However, there are still other users (including reporters) that do not have high commit activity and still have very high number of followers. So, we decided to manually inspect those users with the highest number of followers to identify other factors that could promote their popularity. The obtained association rules could not provide that information, so we deepened the analysis with descriptive statistics.

We have filtered in the +200 commits range, those users with more than 1000 commits. We found 34 users, from those none were only assignees. Indeed, 20 were only reporters and 14 were both reporter and assignee. Out of these 34 users, 31 have registration time greater than 4 years and the other 3 users have from 3 to 4 years. Curiously, the number of commits has divided this set of users with more than 1000 followers into two predominant subsets shown in Figure 9. In one subset, we can observe users with more than 1000 commits, and the other subset, we have users with less than 100 commits. In other words, although the commit activity can help to achieve the highest levels of popularity, it is not necessarily the major factor because almost half of the highest popular have low

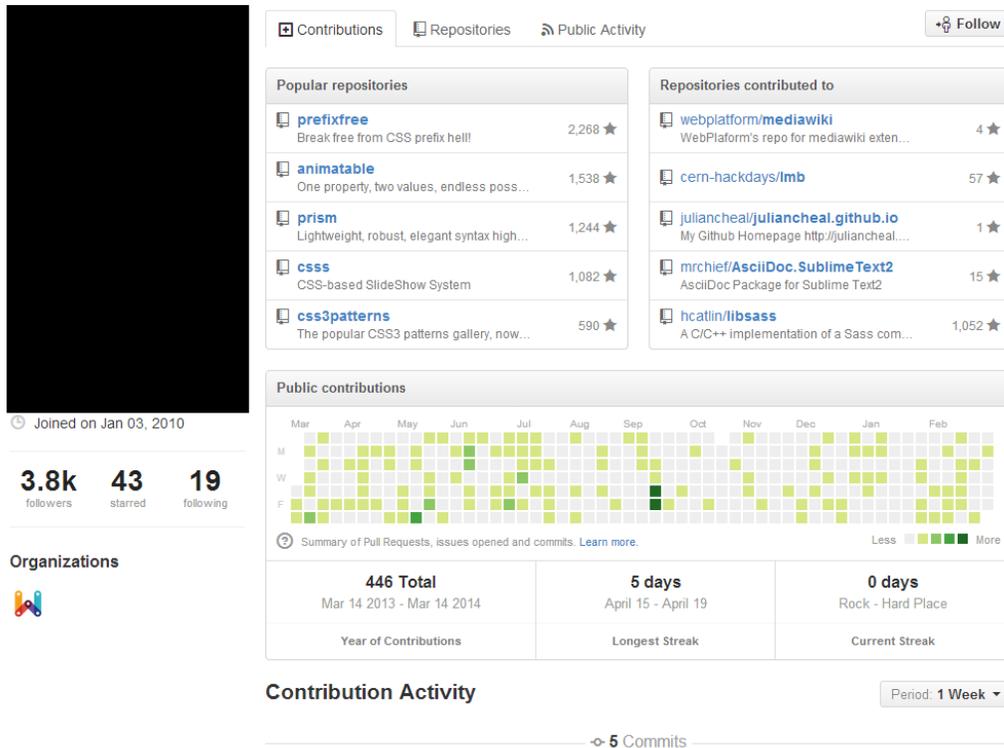


Fig. 10. An example of user with +1000 followers and <100 commits

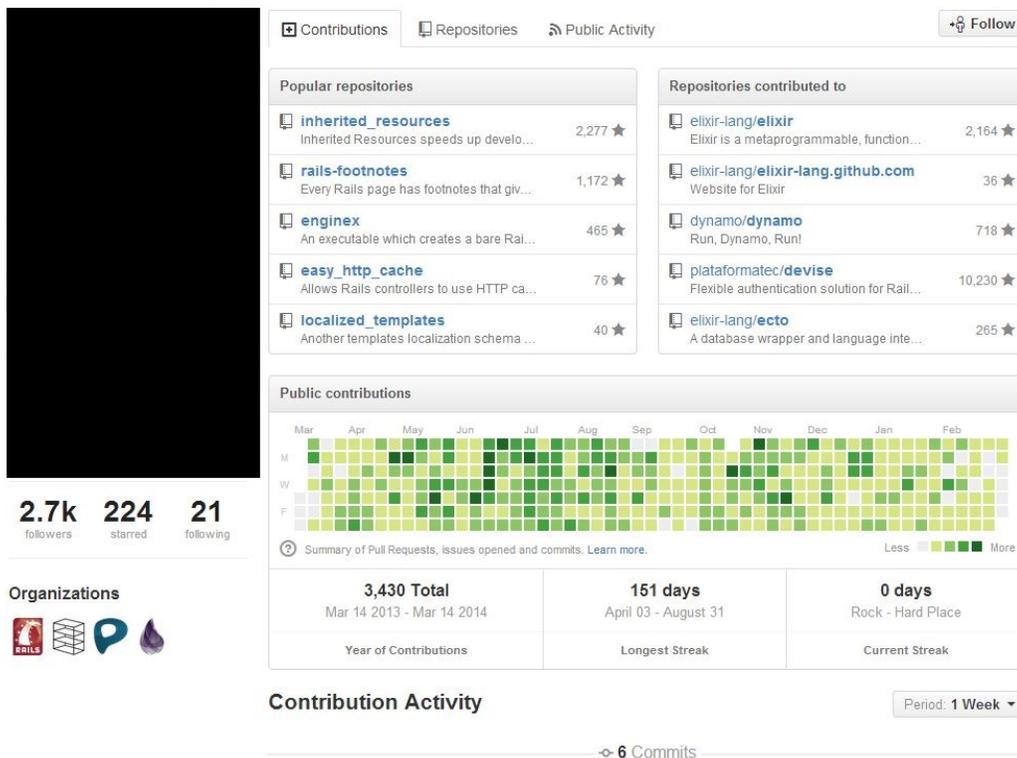


Fig. 11. An example of user with +1000 followers and +1000 commits

commit activity. Indeed, we could find a significant part (~30%) of users with more than 1000 commits that have few followers (0-100).

So, we decided to investigate those top-level popular users that are low profile committers to understand what could explain their popularity. Out of those 15 users, four are owners of important projects: html5 - boilerplate, jquery, homebrew e rails. So, the ownership of important projects could help to explain high popularity. However, we could not find in the dataset information that could help to explain the high popularity of the other 11 users. So, we decided to manually investigate the activity of those users outside the Github. We have observed that those users have distinguished participation in the web with their blogs, their personal social network, and even with publication of printed books. One of these users did not fix any bug, is not a project owner, follows just a few users, and has more than 2000 followers. However, this user writes blog, acts in several social networks and is a book author. Other user, shown in Figure 10 also has low activity, but has more than 3,000 followers, maybe because of the established blog and high activity in social networks. On the other hand, in Figure 11, we can observe a user with more than 1000 commits and 2,700 followers. In this case, the number of commits represents the importance of the developer in the respective project and consequently, enhance the number of followers.

#### IV. RELATED WORK

The concern of being popular is somehow related to reputation scores available in other social-techno sites. For instance, in Stackoverflow, users are scored according downvotes and upvotes on their questions and answers. In [3], they conducted an analysis of how users pursue their reputation, and achieved that new contributors who want to earn high reputation scores quickly may have some rules to follow, such as, answering questions related to tags with lower expertise density, answering questions promptly, being the first one to answer a question, being active during off peak hours, and contributing to diverse areas.

The mechanism for achieving reputation can also be viewed as a kind of gamification. In [8] they conduct a literature review to understand how much does gamification works in a general sense. In the case of Stackoverflow, Grant and Betts [7], has shown that badges can be used to influence user behaviour. They have shown an increase in user activity related to a badge immediately before it is awarded when compared to the period after the grant.

Other study has analyzed the impact of the activity in the reputation [11]. Interestingly, they found many users participating in non-core activities that had greater than one badge and/or high reputation score. This result is similar to ours in the sense that it unveils that there are other factors that impact reputation, as we have shown.

In [10], the authors have shown that *rockstars*, i.e., highly popular users affect the way and the intensity that followers act in their projects, corroborating with our claim that gaining high popularity is very important to the individuals and their respective projects. Moreover, our results suggest that the activity outside the Github is very important to enhance rockstars popularity and thus enhancing the activity on their projects.

#### V. FINAL REMARKS

In this work, we have analyzed the profile of bug reporters and assignees in a Github dataset with the top starred projects of Github. We have used descriptive statistics and association rule mining to support our findings.

We have observed that the number and the profile of reporters and assignees are quite different. Because assignees have more specialized activity their are present in lower number, as expect. On the other hand, reporters are users with relative lower registration time because they can more easily contribute in bug detection.

With respect to the popularity, we could observe that the commit activity may influence on the popularity. However, there are other factors outside the Github activity that influence the popularity. We observe that popularity outside the Github, for example, in blogs or social networks, is likely important to improve the popularity inside the Github and should be considered as an important factor to the project success.

#### VI. ACKNOWLEDGMENTS

This work was partially supported by FAPEMIG grant CEXAPQ-2086-11 and CNPQ grant 475519/2012-4.

#### REFERENCES

- [1] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. In *ACM SIGMOD Record*, volume 22, pages 207–216. ACM, 1993.
- [2] R. Baeza-Yates and D. Saez-Trumper. Online social networks: Beyond popularity. In *Proceedings of the 22Nd International Conference on World Wide Web Companion*, WWW '13 Companion, pages 489–490, Republic and Canton of Geneva, Switzerland, 2013. International World Wide Web Conferences Steering Committee.
- [3] A. Bosu, C. S. Corley, D. Heaton, D. Chatterji, J. C. Carver, and N. A. Kraft. Building reputation in stackoverflow: A empirical investigation. In *Proc. of the MSR'2013* pages 89–92, Piscataway, NJ, USA, 2013. IEEE Press.
- [4] Y. Cha, B. Bi, C.-C. Hsieh, and J. Cho. Incorporating popularity in topic models for social network analysis. In *Proceedings of the 36th International ACM SIGIR Conference*, pages 223–232, New York, NY, USA, 2013. ACM.
- [5] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb. Social coding in github: Transparency and collaboration in an open software repository. In *Proc. of CSCW'12*, pages 1277–1286, New York, NY, USA, 2012. ACM.
- [6] G. Gousios. The GHTorrent dataset and tool suite. In *Proc. of MSR'13*, pages 233–236, 2013.
- [7] S. Grant and B. Betts. Encouraging user behaviour with achievements: an empirical study. In *Proc. of MSR'2013*, pages 65–68. IEEE, 2013.
- [8] J. Hamari, J. Koivisto, and H. Sarsa. Does gamification work? – a literature review of empirical studies on gamification. In *Proceedings of the 47th Hawaii International Conference on System Sciences. HICSS*, 2014.
- [9] G. Kazai and N. Milic-Frayling. Trust, authority and popularity in social information retrieval. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management, CIKM '08*, pages 1503–1504, New York, NY, USA, 2008. ACM.
- [10] M. J. Lee, B. Ferwerda, J. Choi, J. Hahn, J. Y. Moon, and J. Kim. Github developers use rockstars to overcome overflow of news. In *CHI '13 Extended Abstracts on Human Factors in Computing Systems, CHI EA '13*, pages 133–138, New York, NY, USA, 2013. ACM.
- [11] V. S. Sinha, S. Mani, and M. Gupta. Exploring activeness of users in qa forums. In *Proceedings of the Tenth International Workshop on Mining Software Repositories*, pages 77–80. IEEE Press, 2013.

# APT: Approximate Period Detection in Time Series

Rasaq Otunba  
George Mason University  
Fairfax, VA 22030  
rotunba@gmu.edu

Jessica Lin  
George Mason University  
Fairfax, VA 22030  
jessica@gmu.edu

## Abstract

Period detection from time series is an important problem with many real-world applications such as weather forecast, stock market predictions, electrocardiogram analysis, periodic disease outbreak. In this work, we present a novel approximate period detection method for time series. The simplicity of our algorithm and its adaptability for high dimensional datasets using renowned tools and techniques such as locality sensitive hashing (LSH) and MapReduce (using the Hadoop framework for example) make it easier to implement for practical purposes. We performed experiments to compare our technique with a classic period detection technique and two state-of-the-art techniques in terms of accuracy and noise-resilience.

## 1 Introduction

Periodicity is the repetition of a pattern at regular intervals; such regular interval is referred to as a period. We are concerned with the detection of both the period and the periodic pattern in this work. Periodicity can be found in medicine, agriculture, financial market and day-to-day human activities. Examples include a commuter's travel schedule, seasonal sales data, regional sunspot cycle, utility (water) consumption, electrocardiogram. Mining the periodicity in such phenomena can provide useful insights, help make better predictions, help determine structural similarity [17] and detect anomalies among other things. Most of the existing methods for time series periodicity detection assume perfect periodicity and can only detect single periods, which is hardly the case in most natural phenomena. Much of the world's datasets contain mixed (multiple) periods in addition to being noisy and incomplete. This results in periodic patterns that are not identical, and the intervals between them may not be exactly the same lengths. The noisy characteristic of real datasets warrants more robust periodicity mining techniques.

---

\*This research is partially supported by the National Science Foundation under Grant No. 1218325.

A period detection method like the one presented in this work can be regarded as robust if it is able to detect mixed imperfect (approximate) periodicity with relatively high accuracy. In general, three types of periodic patterns can be found in literature: partial periodicity, symbol periodicity, and segment periodicity [14]. We adapt the definitions and describe the following types of periodicity in real-valued time series data. We focus on the discovery of segment periodicity in this work.

- Periodicity is partial if at least one data point in the period, in addition to at least one variable data point is periodic. For instance, in time series  $S = [0, 1, 5, 1, 0, 1, 7, 7, 0, 1, 1, 9, 0, 1, 5, 7]$ , the sequence  $[0, 1]$  is periodic with period  $p = 4$ ; and a partial periodic pattern  $[1, 0, *, *]$  exists in  $S$ , where  $*$  denotes a variable symbol.
- A time series exhibits segment periodicity if the entire pattern is periodic. For instance, the time series  $S = [0, 1, 2, 0, 1, 2, 0, 1, 2]$  has a segment period  $p = 3$ . The periodic segment is  $[0, 1, 2]$ .

Most existing periodicity mining techniques are designed for discrete sequences [13]. In previous work [13], we proposed an approximate period detection algorithm that utilizes SAX (Symbolic Aggregate approxImation) [11], a commonly used time series discretization technique, and grammar induction. While discretization reduces data complexity and simplifies computation, it also results in information loss. In this work, we propose a novel algorithm, APT, that bypasses the discretization step and detects multiple, full-cycle, approximate periodic patterns from the original time series directly. Our distance-based algorithm returns pruned candidate periods and patterns in a time series, and ranks them in the order of significance.

The remainder of the paper is organized as follows. Section 2 discusses related work. We outline preliminaries in Section 3. We describe our approach in Section 4. Experimental evaluation is presented in Section 5. We conclude and discuss future work in Section 6.

## 2 Related Work

Autocorrelation and Fourier Transforms are perhaps two of the most popular periodicity detection techniques. Autocorrelation-based technique is able to detect short and long periods, but has difficulty in identifying the true period due to the fact that the multiples of the true period will have the same power as the true period. On the other hand, Fourier transforms suffer from a number of problems: spectral leakage, which causes a lot of false positives in the periodogram, and poor estimation of long periods due to issues with low frequency regions or sparseness in data [10]. Some methods combine both autocorrelation and Fourier transforms [2, 17].

Some existing algorithms detect only the primary period [6, 8] while others detect all candidate periods [14, 17]. Some methods detect all three types of periodicity listed in the previous section [6, 15, 17, 18] while others detect only a subset. As an example, the methods proposed by Han et al. [7] and Elfeky et al. [4, 5], respectively, detect only segment periodicity. Yang et al. [18] proposed a linear-time distance-based technique to detect periods in a subsection of the time series otherwise referred to as partial periodicity. Techniques proposed in [6, 15] also detect partial periodicity. Most of the aforementioned techniques suffer from noise sensitivity. Rasheed et al. [17] proposed a noise-resilient algorithm to detect periodicity in time series using suffix trees. The time complexity for their proposed method can rise to the order of  $O(n^3)$  and the multiple candidate periods returned are not ranked. WARP [5] was also developed to be noise-resilient but it has large space and time complexity.

## 3 Preliminaries

In this section we define periodicity, approximate periodicity and the problem addressed in this work.

**Definition 1.** Let  $S = [t_0, t_1 \dots t_{n-1}]$  be a time series of length  $n$ , i.e.  $|S| = n$  and  $t_i \in \mathbb{R}$  for  $i \geq 0$  and  $i < n$ .  $S$  is said to be periodic if  $S(t) = S(t + p)$ , where  $t < n - p$ .  $T$  is a periodic subsequence of  $S$  such that,  $T = [t_0, t_1 \dots t_{p-1}]$ , i.e.  $|T| = p$ ,  $p \geq 1$  and  $p \leq \frac{n}{2}$ .  $|T|$  is called the period.

**Definition 2.** Let  $S$  be  $n$ -long sequence of real numbers and  $p$  be a period of  $S$ . We assume the periodic sequence  $T$  to be the first  $p$  points in  $S$ . We generate a periodic sequence  $S_T$  by concatenating  $T$   $\frac{n - (n \bmod p)}{p}$  times and to the first  $(n \bmod p)$  data points of  $T$ .

**Definition 3.** Let  $S$  be  $n$ -long sequence of real numbers. Let  $r$  be a ranking function defined on real numbers.  $T$  is a periodic pattern of  $S$  with  $\epsilon$  error on  $S$  if there exists a subsequence  $T$  such that  $r(S, S_T) = \epsilon$  i.e.  $r$  evaluates the

rank of assuming  $T$  is the correct period of  $S$  relative to other candidate periods. The subsequence  $T$  is called an approximate periodic pattern of  $S$  with error  $\epsilon$ . If there is another periodic pattern  $Q$ ,  $T$  is a more significant periodic sequence if  $r(S, S_T) < r(S, S_Q)$  and vice versa.

**Definition 4.** We use vector notation for all occurrences of a set of periodic subsequences in a sequence of real numbers. Let vectors  $v_1, v_2 \dots v_z$  represent  $z$  such sub patterns. We define the average periodic pattern,  $v_a$  as the mean of the  $z$  vectors i.e.

$$v_a = \frac{v_1 + v_2 + \dots + v_z}{z} \quad (1)$$

We can now define our approximate period detection problem:

**Definition 5.** Given a function  $r$ , and time series  $S$  of length  $n$  over real numbers, compute the approximate periodic patterns  $T$  and corresponding periods  $|T|$  under the function  $r$ .

## 4 Our Approach

We will describe the fundamental premise of our approach and our algorithm in this section. Assume  $T$  is a perfect periodic pattern of a given time series. Let us also assume  $Q$  and  $P$  are approximate periods of  $S$  where  $Q$  is a more significant period. We can generate periodic time series  $S_T, S_Q$  and  $S_P$  respectively according to definition 2. The fundamental premise of our approach is that the distance between  $S$  and  $S_T$  is zero, assuming that  $T$  starts at the beginning of the time series. Analogous to the previous statement,  $Distance(S, S_Q) < Distance(S, S_P)$ . Algorithm 1 shows the pseudocode of our technique and we describe it subsequently in this section.

We iterate over period values from 2 to  $\frac{n}{2}$  in Line 6. Periodic sequences for each period is generated on each iteration, e.g. if  $S = [0, 1, 2, 0, 1, 2, 0, 1]$  and the current iteration has  $i = 2$ , the periodic time series of  $S$  on  $|T| = 2$  is  $[0, 1, 0, 1, 0, 1, 0, 1]$ . The distance between each generated sequence and  $S$  is computed on each iteration. The period values are then ranked according to their distances from  $S$ . The period corresponding to the smallest distance is the most significant. We use Euclidean distance as the distance function in this experiment due to its widely acclaimed success and simplicity. We implement early abandonment [9] to speed up the distance computations. More specifically, the distance function in Line 7 accepts a threshold ( $dist$ ) as a third parameter and terminates the distance computation if the distance computed so far equals to or is greater than the  $dist$  value. Algorithm 2 shows the algorithm listing for the euclideanDist function called in Line 7 of Algorithm 1. The data structure,  $pat$ , initialized in Line 2

---

**Algorithm 1** APT algorithm

---

```
1: procedure APT( $S$ )  $\triangleright S = [t_0, t_1 \dots t_{n-1}]$ 
2:    $patt \leftarrow$  new list of period objects
3:    $dist \leftarrow \infty$ 
4:    $period \leftarrow 2$ 
5:    $cDist \leftarrow 0$ 
6:   for  $i \leftarrow 2, \frac{n}{2}$  do  $\triangleright$  get candidate periods in this loop
7:      $cDist \leftarrow euclideanDist(S, i, dist)$ 
8:     if  $cDist < dist$  then
9:        $dist \leftarrow cDist$ 
10:      if  $i = period + 1$  then  $\triangleright$  prune candidates
11:         $patt.remove(patt.size() - 1)$ 
12:      end if
13:       $period \leftarrow i$ 
14:       $patt.add(newPeriod(dist, period, null))$ 
15:    end if
16:  end for  $\triangleright$  now iterate over the candidate periods
17:  for  $x \leftarrow 1, y$  do  $\triangleright y$  is the # of candidate periods
18:    compute the average periodic pattern for each
19:    candidate according to equation 1
20:  end for
21:  sort  $patt$   $\triangleright$  sort on dist field in ascending order
22:  return  $patt$   $\triangleright$  return ranked candidate periods
23: end procedure
```

---

stores the candidate periods. A Period data structure has 3 fields; the periodic pattern stored in an array, the error of approximation of the Period and the size of the periodic pattern. We prune the candidate periods in the list to remove superfluous candidates in Line 11 based on the condition in Line 10. This pruning is based on keeping only non-consecutive minimum values (points of inflection) of all the errors (distances from the  $r$  function) against the periods. A look at the error-period pairs reveals a progressive decrease in the distance values as it approaches a candidate period, hence, the pruning leaves only the highest period value corresponding to a minimum e.g. if 130, 131 and 132 are candidate periods, only 132 is kept in the list of candidate periods after pruning 130 and 131. The second for-loop beginning in Line 17 computes the average periodic pattern for each candidate period discovered in the previous for-loop. Our algorithm has space complexity in the order of  $O(n)$  and time complexity in the order of  $O(n^2)$  for a time series of size  $n$ . The time and space complexities of our algorithm is still competitive with respect to existing algorithms especially considering the operation of our algorithm on the original time series and not on an approximation like most techniques. The time complexity of APT can also be further reduced to  $O(m * n)$  where  $m$  is the number of periods we are concerned with and  $m < \frac{n}{2}$ . In this case the loop in Line 6 of Algorithm 1 is run  $m$  times.

---

**Algorithm 2** euclideanDist algorithm

---

```
1: procedure EUCLIDEANDIST( $S, i, dist$ )
2:    $newdist \leftarrow 0$ 
3:   for  $j \leftarrow 1, n$  do
4:      $newdist \leftarrow newdist + (t_j - t_{j \bmod i})^2$ 
5:     if  $newdist \geq (dist)^2$  then
6:       return  $dist$ 
7:     end if
8:   end for
9:   return square root of ( $newdist$ )
10: end procedure
```

---

#### 4.1 APT AND IMPLICATIONS FOR HIGH DIMENSIONAL TIME SERIES

As stated earlier, our algorithm is distance based in the Euclidean space. If we store the original time series and the generated periodic sequences for each potential period according to definition 2, we can reduce the period detection problem to the popular approximate nearest neighbor problem (ANN). The original time series will be the query point while all the periodic sequences will constitute the search pool and the periods of the nearest periodic sequences will be candidate periods. A number of methods have been developed for the ANN problem in high dimension. For example, the method in [3] involves the use of the well-acclaimed locality sensitive hashing (LSH) and the method in [1] uses kd-trees, box-decomposition trees, and other search strategies. These methods also support Euclidean distance computations. Implementations of these methods are readily available and they have been widely used with success. This makes APT suitable for efficient period detection in high dimension. This adaptation for period detection in high dimension as described here leads to increased memory requirement to hold all the periodic sequences but reduced time complexity. Much of the speed will be gained by not performing the distance computation on the original high dimension data. Instead, the ANN technique adopted computes the nearest neighbor efficiently in a lower dimension size. It is worth mentioning that there are cases where accuracy is important. In such cases, exact solution is desired. In fact, after performing experiments, Ferhan et al. [16] found that a brute-force approach should not be readily dismissed especially when high recall is desired. Speed could be further improved by implementing our algorithm in a high performance-computing paradigm such as MapReduce. The nearest neighbour search nature of our algorithm makes it suitable for MapReduce as seen in [12]. Space and avoidance of redundancy preclude the inclusion of such details in this work. Hence, we direct the interested reader to consult the references [1, 3, 16] if necessary and we will consider such experiments in the future.

## 5 Experiment

In this section we evaluate APT, a classic technique (Fast Fourier Transforms - FFT) and two state-of-the-art techniques (WARP [5] and MBPD [13]) on synthetic, and real datasets. Experiments were performed on a 2.7GHz, Intel Core i7, MAC OS X version 10.7.5 with 8GB memory. With FFT, the frequency with the highest spectral power from FFT of the dataset is converted into time domain and considered as the most significant period. The period corresponds to the minimum warping cost in WARP as described in [11]. WARP required more memory space than what was available on our machine; hence we down sampled the datasets for WARP where necessary. To achieve this down sampling, we limited the size of all datasets to 1000 points and the periods of synthetic datasets to 100.

### 5.1 Datasets

We used 11 datasets to demonstrate the various factors that can affect the performance of a periodicity detection algorithm. Those factors include the type of periodicity, noise (insertion, deletion or modification), and lengths in our experimental evaluation. We ensured the length of each dataset is a power of 2 to avoid introducing bias by padding the dataset with zeros in order to use FFT for comparison. Seven of the datasets are synthetic (S\_ONE - S\_SEVEN) with 65636 data points. We modified S\_SIX and S\_SEVEN to include mixed periods. The size of the real datasets range from 2048 - 262144 data points. More information about the datasets can be found in [13].

**Table 1:** Ranking error rate on synthetic and real datasets

Datasets	APT	WARP	FFT	MBPD
S_ONE	0.0000	0.0000	0.0000	0.0000
S_TWO	0.0000	0.0000	0.0000	0.0000
S_THREE	0.0000	0.0000	0.0000	0.0000
S_FOUR	0.0001	0.0000	0.0000	0.0000
S_FIVE	0.0000	0.0000	0.0000	0.0000
S_SIX	0.0000	-	0.0000	0.0000
S_SEVEN	0.0000	-	0.0000	0.0000
POWER	0.0012	-	0.0001	0.0000
MFCC	0.0000	-	0.0009	0.0000
SOLAR	0.0010	-	0.0001	0.0000
SUNSPOT	0.0024	-	0.0005	0.0000

### 5.2 Results

We evaluated the performance of our method against FFT, WARP and MBPD with respect to the ranking error rates on both the synthetic and real datasets as shown in Table

**Table 2:** # of false positives on synthetic and real datasets

Datasets	APT	WARP	FFT	MBPD
S_ONE	0.0000	0.0000	0.5000	0.0005
S_TWO	0.0000	0.0000	0.5000	0.0007
S_THREE	0.0001	0.0000	0.5000	0.0056
S_FOUR	0.0001	0.0000	0.5000	0.0050
S_FIVE	0.0000	0.0000	0.5000	0.0011
S_SIX	0.0004	-	0.5000	0.0005
S_SEVEN	0.0008	-	0.5000	0.0003
POWER	0.0004	-	0.5000	0.0049
MFCC	0.0000	-	0.5000	0.0018
SOLAR	0.0001	-	0.5000	0.0023
SUNSPOT	0.0030	-	0.5000	0.0166

**Table 3:** Period error rate on synthetic datasets

Datasets	APT	WARP	FFT	MBPD
S_ONE	0.0000	0.0000	0.0637	0.0000
S_TWO	0.0000	0.0000	0.0637	0.0000
S_THREE	0.0005	0.0000	0.0637	0.0009
S_FOUR	0.0006	0.0000	0.0637	0.0011
S_FIVE	0.0001	-	0.0637	0.0000
S_SIX	0.0000	-	0.0923	0.0000
S_SEVEN	0.0000	-	0.0800	0.0000

**Table 4:** Period values detected real datasets

Datasets	APT	WARP	FFT	MBPD
SOLAR	872	-	910.22	870.60
MFCC	36460	-	37449	36340
POWER	328	577	334.37	328.98
SUNSPOT	133	516	136.53	135.85

1. Ranking error rate in this work is computed in terms of the size of the time series i.e. the number of falsely ranked candidate periods divided by the size of the time series. In Table 2, we record false positive rates. The false positive rate is computed as a ratio of the number of false positives to the size of the dataset.

$$error\ rate = \frac{|expected\ value - actual\ value|}{actual\ value} \quad (2)$$

In Table 3, we show the period error rates of the period value detected on the synthetic datasets by three techniques. The period error rates are computed with equation 2. Since we do not know the exact periods in the real datasets, we did not evaluate the error rate of the period values. Table 4 contains the periods detected on the real datasets. The estimates for the real datasets are SOLAR: 870 - 875; MFCC: 35K - 38K; POWER: 328 - 338; SUNSPOT: 132 - 137. As seen in Table 1, APT has the desired lower false positive

rates compared to FFT and WARP on the real datasets but MBPD performs better in this regard. A low false positive rate indicates the ability to retain only relevant results and saves user the time to look through irrelevant results. From the results in Tables 2, 3 and 4, APT detects the period more accurately than FFT, WARP and MBPD, albeit MBPD performed very competitively. We did not record some of the results for WARP in cases where the results were unreasonable e.g. WARP returned the first value tested (2) on the SOLAR dataset. We also did not record some of the results for WARP when the size of the dataset caused an out of memory exception and down sampling could cause the experiment to lose the validity, as is the case with the real datasets. WARP could not accurately detect the period in some of the real datasets and the synthetic datasets with mixed periods (S\_SIX and S\_SEVEN). FFT, APT and MBPD were able to detect these periods. With all things considered, APT is shown to be more accurate than FFT and WARP. APT performs competitively with MBPD.

FFT has high false positive rate. The false positive rates are approximately half the size of the real time series input. This is because the first  $\frac{n}{2} - 1$  energy components are typically considered since the second half of the FFT (coefficients from  $\frac{n}{2} + 1$  to  $n - 1$ ) can be ignored due to the complex conjugate symmetry with the first  $\frac{n}{2} - 1$  coefficients, for a real time series of size  $n$ . The coefficient at  $\frac{n}{2}$  represents energy at the Nyquist frequency, but this is also ignored by attenuation.

## 6 Conclusion

We present an approximate period detection scheme in this work. From the results of the experiments performed, our technique detects periods more accurately compared to a classic method and two other state-of-the-art methods. Our method detects mixed periods. As future work, we intend to investigate on approaches to increase the efficiency and accuracy of our technique. We would like to perform experiments on high dimensional datasets using efficient high dimensional similarity computation techniques such as locality sensitive hashing (LSH) and high performance techniques such as MapReduce. We would also like to make our algorithm detect partial or subsection periodicity. One limitation of our technique is that the periodic pattern is assumed to start in the beginning of the dataset. We would like to relax this requirement in the future without accruing too much computational burden on the algorithm.

## References

[1] Arya, S., Mount, D., Netanyahu, N., Silverman, R., and Wu, A. (1994). An optimal algorithm for approximate

nearest neighbor searching. *SODA*.

- [2] Berberidis, C., Aref, W., Atallah, M., Vlahavas, I., and Elmagarmid, A. (2002). Multiple and Partial Periodicity Mining in Time Series Databases. *ECAI*.
- [3] Datar, M., Immorlica, N., Indyk, P., and Mirrokni, V. (2004). Locality-Sensitive Hashing Scheme Based on p-Stable Distributions. *SCG*.
- [4] Elfeky, M., Aref, W., and Elmagarmid, A. (2005a). Periodicity detection in time series databases. *TKDE*.
- [5] Elfeky, M., Aref, W., and Elmagarmid, A. (2005b). WARP: time warping for periodicity detection. *ICDM*.
- [6] Han, J., Dong, G., and Yin, Y. (1999). Efficient mining of partial periodic patterns in time series database. *ICDE*.
- [7] Han, J., Gong, W., and Yin, Y. (1998). Mining Segment-Wise Periodic Patterns in Time-Related Databases. *KDD*.
- [8] Indyk, P., Koudas, N., and Muthukrishnan, S. (2000). Identifying Representative Trends in Massive Time Series Data Sets Using Sketches. *VLDB*.
- [9] Lee, J., Kim, S., Kim, B., and Moon, Y. (2011). Maximizing the Early Abandon Effect in Time-Series Similar Sequence Matching. *ICIEIS*.
- [10] Li, Z., Wang, J., and Han, J. (2012). Mining event periodicity from incomplete observations. *KDD*.
- [11] Lin, J., Keogh, E., Lonardi, S., and Chiu, B. (2003). A Symbolic Representation of Time Series, with Implications for Streaming Algorithms. *Workshop on Research Issues in DMKD*.
- [12] Lu, W., Shen, Y., Chen, S., and Ooi, B. (2012). Efficient Processing of k Nearest Neighbor Joins using Mapreduce. *VLDB*.
- [13] Otunba, R., Lin, J., and Senin, P. (2014). MBPD: Motif-Based Period Detection. *BigPMA workshop at PAKDD*.
- [14] Rasheed, F., Al-Shalalfa, M., and Alhaji, R. (2011). Efficient Periodicity Mining in Time Series Databases Using Suffix Trees. *TKDE*.
- [15] Sheng, C., Hsu, W., and Lee, M. (2006). Mining Dense Periodic Patterns in Time Series Data. *ICDE*.
- [16] Ture, F., Elsayed, T., and Lin, J. (2011). No Free Lunch: Brute Force vs. Locality-Sensitive Hashing for Cross-lingual Pairwise Similarity. *SIGIR*.
- [17] Vlachos, M., Yu, P., and Castelli, V. (2005). On Periodicity Detection and Structural Periodic Similarity. *SDM*.
- [18] Yang, J., Wang, W., and Yu, P. (2003). Mining Asynchronous Periodic Patterns in Time Series Data. *TKDE*.

# Exploratory Data Analysis of Software Repositories via GPU Processing

Jose Ricardo da Silva Junior,  
Esteban Clua, Leonardo Murta  
Universidade Federal Fluminense  
Niterói - Brazil  
{jricardo,esteban,leomurta}@ic.uff.br

Anita Sarma  
Computer Science and Engineering  
University of Nebraska, Lincoln  
Lincoln – United States  
asarma@cse.unl.edu

**Abstract**— Analyzing software repositories with thousands of artifacts is data intensive, which makes interactive exploration analysis of such data infeasible. We introduce a novel approach, **Dominoes**, that can support automated exploration of relationships amongst project elements, where users have the flexibility to explore on the fly the numerous types of project relationships. **Dominoes** organizes data extracted from software repositories into multiple matrices that can be treated as domino pieces (e.g., [commit|method]). It allows connecting such pieces based on a set of matrix operations to derive additional domino pieces. These derived domino pieces represent semantics on project entity relationships (e.g., number of commits in which two methods co-occurred) and can be used for further explorations. This opens a vast possibility of data analysis, since these domino pieces can be iteratively combined. Our proposed matrix representation and operations allow for fast and efficient processing of a large volume of data by using a highly parallel architecture, such as GPUs.

**Keywords**- Exploratory data analysis; software dependencies; GPU computing

## I. INTRODUCTION

When working on a software project, developers often need to answer numerous questions, such as: which other methods do I need to edit if I make this change, who was the person who last edited this method, who has expertise in this module, who do I need to coordinate my changes with, and so on [1]. Since software development leaves behind activity logs (i.e., commits recorded in the version control system and tasks recorded in the issue tracking system), it is possible to answer these questions by analyzing these software repositories. However, finding these answers from the repositories is not easy since there is extensive amount of data that is accrued over the project's lifecycle and this data is typically stored across different repositories [2]. This makes creating the right queries a nontrivial task [1].

Several research prototypes attempt to help in project explorations. For example, Tesseract [2] allows interactive investigation of relationships among files, developers, and issues through a network representation. Information Fragments [1] allows a user to compose information from tasks, change sets and teams to explore the relationships between these entities. CodeBook [3] builds a graph of all relationship, and then provides specific applications for

answering specific questions (e.g., finding related developers or artifacts).

There are several critical deficiencies with these current approaches. First, these tools often focus only on a particular development aspect (e.g., EEL [4] helps in expert identification). Second, these tools typically allow explorations of specific relationships that are fixed a priori (e.g., Tesseract preprocesses the sets of dyadic relationships first). Third, these tools often need a complete history of the project (e.g., in order to traverse the relationship graph, Codebook requires the full history). Fourth, all these tools operate at coarse granularity (file level). Identifying change impact information at a fine-grained level can highlight nuanced relationships. For instance, although two developers have edited the same file, it is possible that these developers have complementary expertise in the functionality (method) of the file. Finally, these tools need to restrict the data that can be analyzed because performing interactive data analytics of software archives through visual explorations of relationships among project elements is infeasible at the scale of operation that is needed.

In this paper, we present a novel approach – **Dominoes** – designed to enable interactive exploratory analysis of relationships of different software entities at varying levels of granularity by utilizing matrix operations that can be computed via GPUs. Our approach organizes data from a software repository into multiple matrices that are treated as domino tiles, such as [developer|commit], [commit|method], [class|method], amongst many other combinations. Just as in the **Dominoes** game, where joining two congruent squares edge to edge can form a rectangle, our matrices can be combined to create additional (derived) matrices. This derivation process is guided by a set of matrix operations, such as addition, multiplication, and transposition. For example, a computation of logical coupling at the method level can be achieved as  $[\text{method}|\text{method}] = [\text{commit}|\text{method}]^T \times [\text{commit}|\text{method}]$ . With this new (derived) domino tile, we can derive dependency among developers as  $[\text{developer}|\text{developer}] = [\text{developer}|\text{commit}] \times [\text{commit}|\text{method}] \times [\text{method}|\text{method}] \times [\text{commit}|\text{method}]^T \times [\text{developer}|\text{commit}]^T$ . Many such different explorations are possible, with each derived domino tile representing a particular aspect in software engineering.

A primary goal of **Dominoes** is to enable users to explore the relationships in their project elements across different levels of granularity. Therefore, the granularity aspect is a central

construct, with one of the domino tile types being that of “composition” (e.g. [package|class], [file|class], and [class|method]). Connecting any other domino tile with these composition tiles or their transpose allows navigation from coarse-grained to fine-grained analysis or vice versa.

Explorations of such relationships at a fine-grained level (methods across different versions of the software) while more accurate, can lead to extremely large data sets to be analyzed. Dominoes implements the exploratory analysis of software project entities as linear algebra operations over matrices, which can be parallelized in GPU (Graphics Processing Unit) [5]. This allows boosts in performance of about three orders of magnitude [7]. Therefore, Dominoes opens a new realm of exploratory software analysis, as endless combinations of domino pieces can be experimented with in an exploratory fashion.

## II. DOMINOES APPROACH

In this section we describe our overall approach and then focus on the matrices and how we operate over them. Dominoes extracts data from a software repository and converts them into multiple matrices, correlating the desired attributes in lines and columns. This strategy allows data manipulation and its operations using parallel architecture. In our case, this fact allows interactive manipulations even with large datasets, as we are using GPU to perform matrix operations.

We represent a matrix as  $M$  and its transpose by using a superscript ( $M^T$ ). Individual elements in a matrix are denoted as  $M[i,j]$ . The operator “ $\times$ ” represents matrix multiplication. It is important to note that when multiplying two matrices the number of columns in the first operand must be equal to the number of rows in the second operand. In our case the column and rows of the operand over which we are multiplying also needs to be the congruent (same project element), similar to the Dominoes game. In other words, we can multiply [developer|commit]  $\times$  [commit|method], but not [developer|commit]  $\times$  [method|method].

### A. Dominoes Tiles

Dominoes includes *basic building tiles*, which can be combined to create *derived building tiles*; which can be further combined with other basic or derived tiles. The *basic building tiles* are created by extracting data from existing software repositories (version control systems, issue tracking systems, etc.). For example, commits, issues, discussions about a commit, or pull request can be collected from GitHub. The *basic building tiles* around commits include:

- [class|method] (CIM): relationship between a class and its constituent methods, where cell  $CIM[i,j]$  has a value of 1 when class  $i$  contains method  $j$ .
- [commit|method] (CM): relationship between commits and methods, where cell  $CM[i,j]$  has a value of 1 when commit  $i$  adds or changes method  $j$ . Note that the index  $i$  does not necessarily express the commit id.

- [developer|commit] (DC): relationship between developers and their commits, where cell  $DC[i,j]$  has a value of 1 when developer  $i$  is the author of commit  $j$ .
- [bug|commit] (BC): relationship between commits and bugs, where cell  $BC[i,j]$  has a value of 1 when commit  $j$  fixed bug  $i$ .

These *basic building tiles* can then be combined to form a series of *derived building tiles*. Here we show a small set of *derived building tiles* that can be computed using only multiplication and transposition operations:

- [method|method] ( $MM = CM^T \times CM$ ): represents method dependencies, where  $MM[i,j]$  denotes the strength of the dependency of method  $j$  on method  $i$ . The rationale of this matrix is based on logical dependencies, as elements that are co-committed share some program logic. Note that we can also create an MM matrix through program analysis, in which case it would be termed as a *basic building tile*. Such MM matrices have been explored by Steward in creating Design Structure Matrices [8]. In Section III-B we explore more elaborate ways for computing MM.
- [class|class] ( $CICI = CIM \times MM \times CIM^T$ ): represents class dependencies, where  $CICI[i,j]$  denotes the strength of the dependency of class  $j$  on class  $i$ . Note that using the composition tile, we can provide analysis results at a higher-level of abstraction easily.
- [bug|method] ( $BM = BC \times CM$ ): represents the methods that were changed to fix each bug. This matrix could be used to identify which methods are “buggy”.
- [developer|method] ( $DM = DC \times CM$ ): represents the methods that a developer has changed. This matrix could be used to identify experts on a particular method as well as if there is breadth in expertise for a given method.
- [developer|class] ( $DCI = DM \times CIM^T$ ): represents classes that a developer has changed. DCI uses the composition operation to provide expertise information at the class level, which is typically used during bug triaging [3].
- [developer|developer] ( $DD = DM \times MM^T \times DM^T$ ): represents the expertise dependency among developers, where developer  $j$  depends on some knowledge of developer  $i$ , because of underlying technical dependencies in their work. It is worth no notice that this *derived building tile* used other *derived building tile* (MM and DM) in its definition.

By applying the composition operation, the above software engineering constructs can switch to class or file grains.

### B. Specialized Operations

Our basic matrices are typically binary, that is,  $M[i,j]$  is either 1 or 0, whereas our derived matrices are not. This is largely because commits are atomic transactions and therefore most associated matrices with commits are binary. In the case of derived matrices cell values have associated semantics. Simple operations such as multiplication and transposition allow us to compose different types of domino tiles to derive more complex matrices and, thereby, different software

engineering constructs. However, there are three “specialized” operations that can be applied on derived matrices where individual cells are not binary.

Let us take the example of the MM matrix. The diagonal shows how frequently a method has been changed and each cell ( $M[i,j]$ ) shows how frequently a method ( $i$ ) has changed with another method ( $j$ ). This semantics is equivalent to *absolute support*, largely adopted in the data mining community. The support of an item set is defined as the proportion of transactions in the dataset that contains the item set. According to [9], the rule  $X \rightarrow Y$  has support  $s$  if  $s\%$  of transactions contain  $X \cup Y$ . As this operation pattern of multiplying a matrix by its transpose is very popular and semantic rich, we treat it as a specialized operation and is computed according to Eq. 1.

$$M^{\text{sup}} = M \times M^T \quad (1)$$

The semantics of support allows us to answer software engineering related questions regarding the strength of the relationships. For example, if we are interested in predicting the other files a developer needs to edit because of a change, we can use the concept of logical dependencies to identify all those methods that are dependent of the edited method and may also need to be changed. We could use the  $MM = MC^{\text{sup}}$  matrix to answer this question.

Unfortunately, support is transitive, where  $M^{\text{sup}}[i,j] = M^{\text{sup}}[j,i]$ . Consequently, using support to represent dependencies is not precise, as program dependency is not transitive. For example, in our scenario, the area of a Cylinder depends on the circumference of the Circle, but not vice versa.

In order to obtain a more precise relationship that reflects the direction of the dependency, Zimmermann et al. [10] use *confidence* to represent logical coupling. This metric suggests which artifacts should be modified together, given that a specific artifact is being modified. According to [11], the rule  $X \rightarrow Y$  has confidence  $c$  if  $c\%$  of transactions that contain  $X$  also contain  $Y$  [11]. In the context of our approach, when applied to compute MM matrix, confidence quantifies the occurrence of an entity (e.g., method) change given that the other entity (e.g., method) has also been changed. The confidence operator is computed according to Eq. 2.

$$M^{\text{conf}}[i,j] = \frac{M^{\text{sup}}[i,j]}{M^{\text{sup}}[i,i]} \quad (2)$$

Confidence does not have a transitive property among elements, so it is possible to define different levels of dependency for each pair. However, confidence suffers from another type of problem. In the context of data mining, confidence is used to quantify relations such as “*those who buy product X also buy product Y*”. In this case, if product “ $Y$ ” is presented in almost all orders, purchase of any product will lead to a high confidence in buying “ $Y$ ”. For this reason, analyzing confidence alone tends to be imprecise, and can exhibit false relationships.

To address this problem we can use a third metric – *lift* [9]. Lift measures the influence of the antecedent in the frequency of the consequent. Formally, the rule  $X \rightarrow Y$  has lift  $l$  if the frequency of  $Y$  increases in  $l$  times when  $X$  occurs. According

to this definition, we are interested in dependencies with lift greater than 1, as any other value implies irrelevant (coincidental) relationships. The lift operator is defined by Eq. 3, where the scalar multiplication by the number of commits ( $M^{\text{rows}}$ ) transforms the absolute support ( $M^{\text{sup}}$ ) into relative support (values ranging from 0 to 1).

$$M^{\text{lift}}[i,j] = \frac{M^{\text{conf}}[i,j] \times M^{\text{rows}}}{M^{\text{sup}}[j,j]} \quad (3)$$

### III. DISCUSSION

Here we describe our approach by drawing on a scenario detailed in Section III-A. We apply our approach in identifying artifact dependencies on Apache Derby<sup>1</sup>, an open source relational database project, as explained in Section III-B. Finally, in Section III-C we discuss the Dominoes architecture.

#### A. Scenario Evaluation

Consider a scenario where three developers (*Alice*, *Bob*, and *Carlos*) work together on a “geometry project”, consisting of four classes (Circle, Cylinder, Cone, and Shape). Circle has a method *circumf()* that calculates its circumference. Shape has a method *draw()* to render a shape. Finally both Cylinder and Cone have methods *area()* to calculate the area of the respective shapes. Table I describes five commits and their change descriptions. Table II shows which commits modified which method. Note that this is an intentionally simple example to explain the concepts in the paper.

Figure 1 represents the support, confidence, and lift values for the MM matrix, where  $C_i$  represents *Circle.circumference()*,  $C_y$  – *Cylinder.area()*,  $C_o$  – *Cone.area()*, and  $S$  – *Shape.draw()*.

TABLE I. COMMITS MADE BY DEVELOPERS

Commit #	Developer	Description
C <sub>1</sub>	Alice	Change type of function parameter to compute the radius (Circle) and how to render it (in Shape)
C <sub>2</sub>	Carlos	Change the side of Cone and how to render it
C <sub>3</sub>	Alice	Change how a Shape is rendered
C <sub>4</sub>	Alice	Calculation of how circumference and area are calculated using PI. Required modification on how to draw a Shape
C <sub>5</sub>	Bob	Modify the height calculation of a cylinder and how it is rendered

TABLE II. METHODS CHANGED FOR COMMIT

Commit #	Circle circumf()	Cylinder area()	Cone area()	Shape draw()
C <sub>1</sub>	1	1	0	1
C <sub>2</sub>	0	0	1	1
C <sub>3</sub>	0	0	0	1
C <sub>4</sub>	1	1	1	1
C <sub>5</sub>	0	1	0	1

<sup>1</sup> Derby Repository: <https://github.com/apache/derby>

	Ci	Cy	Co	S		Ci	Cy	Co	S		Ci	Cy	Co	S
Ci	2	2	1	2	Ci	1	1	0.5	1	Ci	2.5	1.6	1.2	1
Cy	2	3	1	3	Cy	0.6	1	0.3	1	Cy	1.6	1.6	0.7	1
Co	1	1	2	2	Co	0.5	0.5	1	1	Co	1.2	0.8	2.5	1
S	2	3	2	5	S	0.4	0.6	0.4	1	S	1	1	1	1
	Support					Confidence					Lift			

Figure 1. Support, Confidence, and Lift calculated from previous scenario.

If we consider the confidence matrix, we notice that the dependency from  $Cy$  to  $Ci$  (100% conf.) is stronger than from  $Ci$  to  $Cy$  (60% conf.), because whenever  $Ci$  was changed it also required changes to  $Cy$  (commits  $C_1$  and  $C_4$ ) (see Table II). However,  $Cy$  was changed once without  $Ci$  (commit  $C_5$ ). With such a confidence analysis we can state that  $Cy$  (always) depends on  $Ci$ , but  $Ci$  does not necessarily depend on  $Cy$ . Therefore, using confidence to derive the DD matrix would identify that Bob should communicate with Alice, but not necessary the other way around.

The confidence matrix also indicates high dependency from  $S$  to all other methods. However, this occurs not because  $S$  really depends on all other methods, but because  $S$  was changed in all commits, independently (see Table II). The lift matrix eliminates such coincidental dependencies, keeping only dependencies between  $Cy$  and  $Ci$ , and  $Co$  and  $Ci$ , since all other values are either equal to or below 1.

In summary, support alone is not sufficient to indicate dependencies among project entities, but helps in eliminating dependencies that appear by chance (e.g.,  $Co$  and  $Ci$ ). In a large project with thousands of commits thresholding on a predefined support level can help eliminate accidental dependencies. On the other hand, lift plays a complementary role of identifying dependencies to elements that are very frequent (e.g.,  $S$ ) and therefore may be a cause of coincidental changes. Finally, confidence is important to identify the direction of the dependency (e.g., from  $Cy$  to  $Ci$ ). With such an analysis, we find that the only real dependency in our scenario is from  $Cy$  to  $Ci$ , which would lead to a communication requirement from Bob to Alice in the DD matrix.

Our approach, therefore, provides four distinct advantages. First, the confidence measure allows more nuanced investigations (e.g., direction of dependency). Second, the use of lift measure increases the accuracy of the finding by filtering out common, but unrelated changes. Third, the fine-grained analysis from the method level increases accuracy, since we can identify dependencies among individual methods. Therefore, if we find that  $Cy$  depends on  $Ci$ , we can find that Bob needs to coordinate with Alice who is working on  $Ci$  and not another developer who is working on the same file (Circle, but on a different method). Finally, GPU processing allows these investigations to be performed interactively.

### B. Derby Analysis

In order to evaluate Dominoes in a real setting, we applied it over Apache Derby, an open source project. This evaluation aims to demonstrate how solely working with support is error prone for finding artifact dependencies. All analyses were made considering the repository data from 08/11/2004 to 01/23/2014, which comprises **7,578** commits, **36** distinct

developers, **34,335** files, and **305,551** methods committed during approximately 10 years. This evaluation was performed at the file-level for easier interpretation of the results. However, as discussed before, Dominoes can easily navigate from coarse to fine grained analysis and vice-versa.

After processing the data, we found that due to the project characteristics of Derby, the lift analysis does not filter out any coincidental dependencies. This is because the Derby file dependencies are highly clustered, causing low support and a very high lift. When we filter the lift values by thresholding on 1, no data points were eliminated. Therefore, we continued with the support/confidence analysis.

Table III presents the top 5 logical dependencies in terms of support and with the biggest difference in confidence, considering a support threshold above 30. It is important to remember that confidence is not transitive.

Considering the first case as an example, it is possible to observe that using the common approach that is based on support, artifacts `DataDictionary.java` and `DataDictionaryImpl.java` would be considered as dependent to each other as they have a high support (in fact, 79 is the highest value of absolute support in the whole system). However, when observing the confidence, it is possible to see that only `DataDictionaryImpl.java` has dependency with `DataDictionary.java`, which is reasonable as changing a method implementation normally does not result a change to its interface. The following two rows are also interface/implementation cases, presenting the same behavior. In the fourth row, we have a composition case, where `DRDAConnThread.java` possesses a `DRDAStatement.java` instance. In this case, modifying the latter does not necessarily imply a modifications in the former. However, there is a high likelihood of a related change in the opposite direction, that is, modifications in `DRDAStatement.java` can change method signatures used by `DRDAConnThread.java`, for instance.

Finally, the last case is a class specialization, thus normally requiring modification to both files, with a slightly higher dependence from the subclass to the superclass. These analyses show the importance of using confidence to identify the direction of the dependencies.

TABLE III. TOP 5 LOGICAL DEPENDENCIES IN TERMS OF HIGH SUPPORT AND BIGGEST CONFIDENCE DIFFERENCE

Artifact A	Artifact B	Support	Conf. (A-B)	Conf. (B-A)
<code>DataDictionary.java</code>	<code>DataDictionaryImpl.java</code>	79	0.88	0.37
<code>DD_Version.java</code>	<code>DataDictionaryImpl.java</code>	45	0.78	0.21
<code>LanguageConnectionContext.java</code>	<code>GenericLanguageConnectionContext.java</code>	44	0.86	0.48
<code>DRDAConnThread.java</code>	<code>DRDAStatement.java</code>	37	0.22	0.68
<code>ResultSetNode.java</code>	<code>SelectNode.java</code>	36	0.54	0.45

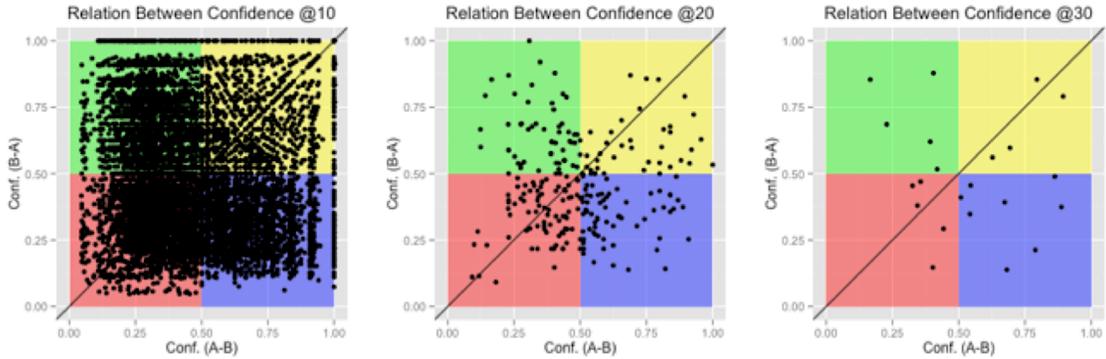


Figure 2. Relation among confidence for various support threshold. The leftmost chart considers a threshold of 10, while the middle uses 20, and finally the rightmost uses 30.

Besides these five top dependencies, Figure 2 presents a scatter plot chart with all dependencies at three specific support levels (10, 20, and 30). This chart plots each dependency according to its confidence in both directions (A-B and B-A). This way, dependencies with the same confidence value in both directions are plotted along the diagonal. As we can see, however, there are several cases where points are located far from the diagonal. When we consider the rightmost chart in Figure 2 (support threshold at 30) for discussion we can observe some distinct patterns. The red quadrant shows that both  $\text{conf}(\text{A-B})$  and  $\text{conf}(\text{B-A})$  are less than 0.5, thus containing weak bidirectional dependencies. The yellow quadrant, on the other hand, shows dependencies where both  $\text{conf}(\text{A-B})$  and  $\text{conf}(\text{B-A})$  are above 0.5, thus containing strong bidirectional dependencies. Finally, the green and blue quadrants show unidirectional dependencies with highest divergences among confidence. In this case, dependencies from these quadrants can be erroneously classified as bidirectional if we are to analyze dependencies solely by support.

Performing an analysis such as the one in Figure 2 can unveil how inaccurate dependencies extracted from support-based approaches tend to be. As demonstrated for the Derby project, only the yellow-quadrant dependencies should be classified as bidirectional. Both blue and green quadrants present unidirectional dependencies.

In this evaluation, the FC (i.e., [file|commit]) matrix was of size  $34,335 \times 7,578$ . The generation of  $\text{FC}^{\text{sup}}$  and  $\text{FC}^{\text{conf}}$  using GPU (NVIDIA GeForce GTX580) took about 0.7 minutes. However, doing the same computation using CPU (Intel Core 2 Quad Q6600) would take 696 minutes. This shows that we got a speedup of three orders of magnitude when GPU is in place – with just the simple calculation that requires 1 transposition and 3 multiplication operations. Similarly, when we process MC (i.e., [method|commit]), which is  $305,551 \times 7,578$ , it takes about 5 minutes in GPU, being impossible to be processed on CPU in a reasonable amount of time.

### C. Dominoes Architecture

Dominoes Architecture is designed in a way that data from a software project repository is extracted and the associated change information is archived. Currently, we are mining projects that use Git by cloning and accessing the local repo. The local repo is then preprocessed to generate a tree of all

modifications performed in all commits by analyzing which files, packages, classes, and methods were modified. It is important to note that information of each modification is decomposed to get a fine-grained view of the changes by using the Eclipse ASTParser (suitable for Java-based projects). For example, even if we represent changes at the package level (for a coarse-grained analysis), we know exactly which class was modified, as well the methods. This information is then stored in a relational database. Furthermore, after the initial data collection, information about subsequent changes can be updated incrementally to the database.

Basic Dominoes tiles are then constructed on the fly and become available to be used. Depending on the type of operation required by the user, these domino tiles are sent to the GPU, which performs the desired operations to generate the derived domino tiles. These derived domino tiles can also be saved as a template piece, should that piece be used extensively in calculations.

## IV. RELATED WORK

Our related work can be divided in two main groups: approaches that determine dependencies amongst artifacts or developers and approaches that provide support for exploratory analysis. There are numerous approaches that focus on identifying structural dependencies (through syntactic analysis) or logical dependencies (through change history) amongst artifacts. Cataldo et al. [12] stands out as they use matrices to process dependencies among developers based on dependencies among artifacts. In their approach, both structural dependencies and logical dependencies become Task Dependency ( $T_D$ ) matrices, and change requests, associating developers to artifacts, becomes Task Assignment ( $T_A$ ) matrix. These matrices are used in an equation that indicates coordination requirements  $T_A \times T_D \times T_A^T$ . Our approach generalizes this idea by allowing different kinds of exploration over matrices. Finally, our identification of relationships is innovative, as it allows combining support, confidence, and lift, to compose the dependency matrix depending on the research need.

Tools that enable exploratory analysis provide either predefined questions or are very limited to derive information that was not conceived beforehand. In the case of Tesseract [2], for example, the available relationships are preprocessed and

the matrices are fixed at a coarse grain (file-file, file-developer, file-bug, bug-developer). CodeBook [3] is a similar approach that builds a graph of all relationship and then provides applications for answering specific questions (e.g., identifying related developers or artifacts). Gall et al. [13] built a tool for mining software archives at a fine grain in order to compare source code changes. From these analyses, recommendations such as change type patterns and consistency of changes can be made. Instead of a recommendation system, Dominoes provides a generic and flexible platform for exploratory analysis of project elements at a fine-grain level, which is compatible with multiple data types and relationships. Its interactive capabilities are mainly possible due to the adoption of GPU. It is important to state we have already used GPU for solving software engineering problems. In a previous work [6] we achieved boosts of two orders of magnitude when running image diff, patch, and merge operations in GPU.

## V. CONCLUSION

Dominoes is an exploratory data analysis approach that allows users to select information about different project elements and their interrelationships from a repository. Relationships are represented by matrices, defined as *basic building tiles* and *derived building tiles*. Both kinds of building tiles can be combined iteratively to reveal deeper, complex relationships. Through such explorations, relationships that have not been computed or published before can be discovered through the operations over these building tiles. As all operations are performed in parallel over GPU, exploratory analysis can occur seamlessly at real time, even when computing relationships in fine-grained data. The current version of Dominoes tool extracts data from a Git repository and operates over the matrices by using GPU kernels in CUDA.

Our evaluation contrasted the use of support alone and the use of support and confidence to distinguish the dependence directions. In the Derby case, employing confidence leads to a more accurate analysis for finding dependencies among artifacts. Moreover, using confidence for thresholding a relationship is more natural for the user, as it represents a normalized value.

The Dominoes architecture was intentionally designed to easily accommodate the definition of new *basic building tiles*, such as relationships mined from communication channels (e.g., email, chat, discussion forums). The same extensibility feature also applies for operations. Besides the basic matrix operations, such as multiplication and transpose, specialized operations can also be plugged into Dominoes, as showed in section III-B for support, confidence, and lift. This leads to a relevant approach for the scientific community, as empirical studies can be reproduced over different corpora in order to validate an investigation. This has the potential of alleviating the pain of setting up an environment for each trial of an investigation.

Although we currently use matrices and GPU underneath Dominoes, other data representations and execution environments could be adopted in the future. For example, relational algebra is a compelling alternative to link sparse data. Moreover, SMP is the *de facto* architecture of modern personal

computers. However, some kinds of analysis, such as reachability (used in impact analysis), can heavily benefit by operating over matrices in GPU. Besides that, due to the characteristic of our data, methods to deal with sparse matrix in order to reduce the memory usage can be adopted.

A concept not discussed in this paper, which is currently under development, is the use of three-dimensional (3D) building blocks. These 3D building blocks consider time as the third dimension over the matrices. In our experience, using this additional dimension collected through a specific time window allows observing the evolution of relationships over time. Another ongoing work is real time visualizations of basic and derived building tiles both in two and three dimensions to support explorations by end users on their own data.

## ACKNOWLEDGMENT

This work is partially supported by CNPq, CAPES, and FAPERJ and NSF through awards: HCC- 1110916 and CCF-1253786.

## REFERENCES

- [1] T. Fritz and G. C. Murphy, "Using Information Fragments to Answer the Questions Developers Ask," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, New York, NY, USA, 2010, pp. 175–184.
- [2] A. Sarma, L. Maccherone, P. Wagstrom, and J. Herbsleb, "Tesseract: Interactive visual exploration of socio-technical relationships in software development," in *Proceedings of the 31st International Conference on Software Engineering*, Washington, DC, USA, 2009, pp. 23–33.
- [3] A. Begel, Y. P. Khoo, and T. Zimmermann, "Codebook: Discovering and Exploiting Relationships in Software Repositories," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, New York, NY, USA, 2010, pp. 125–134.
- [4] S. Minto and G. C. Murphy, "Recommending Emergent Teams," in *Fourth International Workshop on Mining Software Repositories, 2007. ICSE Workshops MSR '07, 2007*, pp. 5–5.
- [5] J. Krüger and R. Westermann, "Linear algebra operators for GPU implementation of numerical algorithms," in *ACM SIGGRAPH 2003 Papers*, New York, NY, USA, 2003, pp. 908–916.
- [6] J. R. da Silva, T. Pacheco, E. Clua, and L. Murta, "A GPU-based Architecture for Parallel Image-aware Version Control," in *2011 15th European Conference on Software Maintenance and Reengineering*, Los Alamitos, CA, USA, 2012, vol. 0, pp. 191–200.
- [7] S. Rajasekaran, L. Fiondella, M. Ahmed, and R. A. Ammar, *Multicore Computing: Algorithms, Architectures, and Applications*. New York, NY: CRC Press, 2013.
- [8] D. V. Steward, "The design structure system: A method for managing the design of complex systems," *IEEE Trans. Eng. Manag.*, vol. EM-28, no. 3, pp. 71–74, 1981.
- [9] S. Tuffery, *Data mining and statistics for decision making*. Chichester, West Sussex; Hoboken, NJ.: Wiley, 2011.
- [10] T. Zimmermann, A. Zeller, P. Weissgerber, and S. Diehl, "Mining version histories to guide software changes," *Softw. Eng. IEEE Trans. On*, vol. 31, no. 6, pp. 429–445, 2005.
- [11] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules in Large Databases," in *Proceedings of the 20th International Conference on Very Large Data Bases*, San Francisco, CA, USA, 1994, pp. 487–499.
- [12] M. Cataldo, J. D. Herbsleb, and K. M. Carley, "Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity," in *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, New York, NY, USA, 2008, pp. 2–11.
- [13] H. C. Gall, B. Fluri, and M. Pinzger, "Change Analysis with Evolizer and ChangeDistiller," *IEEE Softw.*, vol. 26, no. 1, pp. 26–33, Jan. 2009.

# Artificial neural networks for infectious diarrhea prediction using meteorological factors in Shanghai

Yongming Wang

Department of Computer Science &  
Technology  
East China Normal University  
Shanghai, China  
ymwang819@gmail.com

Junzhong Gu

Department of Computer Science &  
Technology  
East China Normal University  
Shanghai, China  
jzgu@ica.stc.sh.cn

Zili Zhou

Department of Computer Science &  
Technology  
East China Normal University  
Shanghai, China  
zlzhou@ica.stc.sh.cn

**Abstract**—Infectious diarrhea is an important public health problem around the world. Meteorological factors have been strongly linked to the incidence of infectious diarrhea. Therefore, accurately forecast the number of infectious diarrhea under the effect of meteorological factors is critical to control efforts. In this paper, a three layered feed-forward back-propagation artificial neural network model (FFBPNN) are developed to predict the weekly number of infectious diarrhea by using meteorological factors as input variable. The meteorological factors were chosen based on the strongly relativity with infectious diarrhea. Also, as a comparison study, the multivariate linear regression (MLR) also was applied as prediction model using the same dataset. Further, since one of the drawbacks of FFBPNN model is the interpretation of the final model in terms of the importance of variables, a sensitivity analysis is performed to determine the parametric influence on the model outputs. The simulation results obtained from the neural network confirms the feasibility of this model in terms of applicability and shows better agreement with the actual data, compared to those from the regression models. The FFBPNN model, described in this paper, is an efficient quantitative tool to evaluate and predict the infectious diarrhea using meteorological factors.

**Keywords**—artificial neural networks; forecasting model; infectious diarrhea; multivariate linear regression; sensitivity analysis; meteorological factors

## I. INTRODUCTION

As a kind of common and important infectious disease, infectious diarrhea has a serious threat to human health and leads to one billion disease episodes and 1.8 million deaths each year (WHO, 2008). Infectious diarrhea in young children is a killer illness, especially in developing countries [1, 2]. In Shanghai of China which is the biggest developing country, the incidence of infectious diarrhea has significant seasonality throughout the year and is particularly high in the summer and autumn of recent years. Hence, a robust short-term (week-ahead) forecasting model for infectious diarrhea incidence is necessary for decision-making in policy and public health.

Infectious diseases have a closely relation with meteorological factors [3] and can affect infectious diseases in a linear or nonlinear fashion [4]. In recent years, there has been a large scientific and public debate on climate change and its

direct as well as indirect effects on human health [5]. The effects of meteorological factors, such as temperature and rainfall, on diarrhea diseases incidence have got much more concerning recently. As far as we are concerned with the prediction of diarrhea diseases in literature, many forecasting models based on statistical methods for diarrhea diseases forecasting have been reported. Zhao N et al. [6] establish multiple regression model rolling forecast of daily incidence of infectious diarrhea in Beijing. Chou WC et al. [7] applied a climate variation-guided Poisson regression model to predict the dynamics of diarrhea-associated morbidity. The results indicated that the maximum temperature and extreme rainfall days were strongly related to diarrhea-associated morbidity. Lloyd SJ et al. [8] undertook a global cross-sectional study of diarrhea incidence in children under 5, and assessed the association with climate variables (temperature and rainfall) by linear regression method.

With regard to the fact that number of meteorological factor that effect infectious diarrhea are too much and the inter-relation among them is also very complicated, prediction models based on statistics methods may not be fully suitable for such type of problems. Nowadays, artificial neural networks (ANNs) are considered to be one of the intelligent tools to understand the complex problems and have been widely used in the medical and health field [9, 10, 11]. To the best knowledge of the authors, there is no works has been carried out to utilize the ANNs method in predicting diarrhea disease. In this paper, an attempt has been made to establish a new ANNs model (FFBPNN) to predict infectious diarrhea in Shanghai with a set of meteorological factors as predictors. Also, as a comparison, a multivariate linear regression (MLR) model was developed for the same purpose using the same dataset. Finally, since one of the drawbacks of ANNs is the interpretation of the final model in terms of the importance of input variables, the calculation is performed using sensitivity analysis.

The rest of this paper is organized as follow. Study area and dataset are briefly described in Sect. II. The prediction method and performance evaluation criterias (PECs) which are used in this paper are introduced in Sect. III. The FFBPNN and MLR models are developed and training results are reported in Sect. IV and Sect. V, respectively. In order to investigate the

performance of the established model, the prediction results of the FFBPNN model are reported in comparison with the MLR model as discussed in Sect. VI. Sect. VII illustrated the sensitivity analysis, and conclusions of this paper are concluded in Sect. VIII.

## II. STUDY AREA AND DATASET

### A. Study area

Shanghai is located in the eastern part of China, and the city has a mild subtropical climate with four distinct seasons and abundant rainfalls. It is the most populous city in China comprising urban/suburban districts and counties, with a total area of 6,340.5 square kilometers. Since the study population was relatively stationary during the time period from 2005 to 2008 with the annual growth rate below 1%, the trend of incidence during that time period could be similarly prescribed by the trend of disease cases number. Hence we used the number of infectious diarrhea instead of incidence as the response variable in our models.

### B. Dataset

The daily data of the infectious diarrhea cases for the period 2005.1.3-2009.1.4 were collected from National Disease Supervision Information Management System. The cases were all clinical or laboratory-confirmed cases and reported by hospital diagnostic. The daily meteorological factors data (including temperature, relative humidity, rainfall, atmospheric pressure, wind speed, and sunshine duration) in Shanghai for same time period were selected and collected from the Shanghai Meteorological Bureau of City Environmental Meteorological Center.

A total of 209 weeks experimental data pairs (infections diarrhea cases vs. nine meteorological factors) were obtained by daily records for constructing the FFBPNN and MLR models. The statistical descriptions of the input and output parameters show that the range of data can cover the various values of input and output parameters and all the dataset are bearing almost similar statistical properties. Description of the input (meteorological factors) and output (the weekly number of infectious diarrhea) parameters for constructing the FFBPNN and MLR models have been given in Table I.

TABLE I. DESCRIPTION OF INPUT AND OUTPUT PARAMETERS

Input/Output	Weekly Parameter (Unit)	Symbol
Meteorological factors	Maximum temperature (°C)	$T_{max}$
	Minimum temperature (°C)	$T_{min}$
	Average temperature (°C)	$T_{avg}$
	Minimum relative humidity (%)	$RH_{min}$
	Average relative humidity (%)	$RH_{avg}$
	Average atmospheric pressure (hPa)	$AP_{avg}$
	Sunshine duration (h)	$SD$
	Average wind speed (m/h)	$WS_{avg}$
	Rainfall (mm)	$R$
Infectious diarrhea	Number of infectious diarrhea	$WNID$

## III. THE PREDICTION METHOD AND PERFORMANCE METRICS

In this section, the modeling approach of the FFBPNN for forecasting is briefly reviewed. Then, the performance evaluation criterias of the prediction model are presented.

### A. Feed-forward back-propagation neural network

ANNs which consist of a large number of simple and highly interconnected computing components, called nodes or neurons are a branch of artificial intelligence methods. ANNs are organized into layers, called input layer, hidden layers, and output layer. These layers further include interconnections between the nodes of successive layers through the weights. The internal weights of the network are adjusted by an iterative process termed training and the algorithm used for this purpose called training algorithm. ANNs have a lot of important capabilities including learning from data, generalization and working with unlimited number of variable. ANNs, with their remarkable ability to derive meaning from complicated or imprecise data, are usually used to model complex relationships between inputs and outputs or to extract patterns in dataset.

The multilayer perception, trained by the standard back-propagation (BP) learning algorithm, is generally known as the FFBPNN. The BP learning algorithm is composed by the forward spread of the data stream and the reverse spread of the error signal. Among the most of different ANNs architectures, the FFBPNN is one of the most commonly and widely used for the purpose of prediction problems [12]. Apart from an input layer receiving inputs from the environment and an output layer generating the network's response, one or more intermediate hidden layers also exist. The typical topology structure of single hidden layer FFBPNN (Fig. 1) is the most widely used model form for forecasting.

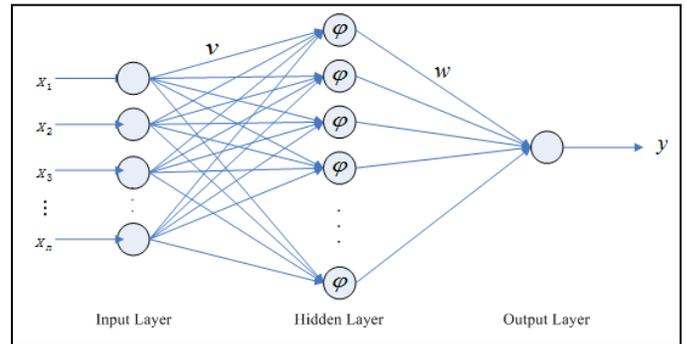


Figure 1. The typical topology structure of FFBPNN

The relationship between the output ( $y$ ) and the inputs ( $x_1, x_2, \dots, x_n$ ) has the following mathematical representation:

$$y = f(x_i) = w_0 + \sum_{j=1}^m w_j \varphi \left( \sum_{i=1}^n (v_{ij} x_i - b_j) \right) - b \quad (1)$$

Where  $m$  denotes the number of hidden nodes,  $w_j$  denotes the weight between the  $j$ -th hidden node and the output node,  $v_{ij}$  denotes the weight between the  $i$ -th input node,  $x_i$ , and the  $j$ -th hidden node,  $b_j$  and  $b$ , denote the biases for the  $j$ -th hidden

nodes and the output node respectively,  $\varphi$  is the transfer function of the hidden set in which the sigmoid function is used.

$$x_{ij} = 0.9 \times \frac{X_{ij} - \min(X_i)}{\max(X_i) - \min(X_i)} + 0.05 \quad (2)$$

### B. Evaluation criteria for model performance

To date, there is no single performance measure has been recognized as the universal standard. As a result, we need to assess the performance based on multiple metrics, and it is interesting to see if different metrics will give the same performance ranking for the models to be tested [21]. Among of PECs, the mean absolute error (MAE), the root mean square error (RMSE), the mean absolute percentage error (MAPE), the correlation coefficient (R) and the coefficient of determination ( $R^2$ ) are the most widely used performance evaluation criterias and will be used in this study. The models with the smallest RMSE, MAE and MAPE and the largest R and  $R^2$  are considered to be the best models.

## IV. DEVELOPMENT OF FFBPNN MODEL

The FFBPNN modeling consists of two steps: the first step is to train the network using training dataset which are used for adjusting the model parameters; the second step is to test the network with testing dataset, which are not used in training step and are used to estimate the generalization error of trained networks. A model is considered good if the error of out-of-sample testing is the lowest compared with the other models. FFBPNN models were trained and tested by means of the Neural Network Toolbox of the MATLAB R2012b.

In this study, the dataset were chosen and segregated in time order. In other words, the dataset of the earlier period were used for training and the dataset of the latest time period were used for testing. From the collected dataset, the dataset form 2005 to 2007 were selected for the development of the FFBPNN models, the remaining dataset (2008) were used to verify the generalization capability of the established FFBPNN model. RMSE value for the testing period was considered for performance evaluation and all testing stage estimates were plotted in the form of hydrograph

### A. Model input and output parameters

Input parameters selection is an important task before the neural network modeling, whether to choose a set of input variables which can best reflect the reason for desired output changes is directly related to the performance of neural network prediction. In this study, the input parameters affecting the infections diarrhea are weekly values of nine meteorological factors. The output parameter is the weekly number of infections diarrhea.

### B. Data pre-processing and post-processing

It was noticed that the range of input and output parameters were different. In order to improve the efficiency and generalization of neural network models, the dataset should firstly be normalized. Because of the use of sigmoid functions in the FFBPNN model, the dataset are normalized between 0.05 and 0.95 rather than between 0 and 1 to avoid saturation of the sigmoid function leading to slow or no learning. The normalized values for each row of input and output parameters were calculated using (2):

Where  $x_{ij}$  is the normalized input/output value,  $X_{ij}$  is the actual input/output value,  $\max(X_i)$  is the maximum input/output value,  $\min(X_i)$  is the minimum input/output value. After the modeling and prediction, a reverse of this procedure is performed, transforming the output data back to the original scale, then these data can be applied to the estimate of the model performance, the comparison between the predicted and the actual values, and the appreciation of the accuracy.

### C. Determination of optimum network and parameters

In the development of FFBPNN model, determination of an appropriate architecture for a particular problem is an important issue as the network topology directly affects its computational complexity and its generalization capability [13]. There is no unified approach for determination of an optimal FFBPNN architecture [14]. The number of input-output layer nodes is determined according the modeling problem being tackled, and in this study the input layer has nine neurons and one neuron in the output layer.

For the number of hidden layer, Hecht-Neilsen [15] indicated that a single-hidden layer FFBPNN is sufficient to approximate the corresponding desired outputs arbitrarily close. Therefore, one hidden layer was preferred in this study. However, the number of neurons is the most critical task in the FFBPNN structure. In this study, the number of neurons in the hidden layer has to be decided based on a series of trial-and-error experiments and it was determined by gradually increasing the number of neurons and observing their effect on the predicted value.

As demonstrated in Fig. 2, various network architectures with respect to hidden node numbers were trained and tested in order to obtain the best model architecture. The network with 9-4-1 was found to be the optimum model architecture for the *WNID* prediction (See Fig. 3) because of its minimum network error (RMSE) values for the training and testing dataset (For each number of hidden neurons, network was trained for 20 times to overcome the randomness of choosing initial weights and biases of neurons).

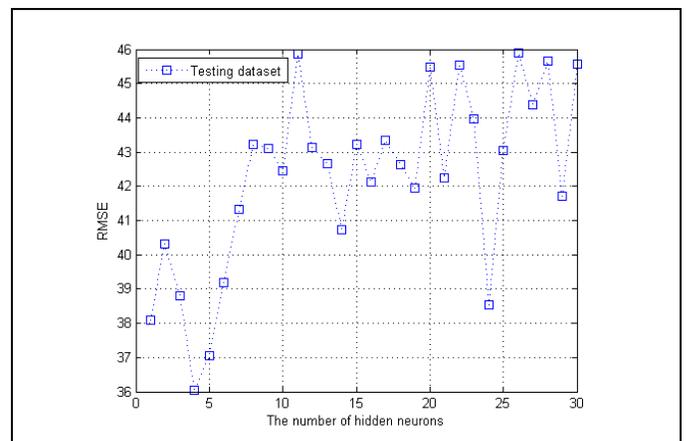


Figure 2. Hidden neurons and network errors for FFBPNN models

A back-propagation learning algorithm was utilized in a FFBPNN trained using the Levenberg-Marquardt (LM) algorithm [16]. An S-shaped nonlinear hyperbolic tangent sigmoid activation function  $f(x)=(1-e^{-x})/(1+e^{-x})$  was used in the hidden layer and a linear activation function (Purelin) was used in the output layer, respectively. In order to develop the FFBPNN model, a very large number of trials with varying number of other network control parameters like learning rate and momentum rate were performed using training dataset through multiple iterations. The optimum architecture of the network and effective number of parameters used in developed FFBPNN model are shown in Table II.

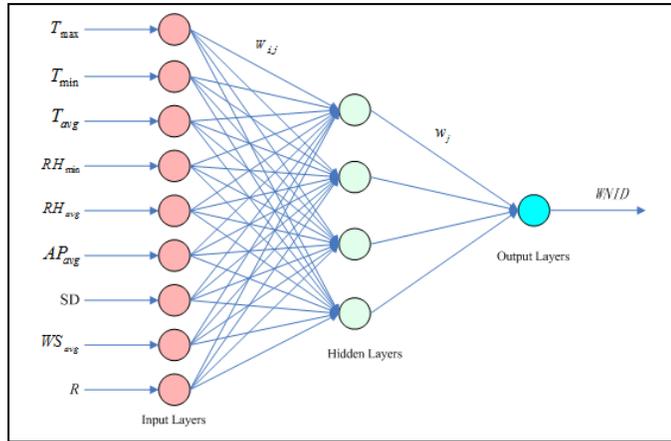


Figure 3. FFBPNN architecture for infections diarrhea predicting

TABLE II. THE NETWORK ARCHITECTURE AND PARAMETERS USED IN OPTIMUM NETWORK ARCHITECTURE

Parameters	FFBPNN
Number of input layer units	9
Number of hidden layer	1
Number of hidden layer units	4
Number of output layer units	1
Momentum rate	0.9
Learning rate	0.74
Error after learning	1e-6
Learning cycle	1500 epoch
Transfer function in hidden layer	Tansig
Transfer function in output layer	Purelin
Training function	TRAINGDM

### V. DEVELOPMENT OF MLR MODEL

In order to compare the results of FFBPNN model with an empirical model, we also developed MLR model for *WNID* forecasting using the same dataset. The *WNID* values were selected as the dependent variable, and the meteorological factors variables were selected as independent variables. MLR is carried out to find out the relationship of several independent variables with a dependent variable. The software Matlab version R2012b was used to calculate the values of these coefficients. MLR was carried out on training dataset to determine the mathematical expressions for the *WNID*. The MLR model by using multiple linear regressions for prediction

of the *WNID* is derived as presented by the following mathematical equation:

$$\begin{aligned}
 WNID = & -1972.7903 - 10.9619T_{max} \\
 & + 20.8158T_{min} - 2.6208T_{avg} - 1.6506RH_{min} \\
 & + 0.2993RH_{avg} + 2.0902AP_{avg} + 5.7734SD \\
 & - 15.7205WS_{avg} + 1.6048R
 \end{aligned} \quad (3)$$

### VI. RESULTS AND DISCUSSION

In this section, the results by using the FFBPNN and MLR models for predicting *WNID* are presented. The optimal structure for ANN model often results in different networks, dependent on the initial random values of the synaptic weights. Therefore, the outcome will, in general, not be the same in two different trials even if the same training data have been used. In this article, we have only presented the best result obtained after 20 trials with the same input-output dataset for different models. The model outputs were transformed back to the original range, and then five key PECs were calculated and compared between actual and predicted values for training and testing dataset.

The prediction performance statistical values of building FFBPNN and MLR models for both the training and testing dataset have been depicted in Table III. By comparing the results, it is clear that the models considered in this study show acceptable prediction accuracy and higher coefficient of determination values with  $R^2$  ( $R^2 > 0.90$  for FFBPNN model and  $R^2 > 0.80$  for MLR model). The MAPE of prediction were always lower than 0.5 in testing dataset for all the models. It is also clear that when the performance measurements are compared for MLR and FFBPNN models, the performance of FFBPNN models in terms of those PECs consistently outperforms MLR model. Therefore, the predicting performance of the FFBPNN model is better than that of the MLR model.

TABLE III. THE RESULTS OF FFBPNN AND MLR MODELS FOR TRAINING AND TESTING DATASET

PECs	Models			
	FFBPNN		MLR	
	Training	Testing	Training	Testing
MAE	20.7628	27.7547	29.8077	35.3774
RMSE	28.3007	36.0526	39.3739	48.9395
MAPE	0.2727	0.3841	0.4337	0.4182
R	0.8783	0.8490	0.8089	0.6968
R <sup>2</sup>	0.9213	0.9125	0.8811	0.8388

Also, comparison between predicted and actual values for the FFBPNN and MLR models by using the training and testing are graphically illustrated in the Figs. 4 and Figs. 5, respectively. The linear least square fit line, its equation, and the  $R^2$  values were shown in these regression figures Fig. 4 (b and d) and Fig. 5 (b and d) for the training and testing data. It

can be obviously seen from the Fig. 4 (a and c) as well as Fig. 5 (a and c) that the FFBPNN-yielded *WNID* predicted values are in closer agreement with the corresponding actual values for training and testing samples than those for MLR model, especially peak points. The underestimation of the peak values and overestimation of the low values are much more for the MLR than the FFBPNN models. The MLR model seems to be insufficient for the forecasting *WNID*, especially the peak values.

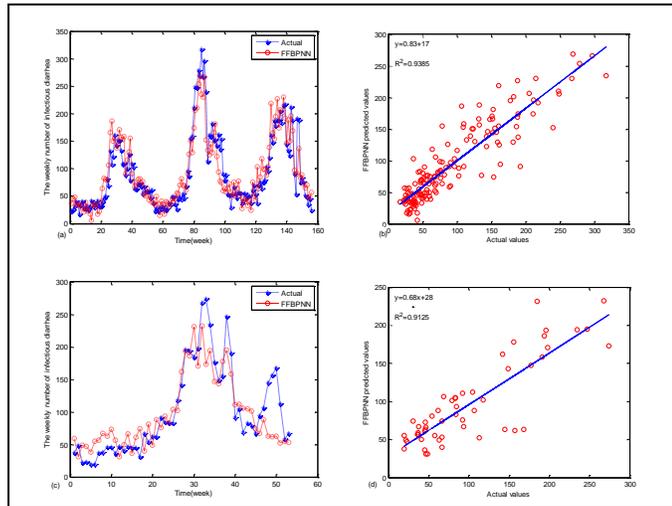


Figure 4. Comparison between actual and FFBPNN predicted result in training and testing dataset. (a) Comparison curves plot of actual vs. predicted trends for training dataset, (b) Scatter plot of actual vs. predicted values for training dataset, (c) Comparison plot of actual vs. predicted trends for testing dataset, (d) Scatter plot of actual vs. predicted values for testing dataset

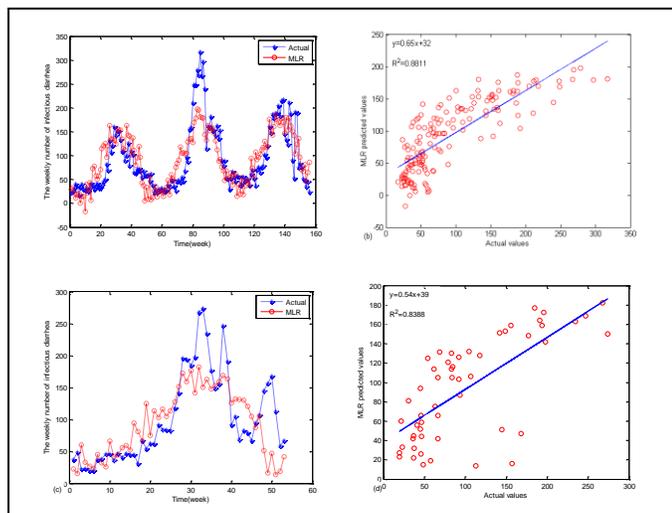


Figure 5. Comparison between actual and MLR predicted result in training and testing dataset. (a) Comparison curves plot of actual vs. predicted trends for training dataset, (b) Scatter plot of actual vs. predicted values for training dataset, (c) Comparison plot of actual vs. predicted trends for testing dataset, (d) Scatter plot of actual vs. predicted values for testing dataset.

The reason of better performances of the FFBPNN model over MLR model may be attributed to the complex nonlinear relationship between infectious diseases and meteorological

factors. This implies the nonlinearity of the investigated phenomenon. The MLR method was chosen only as a representative statistical model in this study. It is based on the polynomial fitting approach. In many other studies [17, 18] multiple linear regression method was also selected as a benchmark model to compare and also found to be unacceptable against the FFBPNN in different application fields for forecasting.

For the FFBPNN models, it is very difficult to know which network structure and network parameters will be the accurate for a given problem (*WNID*). It will depend on many parameters, including the complexity of the problem, hidden neurons, the number of data points in the training set, the number of weights and biases in the network, the error goal, momentum and learning rate. For instance, for the FFBPNN models (As shown in Fig. 2) trained with 3 to 27 hidden neurons produce inconsistent RMSE values, although both models were trained with the same other network parameter. In addition, an appropriately learning rate chosen is significant. The learning rate chosen too large appeared an unstable learning condition, and the too small value caused a slow learning condition [19]. So, the model must be trained through multiple iterations.

## VII. SENSITIVITY ANALYSES

Although the *WNID* can be estimated based on the meteorological factors by using the FFBPNN model, it is difficult to obtain explicit knowledge solely based ANN models, as it is an implicit or a black-box structure. A sensitivity analysis can be used to identify the significance of each input parameter on the objective (output) parameter in the modeling. Identifying sensitive inputs can be helpful for the model designers to carefully treat with such influential parameters in the designing models process. The Cosine Amplitude Method (CAM) is one of the sensitivity analysis methods that used to determine the effect of each individual input on the output [20]. So, in this study, CAM was adopted to determine the effect of each individual input on the output.

In this method, the degree of sensitivity of each meteorological factor is assigned by establishing the strength of the relationship ( $r_{ij}$ ) between the *WNID* and meteorological factors under consideration. The larger the value of CAM becomes, the greater is the effect on the *WNID*. To use this method, all of the data pairs were expressed in common X-space. The data pairs used to construct a data array  $X$  defined as:  $X = \{X_1, X_2, \dots, X_m\}$ . Each of the elements  $X_i$ , in the data array  $X$  is a vector of lengths, that is:  $X_i = \{x_{i1}, x_{i2}, \dots, x_{im}\}$ . Thus, each of the data pairs can be thought of as a point in m-dimensional space, where each point requires m-coordinates for a full description. Each element of a relation,  $r_{ij}$  results in a pairwise comparison of two data pairs. Therefore, the strength of the relation between the data pairs,  $x_i$  and  $x_j$ , is given by using (4):

$$r_{ij} = \frac{\sum_{k=1}^m x_{ik}x_{jk}}{\sqrt{\sum_{k=1}^m x_{ik}^2 \sum_{k=1}^m x_{jk}^2}} \quad (4)$$

Here, the strengths of relations ( $r_{ij}$  values) between the infectious diarrhea (output) and input parameters using the CAM method are shown in Fig. 6. As can be seen, the most effective parameter on the *WNID* is the weekly average temperature, whereas the weekly average rainfall is the least effective parameter on the infectious diarrhea.

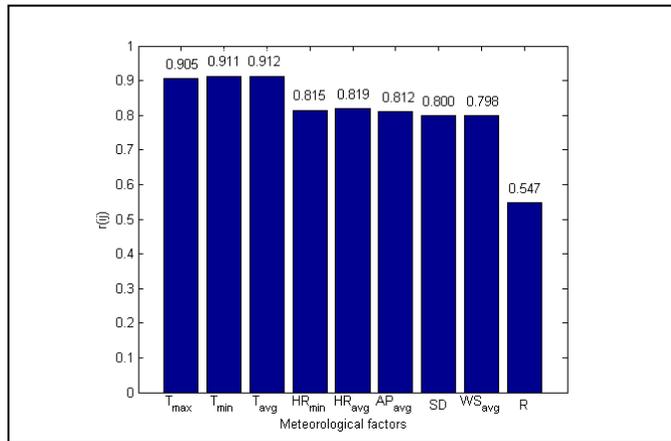


Figure 6. The effect of meteorological factors on *WNID*

## VIII. CONCLUSIONS

In this study, ANNs model (FFBPNN) was developed to predict the weekly number of infectious diarrhea of the Shanghai in China. For achieving this goal, an investigation was carried out to prove the potential of using meteorological factors dataset for producing higher accuracy of prediction. Nine meteorological factors in the Shanghai were considered. In order to obtain true and effective evaluation of the performance of FFBPNN model, the same dataset were also trained and tested by multiple linear regression models. The results presented in this paper suggested that a feed-forward back-propagation neural network (FFBPNN) model with architecture 9-4-1 has the best accurate prediction results in prediction of the weekly number of infectious diarrhea. The experiments results indicate that FFBPNN model shows a good prediction performance as compared to the traditional MLR model. Moreover, sensitivity analysis revealed that most effective meteorological factor on the infectious diarrhea is weekly average temperature, whereas weekly average rainfall is the least effective parameter on the infectious diarrhea in this study. Considering the above results, it can be concluded that the ANNs model has good capability in predicting *WNID*. Therefore, this technique can be used to predict infectious diarrhea. The results of this study may be used as a baseline against which to compare other prediction techniques in the future.

## ACKNOWLEDGMENT

This research is supported by the Fund of The Shanghai Science and Technology Development Foundation (Grant No. 13430710100). The authors are grateful to the editor and anonymous reviewers for their suggestions in improving the quality of the paper.

## REFERENCES

- [1] Charles P. Larson, Lars Henning, Stephen Luby, ASG Faruque, Infectious Childhood Diarrhea in Developing Countries, In: Modern Infectious Disease Epidemiology, Springer New York, 2010, pp. 291-308.
- [2] Koletzko S, Osterrieder S, Acute infectious diarrhea in children, Deutsches Arzteblatt International, 106(33), 2009, pp. 539-547.
- [3] McMichael AJ, Woodruff R, Climate change and infectious diseases, In: The social ecology of infectious diseases, (eds) Mayer KH, Pizer HF. Academic Press: London, 2008, pp. 378-407.
- [4] Lafferty KD, The ecology of climate change and infectious diseases, Ecology, 90(4), 2009, pp. 888-900.
- [5] Grasso M, Manera M, Chiabai A, Markandya A, The health effects of climate change: A survey of recent quantitative research, Int. J. Environ. Res. Public Health, 9(5), 2012, pp. 1523-1547.
- [6] Zhao N, M XH, Lu G, Wei YY, Sun PY Zhang DH, Research on the application of Medical-meteorological forecast model of infectious diarrhea disease in Beijing, In: Bio-Inspired Computing: Theories and Applications (BIC-TA), 2010 IEEE Fifth International Conference, 2010, pp. 100-103.
- [7] Chou WC, Wu JL, Wang YC, Huang H., Sung FC, Chuang CY, Modeling the impact of climate variability on diarrhea-associated diseases in Taiwan (1996-2007), Sci Total Environ, 409(1), 2010, pp. 43-51.
- [8] Lloyd SJ, Kovats RS, Armstrong BG, Global diarrhea morbidity, weather and climate. Climate Research, 34(2), 2007, 119-127.
- [9] Xu JF, Zhou XN, Application of artificial neural networks in infectious diseases, Chin J Parasitol Parasit Dis, 29(1), 2011, pp. 49-54.
- [10] Al-Shayea QK, Artificial neural networks in medical diagnosis, International Journal of Computer Science, 8(2), 2011, pp. 150-154.
- [11] Ma YX, Wang SG, The application of artificial neural network in the forecasting on incidence of a disease, In: Biomedical Engineering and Informatics (BMEI), 3rd International Conference on. IEEE, 2010, pp. 1269-1272.
- [12] Riahi-Madvar, Hossien, Seyed Ali Ayyoubzadeh, and Mina Gholizadeh Atani, Developing an expert system for predicting alluvial channel geometry using ANN, Expert Systems with Applications, 38(1), 2011, 215-222.
- [13] Sudhakaran R, Murugan VV, Sivasakthivel PS, Balaji M, Prediction and optimization of depth of penetration for stainless steel gas tungsten arc welded plates using artificial neural networks and simulated annealing algorithm, Neural Comput Applic, 22(3-4), 2013, pp. 637-649.
- [14] Li G, Shi J, On comparing three artificial neural networks for wind speed forecasting, Applied Energy, 87(7), 2010, pp. 2313-2320.
- [15] Hecht-Nielsen R, Kolmogorov's mapping neural network existence theorem, In: Proceedings of the first IEEE international conference on neural networks, San Diego, CA, 1987, pp. 11-14.
- [16] Moré JJ, The Levenberg-Marquardt algorithm: implementation and theory, In: Numerical analysis, Springer Berlin Heidelberg, 1978, pp. 105-116.
- [17] Piloto-Rodríguez R, Sánchez-Borroto Y, Lapuerta M, et al, Prediction of the cetane number of biodiesel using artificial neural networks and multiple linear regression, Energy Conversion and Management, 65, 2013, pp. 255-261.
- [18] Ranjith P G, Khandelwal M, Artificial neural network for prediction of air flow in a single rock joint, Neural Computing and Applications, 21(6), 2012, pp. 1413-1422.
- [19] Yang Y S, Chou J H, Huang W, et al, An Artificial Neural Network for Predicting the Friction Coefficient of Deposited Cr1-xAlxC Films, Applied Soft Computing, 13, 2013, pp. 109-115.
- [20] Jong YH, Lee CI, Influence of geological conditions on the powder factor for tunnel blasting, Int J Rock Mech Min Sci, 41(3), 2004, pp. 461-461.
- [21] Li G, Shi J, On comparing three artificial neural networks for wind speed forecasting, Applied Energy, 87(7), 2010, 2313-2320.

# Classification of Topic Evolutions in Scientific Conferences

Lin Zhang\*, Yao Guo\*, Xiangqun Chen\*, Weizhong Shao\*, and Lei Chen†

\* Key Laboratory of High-Confidence Software Technologies(Ministry of Education), Peking Univeristy, Beijing, China  
{zhanglin08, yaoguo, cherry, shaowzh}@sei.pku.edu.cn

†Department of Computer Science and Engineering, Hongkong University of Science and Technology, Hong Kong  
{csyuan, leichen}@ust.hk

**Abstract**—The number of scientific publications have been increasing explosively in recent years. Although scholar searching engines and recommendation systems help to find relevant papers, neither of them can build overviews of a certain scientific conference, which is more meaningful and important for researchers to keep up with academic trends. In this paper, we propose the concepts of topic trend and positive topic trend, and then declare the formal and quantitative definitions of topic categories, including hot topic, sunrise topic, sunset topic and emerging topic. We design an LDA-based framework to classify research topics of papers in a scientific conference into topic categories. Experiments are then constructed to study the best parameters for detecting different topic categories.

## I. INTRODUCTION

Researchers are overwhelmed due to the dramatic growth of scientific papers. Take the field of *artificial intelligence* as an example, hundreds or even thousands of research papers are published every year, making it is nearly impossible for an AI researcher to build a quick overview in a short time.

Scholar searching engines and scientific article recommendation systems [1] [2] [3] [4] [5] [6] help researchers to find relevant papers, however they cannot help to build general views over a certain research area. Kinds of works have been proposed to study topic evolutions in scientific literature[7] [8] [9] [10] [11] [12], however the answers to the following questions are still not clear:

- Which topics are hot in recent years?
- Which topics are becoming more and more popular?
- Which topics are disappearing?
- Which topics are emerging?

In this paper, we study the topic evolutions within a specific scientific conference, facing the following challenges:

- How to measure the impact of a paper on different topics, in case that it involves several topics simultaneously?
- How to measure the trends of topics and then classify them into categories, such as hot topics, sunrise topics, sunset topics or emerging topics?

We made the following contributions in this paper. 1) We proposed a novel method to measure the impact of a researcher on different topics; 2) We proposed a novel method to measure

the impact of a paper on different topics; 3) We proposed the concepts of topic trend and positive topic trend, and made the formal and quantitative definitions topic categories, including hot topic, sunrise topic, sunset topic and emerging topic; 4) We studied the best parameters for detecting different topic categories.

Given a certain conference  $c_{year}$  and its frequency  $f$ , we calculate the topic vector and reputation vector of each paper in  $c_{year}$ , and then associate these papers with their key topics to build  $c_{year}$ 's topic rank. The same process is repeated until topic evolutions during the last decade(or even longer) can be generated<sup>1</sup>. With the resulting topic trends and positive topic trends, topics can be dropped or classified into four categories: hot topic, sunrise topic, sunset topic or emerging topic.

## II. PROBLEM DESCRIPTION

Since a paper  $p$  might be involved with several topics simultaneously, we proposed a vector  $\overrightarrow{topic}(p)$  to indicate  $p$ 's topics and a vector  $\overrightarrow{reputation}(p)$  to indicate  $p$ 's different reputations on different topics. We prune some criterias demonstrated by Matsatsinis in [13], and calculate  $p$ 's reputation vector  $\overrightarrow{reputation}(p)$  as follows:

$$\overrightarrow{reputation}(p) = \overrightarrow{author}(p) + \overrightarrow{reference}(p)$$

where  $\overrightarrow{author}(p)$  indicates the author reputations on  $p$ 's different topics, and  $\overrightarrow{reference}(p)$  indicates the reference reputations on  $p$ 's different topics. We refer  $r_p$ , which is the highest reputation in  $\overrightarrow{reputation}(p)$ , as  $p$ 's key reputation, and the corresponding topic  $t_p$  as  $p$ 's key topic.

**Topic Order** In the paper set  $P_{year}^c$  of a conference  $c_{year}$ , suppose that there are  $n_i$  and  $n_j$  papers related with the topic  $t_i$  and  $t_j$ , respectively. We declare that  $t_i \succ t_j$  if and only if  $n_i \geq n_j$ , which means that  $t_i$  is more popular than  $t_j$  since the number of papers related with  $t_i$  is more than or at least equal to the number of papers related with  $t_j$ .

**Topic Rank** In the paper set  $P_{year}^c$  of a conference  $c_{year}$ , we suppose that there are  $n_1, n_2, \dots$ , and  $n_K$  papers related with the topic  $t_1, t_2, \dots$ , and  $t_K$ , respectively( $n_1 \geq n_2 \geq \dots \geq n_K$ ). Consequently, we define the topic rank  $TR_{year}^c$  as the

<sup>1</sup>The looking-back window size is defined as ten years to evaluate a certain topic with time. Similar idea has been widely employed by academic awards, such as the VLDB 10-year best paper award, the AAAI outstanding paper award and so on.

topic sequence  $\{t_1, t_2, \dots, t_K\}$ , in which  $t_1 \succcurlyeq t_2 \succcurlyeq \dots \succcurlyeq t_K$  can be guaranteed.

**Topic Rank Set** Given a specific conference  $c_{year}$ , we consider the past ten conferences to build topic evolutions, and define the topic rank set  $\mathbf{TR}_{year}^c$  as  $\{TR_{year-9*f}^c, TR_{year-8*f}^c, \dots, TR_{year-1*f}^c, TR_{year}^c\}^2$ .

**Topic Trend** With the topic rank set  $\mathbf{TR}_{year}^c$ , the topic trend of a certain topic  $t_i$  is defined as  $\{i_{-9}, i_{-8}, \dots, i_{-1}, i\}$ , in which  $i_{-9} = \text{rank}(t_i, TR_{year-9*f}^c)$ ,  $i_{-8} = \text{rank}(t_i, TR_{year-8*f}^c)$  and so on. If  $t_i$  is not contained in  $TR_{year-n*f}^c$ , which means that no papers related with  $t_i$  were accepted by the conference  $c$  in the year  $year - n * f$ , then  $i_{-n} = \text{rank}(t_i, TR_{year-n*f}^c) = -1$  ( $1 \leq n \leq 9$ ).

**Positive Topic Trend** The positive topic trend of a certain topic  $t_i$  is defined as the subsequence of  $t_i$ 's topic trend without the ranks of  $-1$ . For example, if  $t_i$ 's topic trend is  $\{8, 35, -1, -1, 9, 33, -1, 44, 37, 21\}$ , its positive topic trend will be  $\{8, 35, 9, 33, 44, 37, 21\}$ .

With the concepts of topic trends and positive topic trends, we define the following four categories:

**Definition 1: Hot Topics** Given a threshold  $\delta$ , if the ratio of the length of  $t$ 's positive topic trend to the length of  $t$ 's topic trend is equal to or larger than  $\delta$ , we classify the topic  $t$  into the topic category of hot topics.

**Definition 2: Sunrise Topics** Given a threshold  $\theta$ , if the ratio of the length of the longest decreasing subsequence(LDS) of  $t$ 's positive topic trend to the length of  $t$ 's positive topic trend is equal to or larger than  $\theta$ , we classify the topic  $t$  into the topic category of sunrise topics.

**Definition 3: Sunset Topics** Given a threshold  $\theta$ , if the ratio of the length of the longest increasing subsequence(LIS) of  $t$ 's positive topic trend to the length of  $t$ 's positive topic trend is equal to or larger than  $\theta$ , we classify the topic  $t$  into the topic category of sunset topics.

**Definition 4: Emerging Topics** If  $\text{rank}(t, TR_{year-n}^c) = -1$  for every  $n$  ( $1 \leq n \leq 9$ ), in other words, the length of  $t$ 's positive topic trend is equal to 1, we classify the topic  $t$  into the topic category of emerging topics.

**Problem** Given a conference  $c_{year}$ , classify, rank and select the papers related with Emerging topics, Sunrise topics, Hot topics and Sunset topics.

**Requirements** First, Papers are classified with their key topics into the following four topic categories: Emerging topics, Sunrise topics, Hot topics and Sunset topics. Suppose that there are  $k_E, k_I, k_H, k_D$  topics in each topic category, and papers in the conference  $c_{year}$  are related with  $K$  topics all together, then  $k_E + k_I + k_H + k_D \leq K$  is guaranteed. Second, papers are ranked with their key topics within a certain topic category. Finally, papers related with the same topic are ranked with their key reputations.

### III. SOLUTION

As shown in Figure 1, given a conference  $c_{year}$  and its frequency  $f$ , we collect papers in  $c$  from  $year - 9 * f$  to  $year$ ,

<sup>2</sup> $f$  represents the frequency of the conference  $c$ .

TABLE I. EXAMPLE

(a) The Papers of a Researcher  $r$

	1st	2nd	3rd	4th	5th	Citation Count
1	70	132	245	<b>206</b>	249	8
2	27	236	159	<b>206</b>	45	6
3	<b>206</b>	258	208	262	93	2
4	<b>206</b>	71	260	251	244	417
5	<b>111</b>	296	27	246	<b>206</b>	31
6	181	296	208	<b>111</b>	27	10
7	70	245	<b>206</b>	132	227	8
8	<b>206</b>	208	182	108	258	90

(b) The References of a Paper  $p$

	1st	2nd	3rd	4th	5th	Citation Count
1	208	8	145	220	232	200
2	117	276	208	293	<b>206</b>	29
3	209	195	208	<b>206</b>	289	140
4	23	44	245	295	181	978
5	<b>206</b>	89	19	256	285	122
6	232	258	299	156	43	53

(c) Vectors of a Paper  $p$

	1st	2nd	3rd	4th	5th
Topic Index	175	219	225	145	227
A. Reputation	234.8	415.7	354.1	25.4	9.7
R. Reputation	33	177.3	142.5	0	51

and then build the corresponding topic rank  $TR_{year-n*f}^c$  ( $0 \leq n \leq 9$ ). With the resulting topic rank set  $\mathbf{TR}_{year}^c$ , we study the topic trend and the positive topic trend of each topic  $t$  in  $TR_{year}^c$ , and then classify  $t$  into a topic category or dropped directly.

#### A. Topic and Reputation Vector

We employ Latent Dirichlet Allocation(LDA) to detect latent topics of a certain paper  $p$  to determine  $p$ 's topic vector  $\overrightarrow{\text{topic}}(p)$ . For each topic  $t$  in  $\overrightarrow{\text{topic}}(p)$ , we calculate  $p$ 's author reputation and reference reputation on  $t$ , and add them together to indicate  $p$ 's reputation on  $t$ .

In this paper, we propose a novel method to evaluate the impact of a certain researcher, which is called scientist reputation, with a set of (topic, average citations) pairs to indicate his/her different impacts on different topics. We collect all of the topics that a certain scientist might have been involved with, and then calculate the average citations from other papers on each topic. For example, the eight papers of a researcher  $r$  are listed in Table I(a), and there will be two pairs (111, 20.5) and (206, 80.29) in his scientist reputation, indicating that his reputation on the topic 206 is more influential than that on the topic 111. We scan  $p$ 's author list, and calculate the scientist reputation of each author. After that, for each topic  $t$  in  $\overrightarrow{\text{topic}}(p)$ , we add scientist reputations on  $t$  of all authors together to determine  $p$ 's author reputation  $\overrightarrow{\text{author}}(p)$ .

In this paper we introduce a vector  $\overrightarrow{\text{reputation}}(p)$  to indicate how the corresponding references of a paper  $p$  impact on its involved topics. For example, the six references of a certain paper  $p$  are listed in Table I(b), and  $p$ 's topic vector is  $\langle 206, 71, 260, 251, 244 \rangle$ . Consequently,  $p$ 's reference reputation on the topic 206 is defined as  $\frac{29+140+122}{3} = 97$ , while its reference reputation on other topics are all zero since no references are available.

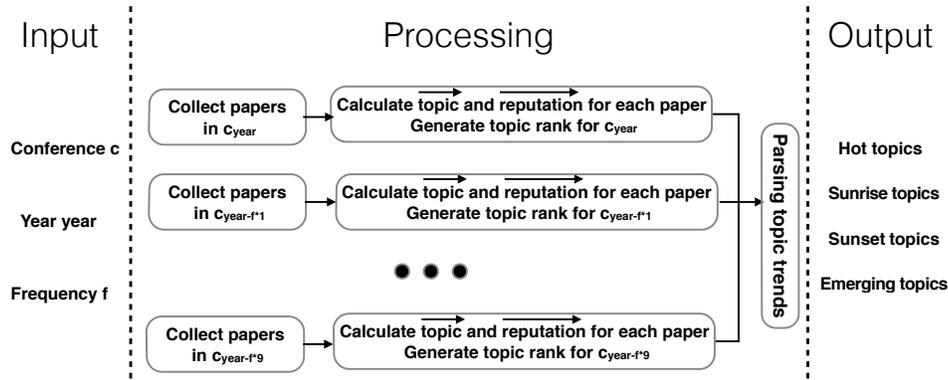


Fig. 1. Framework

### B. Key Topic and Key Reputation

Papers are classified, ranked and selected with their key topics and key reputations. We determine the key topic and key reputation of a paper based on the following hypothesis. When writing an academic paper, it is most likely for the authors to choose the most relevant references, as well as the most influential references on corresponding topics. And a scientist is most likely to publish a new paper which is related with his/her most influential topic, instead of others.

For example, Table I(c) shows the topic vector, the corresponding author and reference reputation vectors of a paper. Since the maximum reference reputation is 177.3, which means the references of the paper are most valuable/influential on the topic 219, we will set the key topic of this paper as the topic 219. And its corresponding key reputation will be  $415.7 + 177.3 = 593.0$ .

### C. Topic Trend and Topic Category

Papers are classified and ranked with their key topics and key reputations to build the topic rank set  $\mathbf{TR}_{year}^c = \{TR_{year-9*f}^c, TR_{year-8*f}^c, \dots, TR_{year-1*f}^c, TR_{year}^c\}$ . With which, the (positive) trend of a certain topic can be plotted. For example, the trend of a certain topic  $t$  is  $\{-1, 6, 25, -1, 22, 7, -1, -1, -1, 27\}$ , so that its positive topic trend is  $\{6, 25, 22, 7, 27\}$ . Consequently, one longest increasing subsequence can be  $\{6, 7, 27\}$ , and one longest decreasing subsequence can be  $\{25, 22, 7\}$ . According to our definitions, if  $\theta$  is set to be 60%,  $t$  can be classified either into sunset topics or sunrise topics. Those ambiguous topics are considered as sunset topics, so that the number of sunrise topics, which are more valuable for researchers, can be shrank.

## IV. EXPERIMENT STUDY

We retrieved the basic information of a paper from DBLP, such as title, authors, venue, and year, then applied an *APPID* from Microsoft Academic Search (MAS) to query the abstract, keywords, citations and references of each paper in the dataset. As a result, we built our dataset of 947,160 papers. We employed Mallet to calculate the topic vector of each paper, with papers published from 1995 to 2000 for training and others for inferring.

TABLE IV. PROBABILITIES

Topic Position	Maximum	Minimum	Average
1st	0.6442	0.0033	0.0856
2nd	0.2387	0.0017	0.0286
3rd	0.1400	0.0015	0.0360
4th	0.1140	0.0015	0.0291
5th	0.0839	0.0015	0.0245
6th	0.0648	0.0014	0.0212
7th	0.0570	0.0014	0.0186
8th	0.0463	0.0014	0.0186

### A. The Length of Topic Vector

Scanning “call-for-papers” of top conferences in computing science, the number of latent topics is set to be 300. For each abstract, Mallet calculates the probability distribution over topics, and sorts these topics in probabilities from high to low. We demonstrate the maximum, minimum and average probabilities on the top eight topic positions in Table IV and determine the length of topic vector as five.

### B. $\delta$ for Hot Topics

We select several conferences in computer science to study the value of  $\delta$  for Hot topic detection. For example, if  $\delta$  is defined as 0.6, hot topics in ICML2005 include the following eight topics: 68, 116, 135, 154, 167, 188, 194 and 274; And if  $\delta$  is strictly set as 1, there are only two hot topics in ICML2005, which are 116 and 188.

We define the topics, selected with  $\delta$  as 0.6, as *latent hot topics*. Then we rank these topics by topic semi-partial order and regard the top five as *target hot topics*, since it is appropriate and acceptable to recommend users with  $7 \pm 2$  items at a time. With the definition of *target hot topics*, we calculate the corresponding precision and recall for different values of  $\delta$ . It is shown that the best value of  $\delta$  for hot topic detection is 0.8 to trade-off between precision and recall. Due to space limitation, we only demonstrate parts of experiment results here in Table III.

### C. $\theta$ for Sunrise and Sunset Topics

With similar experiments, we figure out that the best value of  $\theta$  for sunrise and sunset topic detection is 0.6 and 0.7, respectively. Due to space limitation, we can not report the experiment results and detailed analysis in this paper.

TABLE II. TOPICS AND KEYWORDS

Topic	Keywords(Top 10)
68	agent, agents, multi, multiple, intelligent, intelligence, coordination, distributed, system, systems
116	example, examples, training, learning, machine, artificial, inductive, re-inforcement, hypothesis
135	estimate, estimation, sample, parameter, method, distribution, probabilistic, statistical, density, entropy
154	fast, time, times, speed, efficient, problem, solution, algorithm, approximate, optimal
167	model, models, markov, chain, chains, continuous, discrete, stochastic, hmm, hmms
188	feature, features, classifier, classification, nearest, neighbour, extraction, accuracy, recognition, discriminate
194	match, matching, algorithm, algorithms, pattern, patterns, template, templates, feature, features
274	partition, partitions, decomposition, present, presented, space, spaces, sub-space, dimension, dimensional

TABLE III. HOT TOPIC DETECTION

(a) Different values of  $\delta$ 

Venue	Year	$\delta$				
		0.6	0.7	0.8	0.9	1
ICML	2005	194, 274	68, 135, 154	167		116, 188
	2006		274	135, 167		116, 188
	2007	16, 119, 154, 258, 271		167, 274	135	116, 188
	2008	16, 27, 29, 119, 160, 194, 201, 215, 238, 258	154	167, 274	135	116, 188
	2009	16, 27, 29, 119, 194, 201, 215, 258		154, 167	135, 274	116, 188

(b) Precision and Recall

Venue	Year	Top 5 Hot Topics	$\delta$									
			0.6		0.7		0.8		0.9		1	
			Precision	Recall								
ICML	2005	116, 135, 167, 188, 274	0.6250	1.0000	0.6667	0.8000	1.0000	0.6000	1.0000	0.4000	1.0000	0.4000
	2006	116, 135, 167, 188, 274	1.0000	1.0000	1.0000	1.0000	1.0000	0.8000	1.0000	0.4000	1.0000	0.4000
	2007	116, 135, 167, 188, 274	0.5000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.6000	1.0000	0.4000
	2008	116, 135, 160, 167, 188	0.3125	1.0000	0.6667	0.8000	0.8000	0.8000	1.0000	0.6000	1.0000	0.4000
	2009	116, 135, 167, 188, 215	0.3571	1.0000	0.6667	0.8000	0.6667	0.8000	0.7500	0.6000	1.0000	0.4000

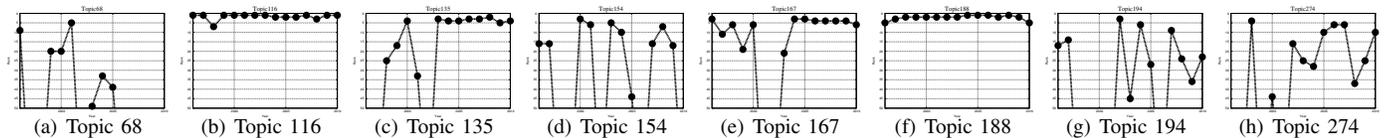


Fig. 2. Topic Evolutions over Years(ICML, 1996-2010)

## V. CONCLUSION

Scholar searching engines and scientific article recommendation systems help researchers to collect relevant papers, but cannot draw general views over a certain research field. Existing works study topic evolutions in scientific literature, however topic trends are not clear enough for researchers to catch up with academic frontiers. In this paper we proposed the concepts of topic trend and positive topic trend, and made the formal and quantitative definitions of topic categories, including hot topic, sunrise topic, sunset topic, and emerging topic. We proposed an LDA-based framework to classify topics of papers in scientific conferences into categories, and study the best parameters for detecting different topic categories.

## ACKNOWLEDGEMENT

This work is supported in part by the High-Tech Research and Development Program of China under Grant No. 2013AA01A605, the National Basic Research Program of China (973) under Grant No. 2011CB302604, the National Natural Science Foundation of China under Grant No. 61103026, 61121063, U1201255, and the NSFC/RGC Joint Research Project (No.60931160444).

## REFERENCES

- [1] T. Strohan, W. B. Croft, and D. Jensen, "Recommending Citations for Academic Papers," in *SIGIR*, 2007, pp. 705–706.
- [2] K. Sugiyama and M.-Y. Kan, "Scholarly Paper Recommendation via User's Recent Research Interests," in *JCDL*, 2010, pp. 29–38.
- [3] J. Tang and J. Zhang, "A Discriminative Approach to Topic-Based Citation Recommendation," in *PAKDD*, 2009, pp. 572–579.
- [4] Q. He, J. Pei, D. Kifer, P. Mitra, and C. L. Giles, "Context-Aware Citation Recommendation," in *WWW*, 2010, pp. 421–430.
- [5] S. M. McNee, I. Albert, D. Cosley, P. Gopalkrishnan, S. K. Lam, A. M. Rashid, J. A. Konstan, and J. Riedl, "On the Recommending of Citations for Research Papers," in *CSCW*, 2002, pp. 116–125.
- [6] C. Wang and D. M. Blei, "Collaborative Topic Modeling for Recommending Scientific Articles," in *KDD*, 2011, pp. 448–456.
- [7] D. Hall, D. Jurafsky, and C. D. Manning, "Studying the History of Ideas Using Topic Models," in *EMNLP*, 2008, pp. 363–371.
- [8] T. L. Griffiths and M. Steyvers, "Finding Scientific Topics," *Proceedings of the National Academy of Sciences*, vol. 101, no. Suppl. 1, pp. 5228–5235, April 2004.
- [9] M. Steyvers, P. Smyth, M. Rosen-Zvi, and T. L. Griffiths, "Probabilistic Author-Topic Models for Information Discovery," in *KDD*, 2004, pp. 306–315.
- [10] D. Zhou, X. Ji, H. Zha, and C. L. Giles, "Topic evolution and social interactions: How authors effect research," in *CIKM*, 2006, pp. 248–257.
- [11] Q. He, B. Chen, J. Pei, B. Qiu, P. Mitra, and L. Giles, "Detecting topic evolution in scientific literature: How can citations help?" in *CIKM*, 2009, pp. 957–966.
- [12] T. Masada and A. Takasu, "Extraction of topic evolutions from references in scientific articles and its gpu acceleration," in *CIKM*, 2012, pp. 1522–1526.
- [13] N. F. Matsatsinis, K. Lakiotaki, and P. Delias, "A System based on Multiple Criteria Analysis for Scientific Paper Recommendation," 2007.

# Applications of Slow Intelligence Frameworks for Energy-Saving Control

Wen-Hui Chen

Graduate Institute of Automation Tech.  
National Taipei University of Technology  
Taipei, Taiwan  
whchen@ntut.edu.tw

Shi-Kuo Chang

Department of Computer Science  
University of Pittsburgh  
Pittsburgh, PA 15260, USA  
chang@cs.pitt.edu

Wen-Ping Hung

TECO Electric & Machinery Co., Ltd  
Household Appliances Group R&D  
Taipei, Taiwan  
thomas@teco.com.tw

**Abstract**—The largest energy consumption, particularly electricity use, for residential users and service industry is attributed to air-conditioning systems. Although novel design can yield energy-efficient air-conditioning systems, there is still room for reducing energy consumption by the control of their operation, especially for those already installed. This study presents a slow intelligence framework for energy-saving control of air-conditioners without affecting users' thermal comfort. The implementation of the proposed approach to a smart space, and the architecture used to integrate the SIS server into the gateway as an embedded system, are also described.

**Keywords**- Energy saving control; slow intelligence systems; wireless sensor networks; embedded systems

## I. INTRODUCTION

A country's economy and its electricity use are related. The continuous growth of economy and the increase of people's living standards have caused a sharp increase in electricity demand over the last decade in Taiwan. According to the statistics released by the Bureau of Energy, Ministry of Economic Affairs (BEME), Taiwan, air-conditioning systems are the most energy consuming equipment in a building, accounting for 41 percent of the total electricity use of 7.1 billion kilowatt-hours (kWh) for eleven different service industries in 2011 [1].

As air-conditioning systems are attributed to the most energy consuming equipment, a small percentage of energy saving can lead to a significant reduction of the energy use. As such, various approaches to energy efficient design have been proposed [2]-[5]. In [6], simulations under common scenarios for residential buildings were presented to explore a better way for energy-saving design.

In addition to the design of energy efficiency for heating, ventilation, and air-conditioning (HVAC) units, energy reduction can also be achieved by the control of air conditioners [7]-[9]. In [7], the authors investigated four different control strategies to examine the energy consumption and thermal comfort for an HVAC system in an office building, and concluded some strengths as well as weaknesses for each strategy. In [8], the authors proposed a demand control strategy based on occupancy prediction models. The real time

occupancy data was derived from a wireless sensor network. A review of recent innovative cooling technologies and strategies that reduce energy consumption was revealed in [9].

There are different cooling types used for air-conditioning systems. When chillers are used for cooling to remove heat from a space, the adjustment of a chiller with its optimal loading can reduce the power demand [10]. This is because the power consumption of a chiller is highly affected by its coefficient of performance (COP), which is optimal when the chiller is operated at or near full load. When the cooling is achieved through a simple refrigeration cycle or the free cooling process, which is commonly used for residential houses and buildings, correct operating settings of air-conditioners can reduce energy consumption.

The environmental data such as temperature and humidity, and the operating settings of air-conditioners are interactively influence. This dynamic characteristic makes the energy-efficient control of air-conditioners become complex. Traditional control methods that work for a fixed operating condition suffer some limitations on tackling this problems.

In this study, an effective control strategy that adopted the principles of slow intelligence systems for energy saving was proposed. With the proposed approach, the reduction of energy consumption can be reached through continuous interaction with the environment and evolutionary computation that carries out energy-efficient operation.

The remainder of this article is organized as follows. Section 2 introduces the framework of the proposed energy saving control system. Section 3 describes communication networks and protocols used in this study. Section 4 provides experimental results and discussions. Section 5 presents the architecture to integrate the SIS server into the gateway as an embedded system, which is beneficial to the realization of the proposed system as a commercial product. Conclusions are drawn in Section 6.

## II. FRAMEWORK OF THE PROPOSED ENERGY SAVING CONTROL SYSTEM

Wireless sensor networks (WSN) consist of spatially distributed measurement nodes and a gateway to collect data.

The proposed framework for energy saving control adopted WSN to transmit data to the slow intelligence system (SIS) server, as shown in Fig. 1. The features of WSN include low power consumption, good scalability, and ease of deployment. The sensor nodes were used to measure environmental data and send it to the gateway that provides connectivity to the server for further analysis and the selection of energy-efficient settings in a real time fashion.

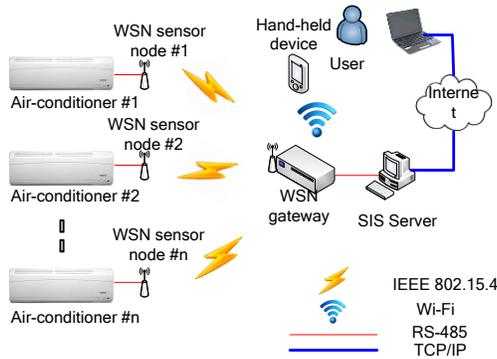


Figure 1. A framework of communication networks for energy saving control.

For residential air-conditioners, the fan speed and the operating mode of compressors are two controllable variables. Therefore, the selection of a control strategy for an air-conditioner can be considered as the search of an efficient mode for the fan speed and the compressor according to environmental conditions, such as temperature and humidity.

The control signal for adjusting air-conditioners can be sent out by the server or hand-held devices through the gateway that provides three different communication interfaces: RS-485, 802.15.4, and Wi-Fi for wire and wireless connection.

The Wi-Fi function in the gateway was implemented with the CC3000 Wi-Fi module that provides users with an easy way to setup a web server. Therefore, any hand-held devices, such as smart phones and tablets, can easily access data through the connection of socket APIs after login authentication.

In this experiment, we have developed an application on the Android platform for monitoring and controlling air-conditioners. The main functions of the developed application include the selection of operating modes, the display and the record of operating states as well as power consumption. Fig. 2 shows a snapshot of the developed app running on a smart phone.

With the proposed framework, the behavior pattern of using air-conditioners can be obtained. Behavior patterns here refer to a particular way that home appliances were used. However, behavior patterns vary from one person to another. Even for the same person, the satisfaction with a thermal environment may change under different circumstances. Therefore, thermal comfort was also taken into consideration as an impact factor to evaluate the appropriate setting of air-conditioners when considering energy-saving control.



Figure 2. A screenshot of the developed application running on a smart phone.

To determine a better operating mode of air-conditioners, the slow intelligence system was adopted. A slow intelligence system is a system that is able to improve its performance over time [11]. It consists of multiple decision cycles which can be categorized by slow decision cycles and quick decision cycles. The actions of the slow decision cycle may override actions of the quick decision cycle, resulting in poorer performance in the short run but better performance in the long run.

An appropriate control strategy under perceived sensory data can be induced through the evolutionary process of enumeration, elimination, and concentration defined in slow intelligence systems. At first, a set of rules suggested by energy domain experts was stored in the knowledge base to capture the features of energy-efficient operation. Examples of the rule set are listed as follows:

**Rule #1:** If (Temperature  $\geq 26^{\circ}\text{C}$ ) and (Humidity  $\geq 60\%$ ) and (Number of workers  $> 0$ ) then (Cooling mode)

**Rule #2:** If (Temperature  $< 26^{\circ}\text{C}$ ) and (Humidity  $\geq 60\%$ ) and (Number of workers  $> 0$ ) then (Dehumidifying mode)

**Rule #3:** If (Temperature  $< 26^{\circ}\text{C}$ ) and (Humidity  $< 60\%$ ) and (Number of workers  $> 0$ ) then (Air circulation mode)

**Rule #N:** If (Temperature  $< 26^{\circ}\text{C}$ ) and (Humidity  $< 60\%$ ) and (Number of workers = 0) then (Turn on Air conditioners)

Then, all possible combinations of operating modes and fan speeds are listed as candidate control plans, which is the process of **enumeration** in slow intelligence systems. The rules stored in the knowledge base are checked to search for the rule whose antecedent part matches the perceived sensory data readings. Once found, the matching rule is selected, and un-matching rules are excluded.

**Elimination** is the process of ruling out un-matching rules to prevent from revisiting a node during the rule matching process. This makes the search process more efficient as resources are only focused on suitable rules, which is a formation of **concentration**.

The proposed system is environmental awareness by constantly exchanging information with the environment it interacts. The operation of air-conditioners could cause the

change of the temperature and humidity in an environment, so an update of the solution according to environmental data perceived by sensors is required, which is the process of **adaptation** in slow intelligence systems.

During the search of the solution for energy-saving control, SIS updates its experience and shares the new information with other smart home appliances. For example, SIS can share the operating information of air-conditioners with networked electric fans, so those fans can turn on or off automatically. This is the characteristic of **propagation** in slow intelligence systems. For energy reduction, it is better to turn on electric fans to accelerate air circulation than to lower temperature settings at the time when an air-conditioner starts to operate. This information propagation is achievable over home networks in a smart environment.

### III. COMMUNICATION NETWORKS AND PROTOCOLS

In 2003, a number of major domestic home-appliance manufacturers in Taiwan along with Industrial Technology Research Institute initiated an organization called Smart Appliance Alliance (SAA) dedicated to develop a compatible communication platform and test standards for smart home appliances.

Officially approved by SAA, the communication middleware SAA.Net is a bi-directional control protocol used to bridge the embedded controller in a home appliance and the communication module in a smart home appliance through Universal Asynchronous Receiver/Transmitter (UART) interface, as shown in Fig. 3. The current version of SAA.Net is 3.0, which is adopted in this study. The protocol stack architecture of the SAA.Net is illustrated in Fig. 4.

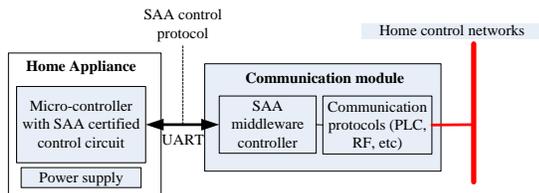


Figure 3. The communication interface for the connection of home appliances.

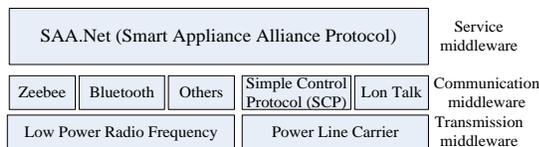


Figure 4. The protocol stack architecture of SAA.Net.

The Modbus protocol is the most widely used industrial control protocol. The format of a message packet, as shown in Fig. 5, was defined to integrate a Modbus controller with SAA certified home appliances. The client-server configuration was adopted to establish the communication link between home appliances and the SIS server.

Fig. 6 illustrates the request and response messages between the server and an air-conditioner. Fig. 6 (a) is the

message for requesting the data at the client side or the air conditioner. Fig. 6 (b) is the response message from air conditioners to the server. Fig. 6 (c) is the control message sending from the server to air-conditioners. Fig. 6 (d) is the format of a response message from air-conditioners to the server.

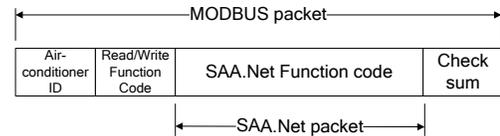


Figure 5. The message frame defined in this study for integrating the MODBUS protocol with SAA.Net.

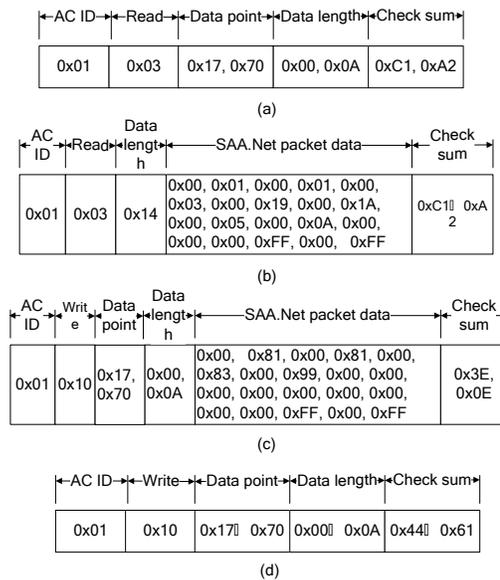


Figure 6. Examples of request and response message frames from server to air-conditioners.

### IV. EXPERIMENTAL RESULTS

An inverter split-type air-conditioner with the capacity of 5.2u kW was used in this experiment. The air-conditioner and the WSN modules were installed in a meeting room, as shown in Fig. 7.

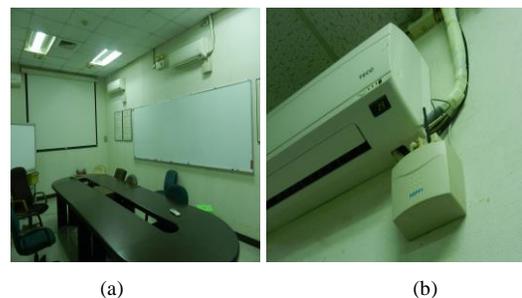


Figure 7. The layout of the meeting room for test and evaluation of the proposed approach: (a) test environment (b) the installation of air-conditioner and WSN modules.

The performance test of the air-conditioner used in this study was conducted at an enthalpy different laboratory of TECO Electric & Machinery Co., Ltd. prior to installation. A 2.4GHz wireless sensor network was deployed for environmental data collection. Table I lists three different types of sensors used in this experiment.

TABLE I. A LIST OF SENSORS USED IN THIS STUDY.

Type of sensors	Description
Infrared motion sensor	Moving objects detection
Indoor temperature sensor	Indoor temperature detection
Indoor humidity sensor	Indoor humidity detection
Outdoor temperature sensor	Outdoor temperature detection

Fig. 8 shows the operating record of the air-conditioner from 8:00 a.m. to 7:30 p.m. for one week. The black bar shown in Fig. 8 indicates the machine is in operation. Note that during the period from 10:39 a.m. to 11:02 a.m., the attendees in the meeting room left for a production line discussion so no person presented during this period. The status of the vacant room was detected by the infrared motion sensor, causing the SIS server to send out a control signal to turn off the air-conditioner.

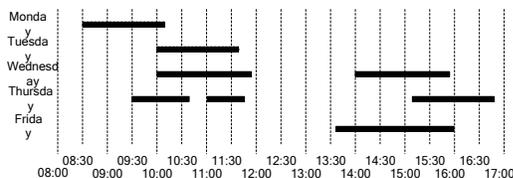


Figure 8. The operation record of the air conditioner for the period of a week.

A comparison of energy reduction with the conventional control and the proposed approach is listed in Table II. Energy reduction around 10% was achieved by using the proposed approach. The conventional control of air-conditioners is manually set by attendees, who tend to set the temperature at a bit lower and raise it when they feel cold. The proposed system can adaptively adjust the temperature according to the perceived thermal comfort and the rules defined in the knowledge base. Additionally, the proposed approach can balance thermal comfort against energy reduction.

TABLE II. A COMPARISON OF ENERGY CONSUMPTION WITH AND WITHOUT THE PROPOSED APPROACH.

Method \ Item	Power consumption (kWh)	Energy consumption (kWh/hour)
Conventional manual control	13.79	0.919
The proposed approach	10.87	0.836

## V. ARCHITECTURE OF AN EMBEDDED SIS SYSTEM

The proposed system adopted a simple but effective approach, which lends itself easily to be implemented in an

embedded system. In this section, we presented the architecture that can be employed to integrate the SIS server into the network gateway as an embedded system on an field-programmable gate array (FPGA) platform, which is beneficial to the realization of the proposed system as a commercial product.

An FPGA is a matrix of configurable logic blocks (CLBs), linked to each other by an interconnection network which is completely reprogrammable. The use of FPGA in modern digital systems has become a common practice. Since FPGAs are reprogrammable silicon chips, they can be employed to implement the slow intelligence function, which is more flexible and can deal with the changes of algorithmic parameters.

With FPGA technology, the computing tasks of algorithms in software can then be transformed down to a configuration file that contains information on how the logic blocks should be wired together. There are two types of CPU cores for the design of FPGA: hard core and soft core. A hard CPU core is a dedicated part of the integrated circuit, whereas a soft CPU core is implemented utilizing general-purpose FPGA logic cells. The following description is based on the soft core Nios II, developed by Altera Corporation as an example of FPGA-based system-on-a-chip (SoC).

A micro control unit (MCU) embedded in the FPGA system is used to reduce the complexity of hardware design needed for SoC architecture. The Nios II with a real-time operating system (RTOS) is used as the built-in MCU, and is responsible for data communication and access control. The Micrium's  $\mu\text{C}/\text{OS-II}$  is selected as the required RTOS.

The architecture of the hardware-implemented SIS-based system is shown in Fig. 9. It consists of modules that communicate through an Avalon bus interface. When the gateway receives the data transmitted from sensor nodes, it will transfer the incoming data through the UART interface to the SIS function module.

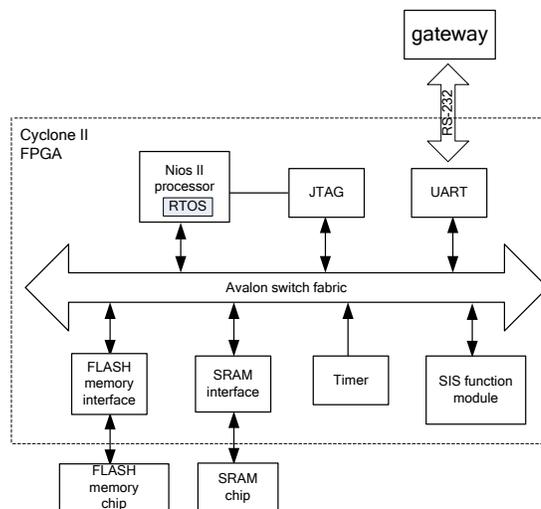


Figure 9. Architecture of the embedded SIS system implemented on Cyclone II FPGA.

## VI. CONCLUSIONS

An energy saving approach to the control of air-conditioning systems based on slow intelligence systems has been proposed. The environmental data required for analysis is collected by deploying temperature, humidity, and infrared motion sensors. A prototype of the proposed system was set up in a meeting room at TECO Co., Ltd. for verifying the feasibility and effectiveness of the proposed framework.

Although this experiment has shown some promising results, like any research, it still has some limitations. Future research based on the results of this study includes the following:

### A. Fusion of Heterogeneous Sensors

Heterogeneous sensors can provide measurements of different physical quantities. By combining various sensory data, more information could be obtained to overcome the weakness of measurement by each individual sensor. For example, an acoustic sensor can be used for auxiliary occupant detection to overcome the deficiency of motion sensors in the detection of people walking in the dead zone. With heterogeneous sensors, a more sophisticated SIS-based controller with multiple computation cycles may become necessary, posing new challenges in its design [12].

### B. Verification with more field tests

Although this study has provided a test scenario at a meeting room, more experiments can be conducted and expanded to different test environments such as residential buildings in a long period to obtain practical verification and feedback.

## REFERENCES

- [1] [http://web3.moeaboe.gov.tw/ECW/populace/news/News.aspx?kind=1&menu\\_id=41&news\\_id=2772](http://web3.moeaboe.gov.tw/ECW/populace/news/News.aspx?kind=1&menu_id=41&news_id=2772), accessed on 28 April 2013.
- [2] C. K. Cheung, R. J. Fuller, and M.B. Luther, "Energy-efficient envelope design for high-rise apartments," *Energy and Buildings*, vol. 37, no. 1, pp. 37-48, 2005.
- [3] O. M. Ai-Rabghi and M. M. Akyurt, "A survey of energy efficient strategies for effective air conditioning," *Energy Conversion and Management*, vol. 45, no. 11-12, pp. 1643-1654, July 2004.
- [4] Z. Yilmaz, "Evaluation of energy efficient design strategies for different climatic zones: Comparison of thermal performance of buildings in temperate-humid and hot-dry climate," *Energy and Buildings*, vol. 39, no. 3, pp. 306-316, 2007.
- [5] S. Mamidi, R. Maheswaran, and Y. Chang, "Smart sensing, estimation, prediction for efficient building energy management," In *Multi-agent Smart Computing Workshop*, 2011.
- [6] C. M. Lai and Y. H. Wang, "Energy-saving potential of building envelope designs in residential houses in Taiwan," *Energies*, vol. 4, pp. 2061-2076, 2011.
- [7] X. B. Yang, X. Q. Jin, Z. M. Du, B. Fan, and X. F. Chai, "Evaluation of four control strategies for building VAV air-conditioning systems," *Energy and Buildings*, vol. 43, no. 2-3, pp. 414-422, Feb. 2011.
- [8] V. L. Erickson, M. A. Carreira-Perpinan, and A. E. Cerpa, "OBSERVE: Occupancy-based system for efficient reduction of HVAC energy," *the 10th International Conference on Information Processing in Sensor Networks*, Chicago, USA, pp. 258-269, April12-14, 2011.
- [9] K. J. Chua, S. K. Chou, W. M. Yang, and J. Yan, "Achieving better energy-efficient air conditioning – A review of technologies and strategies," *Applied Energy*, vol. 104, pp. 87-104, April 2013.
- [10] Y. C. Chang, "Optimal chiller loading by evolution strategy for saving energy," *Energy and Buildings*, vol. 39, pp. 437-444, 2007.
- [11] S. K. Chang, "A general framework for slow intelligence systems," *International Journal of Software Engineering and Knowledge Engineering*, vol. 20, pp. 1-15, 2010.
- [12] S. K. Chang, W. H. Chen, Bin Kao, L. Kuang, and Y. Z. Wang, "The design of pet care systems based upon slow intelligence principles," to appear in *Int'l Journal of Software Engineering and Knowledge Engineering*, 2014.

# PSP support component integrated into a web project management environment

Antonio Marcos Neves Esteca am.esteca@sjrp.unesp.br  
Rogéria Cristiane Gratão de Souza rogeria@ibilce.unesp.br  
Adriana Barbosa Santos adriana@ibilce.unesp.br  
Carlos Roberto Valêncio valencio@ibilce.unesp.br  
Vanessa dos Anjos Borges vanessaborjes123@gmail.com

Computer Science and Statistics Department  
UNESP – São Paulo State University  
São José do Rio Preto, São Paulo, Brazil

*Abstract – Studies show that the unsatisfactory results in software projects are often related to the lack of training and commitment of human resources. In this context, this paper presents a proposal that integrates the employment of the practices proposed by the Personal Software Process (PSP) into the context of project management, aiming to support the self-improvement and commitment of each human resource, helping to manage their individual goals, which contributes to the success of projects as a whole. For this purpose, a PSP support component was created and integrated into a web project management environment, called System to Aid Project Managing (SAPM). The results were evaluated in two steps: a comparative analysis between the new version of SAPM and PSP support tools available in the market and an evaluation by a group of 22 participants, including project managers and software developers. The results revealed that the component built is broader than other tools, with the differential of bringing benefits in the context of project management.*

**Keywords - project management; personal software process; web-based tool**

## I. INTRODUCTION

The quality of software development process helps in getting a quality product, besides allowing the repetition of good results [1]. Therefore, it is necessary to incorporate a set of best practices to the software process, among which is project management [2].

Despite the continuous improvement of managerial techniques, the results obtained in software projects are still far from the expected. According to the CHAOS study [3], 42% of completed projects show changes concerning initial estimates, 21% are aborted, and only 37% are successfully completed and within initial expectations. Among the reasons given by the CHAOS study for the low success rate of software projects are the lack of training and commitment of human resources involved [4].

System to Aid Project Managing (SAPM) is a project management support web environment initially conceived to support management practices [5]. Seeking to contribute to the improvement of human resources maturity by encouraging them to maintain greater discipline and self-improvement during the achievement of their activities, this paper presents a component that was integrated into SAPM to support the

employment of Personal Software Process (PSP), which aims at helping software developers to control, manage and improve their work [6]. As a result, besides SAPM enables the use of best management practices by project managers, now each human resource counts on mechanisms to manage themselves, which assists in meeting individual goals, contributing to the success of projects as a whole.

The rest of this paper is organized as follows: Section II shows an overview of PSP and related works; Section III shows the results obtained with SAPM expansion through the integration of the PSP support component; Section IV shows the results evaluation; finally, Section V shows the concluding remarks and proposals for future works.

## II. PERSONAL SOFTWARE PROCESS

PSP was created by Watts Humphrey to be applied at a personal level by software developers aiming at their continuous improvement. For this, PSP proposes the division of software process into phases and the use of forms to record measurements carried out. Additionally, PSP is divided into six maturity levels to be achieved in a gradual manner, namely [6]: PSP 0 – it maintains the current development process and incorporates the registration of the time spent on each task and the defects generated; PSP 0.1 – it adds the measurement of size of the programs designed and the registration of process improvement proposals; PSP 1 – it adds the generation of size and time estimates to develop the programs and the creation of test reports; PSP 1.1 – it incorporates the creation of schedules to aid in the accomplishment of tasks based on estimates; PSP 2 – it incorporates reviews of codes and of the project; PSP 2.1 – it adds standard templates to design programs.

Empirical studies show that PSP provides several personal and organizational benefits, such as [6, 7]: decreased rate of defects; increase in the productivity; improvement in the estimates generated; reduced costs for software tests; increased chance of project success as a whole due to the proper achievement of the goals established at an individual level.

Given the benefits from PSP, several related works are being developed. One of the research strands conducted aims to propose tools to support the improvement of specific

aspects related to the increase of human resources productivity [8, 9] or improvement of the quality of software products developed [10, 11]. Despite the potential of these works, none of them enables the integration of PSP into a project management environment as SAPM, contributing to the success of software projects.

In addition to research with automation propositions of specific activities, several studies have aimed at improving the personal software development process as a whole. In this context, various studies have proposed computational systems to support PSP employment [12, 13, 14, 15, 16, 17]. However, most of these proposals do not provide support at all PSP levels, or when they do, no implementation support of important elements is offered, such as time measurement of defects correction and process indicators generation.

In this scenario, it is possible to note the importance of this work, which presents a component to support the implementation of all PSP elements with the important advantage of being integrated into a project management environment, the SAPM. This integration enables human resources to act according to their duties in the projects, contributing to the achievement of the project goals as a whole.

### III. PSP SUPPORT COMPONENT

The development of the proposed component started with the study of the main PSP guides [6, 18] in order to identify the elements and techniques to be supported. Next, the requirements were classified and grouped into a new component to be incorporated into the SAPM software architecture [5]. Finally, after restructuring the SAPM software architecture, the identified requirements were implemented and integrated with SAPM environment, using the following computing resources: PHP, JavaScript and HTML; database manager MySQL; Apache web server.

The model established for SAPM operation is shown in Figure 1. As presented in the illustration, the system was divided into two access areas. The Managerial Area is dedicated to the execution of managerial activities under the responsibility of managers and project team members. Its functions are presented by Ref [5]. The Personal Area is provided by the component proposed by this work and it is dedicated to the employment of PSP techniques at a personal scope. This model was established in order to avoid the overburdening of the software process. Since each team member is responsible for the personal application of PSP techniques, the project managers remain responsible only for management activities. Thus, according to the model, project managers will assign activities to team members through the Managerial Area. In turn, the team members will receive these activities in the Personal Area, where they may manage them using the resources provided.

By accessing the system, a user can select a registered project to which it is allocated. At the first access to each project, users must define if they want to use PSP support. Figure 2 (a) illustrates the main page of the Managerial Area,

shown after the selection of a project. In this figure is highlighted the main menu item related to PSP. Figure 2 (b) illustrates the Personal Area main page accessed by the link "Personal Area" in the main menu of SAPM. In the figure example, was selected a project in which the user Bob is at the last maturity level of PSP. The users achieve a new PSP level when they complete a project applying the previous level, which helps them to mature gradually.

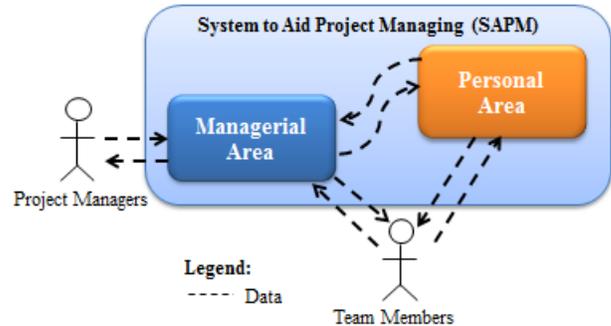


Figure 1. SAPM operation model.

Next, the functions provided in the Personal Area are briefly presented according to each PSP level.

- **Level 0**

The main functions concerning PSP level 0 are: registration of programs to be constructed; registration of tasks involved in the construction of each program; use of Time Recording Log; use of Defect Recording Log; sending programs to test; record of completion of tasks and personal programs.

The register of tasks involved in the construction of programs can be quickly performed, since the system allows the registration of standard tasks, which can be subsequently imported to other programs without the need for typing.

For the employment of Time Recording Log, the user must record the time spent on each task and interruptions occurred.

The use of the Defect Log Recording, in its turn, enables to record the type of each defect found in programs developed and the development phase in which such defects were injected and removed.

Figures 3 and 4 show the devices made available by SAPM to support employment of Time Recording Log and Defect Recording Log, respectively, which operate in the background and record the necessary information by means of stopwatches, which can be controlled by shortcut keys. In the case of Time Recording Log, the only information that the user needs to type is a word that describes each interruption. In turn, for the Defect Recording Log, the user needs only to select the phases of injection and removal of defects, besides the type of defect.

From the data recorded by Time Recording Log and Defect Recording Log, are generated various personal indicators of productivity and quality, such as the percentage of time spent on interruptions and the rate of generation of defects, contributing to developers' self-knowledge, which is one of the main goals of PSP 0.

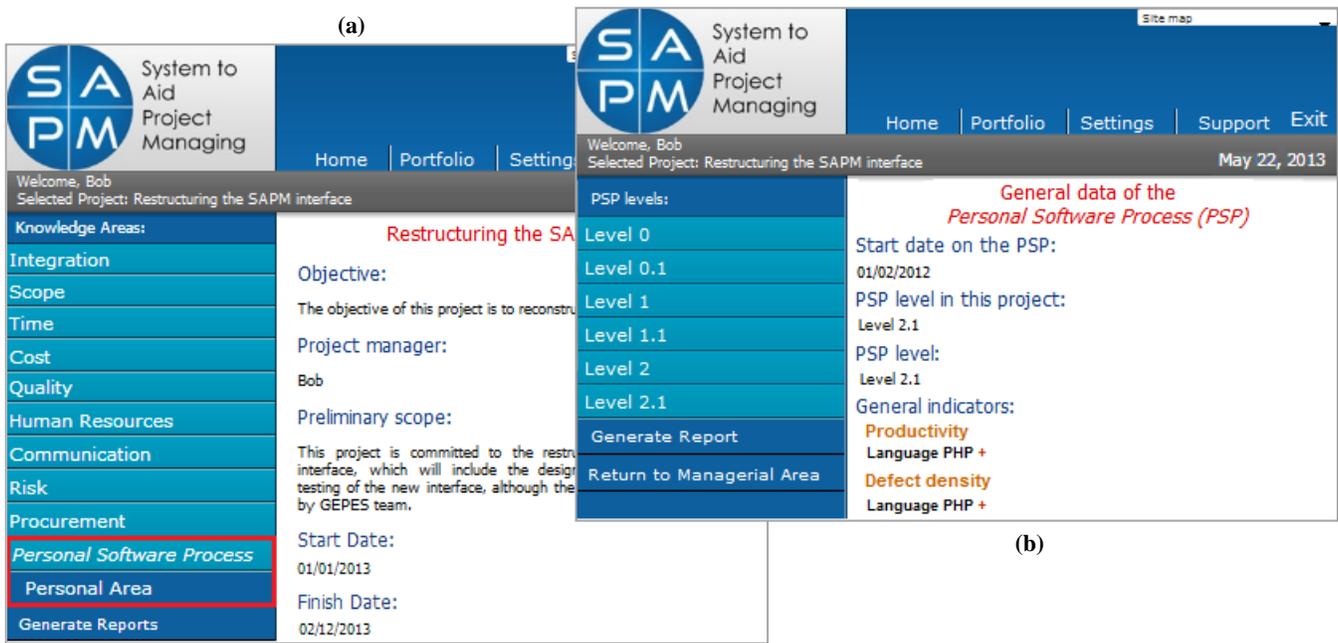


Figure 2. Main Page of Managerial Area (a) and of Personal Area (b).

Figure 3 shows a time recording log interface. At the top, there is a digital clock displaying '20:16' and control buttons for play, pause, and stop. Below the clock is a table with the following data:

Date	Start	End	Interruption	Delta	Activity
27/05/2013	02:12	02:36	2, Meeting 2, Snack	20min	Prog. solution

Figure 3. Time Recording Log.

Figure 4 shows a defect recording log interface. At the top, there is a digital clock displaying '12:16' and control buttons for play/pause and clean. Below the clock is a table with the following data:

Defect	Time	Total Time	Inj. phase	Rem. phase	Type of def.
1	3	3	Coding	Code rev	Function
2	5	8	Project	Project re	Environm
3	4	12	Coding	Code rev	Function

Figure 4. Defect Recording Log.

Also at level 0, users can send for test the files containing the source code of their programs, which are made available to all testers allocated to the project on a page of Managerial Area referring to Quality Management. When a tester accesses the program, he has at his disposal the device for the employment of the Defect Recording Log. Data on the tests performed are stored for the program developer, in order to maintain registers that will be used to generate indicators.

Finally, users can record completion of each task and program. Furthermore, all information collected and generated indicators can be viewed on Project Plan Summary, which is available at all PSP levels. At each level, new information is added to the Summary, as proposed by the PSP.

- **Level 0.1**

The main functions on PSP level 0.1 are: registration of a coding standard for each programming language; registration

of data obtained from software measurements; registration of proposals for improving the process.

The system provides a page where users can register a coding standard adopted for each programming language, being possible to upload a file with the existing standard.

The registration of data obtained from program measurements should be done by filling out a form. In the specific case of counting the lines of code in a program, despite the fact that SAPM does not perform automatic classification, the system indicates internet addresses for downloading free programs that perform this process.

Finally, at level 0.1, the system provides a form to record process improvement proposals, which are also stored as lessons learned at the Managerial Area.

- **Level 1**

The main functions concerning PSP level 1 are: automation of the PROBE method, which aims at generating estimates of size and time required for developing programs; record of estimated time to accomplish each task; registration of tests.

Although PROBE method generates estimates very close to reality, its implementation requires complex procedures and calculations. Therefore, the solution implemented in SAPM automates most of the execution of the method, making its complexity transparent to the users. The only entries that the user needs to provide the system with are: definition of a proxy for each programming language, which consists of a program component whose size is proportional to the time of development [6]; registration of proxy size in programs already constructed using PSP levels 0 and 0.1; registration of actual size for programs whose estimates were performed using the PROBE method. From the record of the actual data size, the degree of accuracy of PROBE method increases with each new program.

From the total estimated time for the development of programs, users can establish the estimated time to develop each task related to the program.

Finally, the user can also record program test reports or send these programs to be tested by software testers.

- **Level 1.1**

The main functions concerning PSP level 1.1 are: distribution of total estimated time for program development between PSP phases; generation of schedule for personal tasks, presenting the Cost Performance Index (CPI) and project progress; presentation of comparison between allocations of human resources made by project managers, and the work plan established by such resources.

The distribution of the total estimated time between PSP phases is automatically performed based on the historical percentage of time spent in each phase. This distribution helps users of the system to define the time of each personal task, based on the estimated duration of PSP phases.

From the sequencing of programs and its activities that must be performed by the user, the system automatically generates a personal schedule for each PSP user, which is represented by a Gantt chart. Along with the schedule, it is also presented the CPI value and the percentage of project progress, calculated by the Earned Value Management (EVM) technique [6].

Finally, at level 1.1 is presented a comparative between human resource allocations made by project managers and by their own human resources, as shown in Figure 5. This comparison can direct managers and team members to review the estimates made, in case of significant differences.

Project Manager Estimation x Personal estimation			Help
Activities of the project	Hours of allocation	Personal programs	Personal Estimate(hours)
Create Access Control	40	Customer registration, Login	31,7
Create a search system	21	Entry of keywords, Search for results, Presentation of results	29

Figure 5. Comparative: estimates from project manager vs. personal estimates.

- **Level 2**

The main functions concerning PSP level 2 are: record of checks performed by means of design review checklist and code review checklist; automatic calculation of indicators of quality and productivity.

Users can apply the checklists proposed by PSP for each program, recording the items checked in the design review and code review phases.

Finally, the system provides a dashboard to display the following indicators on quality and productivity [6]: Defect Removal Yield; Defect Density; Defect Rate; Cost of Quality (COQ); Defect Removal Leverage; Process Quality Index; CPI; Lines of Code/Hour; Time Spent on Interruptions. In addition, the system also allows you to view the evolution of these indicators along the projects, which helps to assess the evolution of personal maturity. In Figure 6, it is illustrated, as an example, the COQ and its evolution.

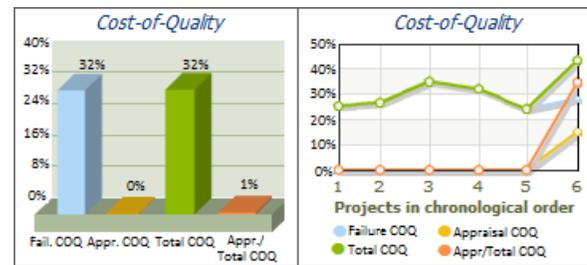


Figure 6. Example of indicators in the Personal Area.

- **Level 2.1**

The main function concerning PSP level 2.1 is the registration of the functional, operational, logical and state specification templates for each program, which represent in tables, respectively, the class diagram, the use case diagram, the pseudo-code and the state diagrams [6].

#### IV. EVALUATION

Results evaluation was performed in two steps: comparative analysis between the new version of SAPM and PSP support tools available in the market, revealing a greater breadth of SAPM; evaluation of the results by a group of 22 participants, including project managers and software developers, which demonstrated the benefits offered by the proposed approach, as well as detected points to be improved, guiding future works.

These two steps are presented in the following subsections.

##### A. Comparative analysis with other PSP support tools

This step consisted in the comparative analysis between SAPM and six other PSP support tools available in the market, namely: Process Dashboard [12]; Jasmine [13]; Hackstat [14]; PSPA [15]; PSP.NET [16]; Eclipse Plugin [17]. The selection of these tools include the most cited in articles consulted during the study of related work. After selection, the requirements to be analyzed were defined, which sought to evaluate the scope of the support to the elements and practices of each PSP level. Such requirements were assessed using an observational analysis that considered a range of 0 to 100%, ranging from 25 to 25%, depending on the coverage offered by the tools to the items considered. Thus, if one criterion is assessed with 75% in tool A and with 50% in tool B, for example, tool A permits an improved approach to this aspect.

Table III shows the results of the evaluation. As it can be seen, in general, SAPM presents coverage equal to or better than the other tools evaluated, except for the requirement “Measurement and Classification of lines of code”, for which the system indicates internet addresses for specific tools that may be easily used together with SAPM.

One of the outstanding points identified is that most tools offer little or no support for PSP 1, which is one of the most important levels, since it allows to obtain very accurate estimates of size and time for software development. Moreover, only PSP.NET tool approaches level 2.1. Finally, it is worth highlighting that SAPM is the only tool analyzed that is integrated into a project management environment.

TABLE III. RESULT OF THE ANALYSIS OF PSP SUPPORT TOOLS

	Process Dashboard	Jasmine	Hackystat	PSPA	PSP.NET	Plug-in Eclipse	SAPM
<b>PSP 0</b>							
Registration of time and defects	100	100	100	100	100	100	100
Registration of defect type standard	0	75	0	0	75	0	50
<b>PSP 0.1</b>							
Registration of coding standard	0	0	0	0	0	0	100
Measurement and classification of lines of code	100	100	100	75	75	75	50
Registration of process improvement proposals	0	0	0	100	0	0	100
<b>PSP 1</b>							
Automatic generation of estimates	100	25	25	25	25	0	100
Registration of test reports	0	25	0	100	50	0	100
<b>PSP 1.1</b>							
Generation of personal schedule	75	75	0	75	75	0	100
Automatic calculation of project progress	25	75	0	75	75	0	100
Automatic distribution of time between PSP phases	0	100	0	0	100	100	100
<b>PSP 2</b>							
Registration of design and code reviews	75	75	25	75	25	25	75
Generation of quality indicators	100	75	50	50	50	50	100
Estimation of defects in each PSP phase	75	75	0	0	75	75	100
<b>PSP 2.1</b>							
Support to design specification	0	0	0	0	100	0	100

In relation to the points where SAPM did not get 100%, the main shortcomings observed were:

1. Registration of defect type standard: SAPM considers only the types of defects proposed by PSP, not allowing the registration of other types;
2. Measurement and classification of lines of code: SAPM does not measure and automatically sorts the lines of code of programs built, making it necessary the use of other tools to obtain measurements;
3. Registration of code and design reviews: SAPM considers only those items proposed by PSP in the review checklists, not allowing the registration of additional.

*B. Vision of SAPM users*

The objective of this stage was to analyze the vision of users concerning the contribution of this work. With this aim, it was performed a course that counted on the participation of 22 professionals from small and medium sized software development enterprises, being 8 project managers and 14 software developers. In Figure 7, it is shown the profile of the participants as to the length of professional experience, revealing the extent of the selected sample. Despite the difference in the level and length of professional experience of

participants, the answers received during the course presented no significant differences.

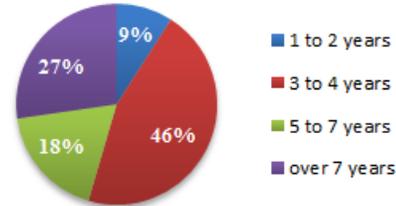


Figure 7. Profile of participants.

The course was divided into two parts, performed in 4 consecutive days with 4 hours a day.

The first part took place in the first 3 days and consisted of the application of lectures that addressed all the PSP elements and practices, with the goal of ensuring that all participants had knowledge of the process.

The second part took place on the last day of the course and consisted of presenting SAPM, addressing both the Managerial Area and the Personal Area, with a focus on PSP support functions.

At the end of each stage of the course, a questionnaire was administered to participants. It is observed that the participants were not identified in the questionnaires, ensuring an unbiased assessment. Also, when starting the second questionnaire it was not possible to alter the responses of the first.

In the first questionnaire, the 14 software developers made an overall assessment of the PSP, judging three assertions as to the degree of agreement, measured on a scale of 1 (strongly disagree) to 5 (strongly agree). The assertions were: **Learning (L)** - A developer who does not know PSP can learn the practices in a short time; **Simplicity (S)** - PSP simplicity encourages its use; **Motivation (Mo)** - I intend to use PSP in my routine.

Now, the second questionnaire was divided into two parts. The first part consisted of the same questionnaire used in the first step, in order to verify any changes in the vision of the software developers after presenting SAPM. The second part sought to determine the view of project managers as to the contribution of PSP to the project management, therefore, it was answered only by the 8 project managers that participated. The following items were evaluated on a scale of 1 (worst evaluation) to 5 (best evaluation): **Estimate (E)** - Aid for time estimation of activities; **Progress (Pr)** - Aid for project progress control; **Quality (Q)** - Aid for project quality control; **Indicators (I)** - Importance of indicators generated.

Moreover, at the end of the first and the second part of the second questionnaire, open fields were left to describe the strengths and weaknesses of SAPM.

Figure 8 shows the comparison between the evaluation of the statements before and after the presentation of SAPM, considering the average of the scores given by the participants. Observing the graph, it is clear that there was a strong increase in agreement as to the assertions, which allows to affirm that participants felt more interested to use PSP due to computational support that met their needs.

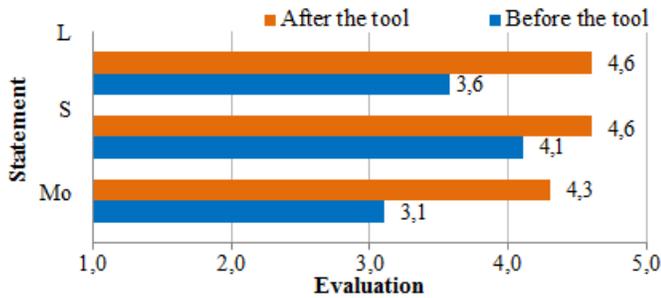


Figure 8. Assessment regarding agreement with assertives.

Figure 9 presents the results for the second part of the second questionnaire, answered by managers. Observing the graph, it can be seen that the evaluation was very good, since all of the queries received average evaluation close to 4,5.

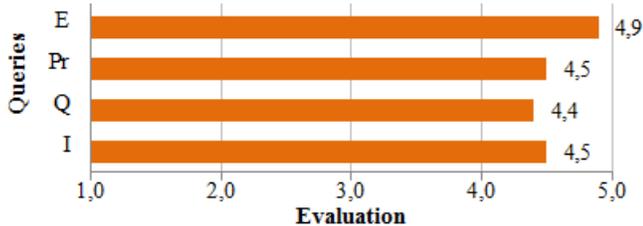


Figure 9. Evaluation of the contribution resulting from the integration of PSP into project management.

According to software developers, the main strengths reported through the open fields were: ease of use of Time Recording Log and Defect Recording Log; ease of generation of estimates; agility to generate various types of reports; quality of panel with quality and productivity indicators. As to weaknesses, stand out: lack of automated classifier of lines of code; availability only for Mozilla Firefox.

Considering the project managers, one of the main advantages of the proposal is that the mechanisms provided in the Personal Area stimulate human resources to commit themselves to the project activities and to the improvement of the development process quality, which is a key element for the good progress of projects. Additionally, the comparison between managers' and human resources' estimates helps to identify the need for changes in the estimates made at an early stage of projects. As a weak point, project managers indicated the lack of general indicators that reveal the teams' average behavior based on individual indicators.

## V. FINAL CONSIDERATIONS AND FUTURE WORKS

This work presented a proposal that integrates the employment of PSP into the context of project management in order to contribute to the improvement of human resources maturity, which helps in meeting the goals of the projects. For this purpose, a PSP support component was integrated into a web project management environment, the SAPM.

Results were evaluated in two stages. The first stage allowed to confirm that SAPM is broader concerning PSP when compared to other tools available in the market. The second stage showed that SAPM makes users more interested in applying PSP, besides bringing benefits in the context of project management, such as the support to the identification of

needs for changes in estimates generated.

Although the results obtained were quite favourable, future studies will be developed to improve the proposed solution concerning weaknesses. Furthermore, in order to contribute to the advancement of research in the context of project management, data collected through the individual employment of PSP will be analyzed to identify indicators that can reflect the behavior of the team involved in the project, which should be made available to project managers to guide their decisions.

## ACKNOWLEDGMENT

This research was partially supported by The State of São Paulo Research Foundation under projects number 2008/07094-2 and 2010/13478-8.

## REFERENCES

- [1] I. Sommerville. *Software Engineering*, 9th ed. Boston, MA: Pearson, 2011.
- [2] PMBoK. *A Guide to the Project Management Body of Knowledge*. 5th ed., Project Management Institute - PMI, 2013.
- [3] Standish Group. *Chaos Report*. Boston, MA: The Standish Group International, 2011.
- [4] W. Jirachiefpattan, "Understanding the influence of individual values on management of software development: Thai IT Professionals," *International Association of Computer Science & Information Technology*, vol. 36, pp. 11 - 15, April 2012.
- [5] R. C. G. Souza, A. M. N. Esteca, A. B. Santos, C. R. Valêncio, and M. T. Honda, "Web System to Aid Project Management," in *Proceedings of the 23rd Conference on Software Engineering and Knowledge Engineering (SEKE)*, United States of America, pp. 325 - 330, 2011.
- [6] W. S. Humphrey. *PSP: A Self-Improvement Process for Software Engineers*. Westford, MA: Addison-Wesley Publishers, 2005.
- [7] C. Wohlin, "The Personal Software Process as a context for empirical studies," *IEEE TCSE Software Process Newsletter* 12, pp. 7 - 12, 1998.
- [8] Epiforge Software. "Grindstone Software". Available at: <http://www.epiforge.com/Grindstone/>, June 2013.
- [9] Responsive Software. "Responsive Time Logger". Available at: <http://responsivesoftware.com/>, June 2013.
- [10] B. Brykczynski, R. Meeson, and D. A. Wheeler, "Software inspection: Eliminating software defects," in *Proceedings of the 6th Annual Software Technology Conference*, Egito, May 1994.
- [11] Bugzilla. "Bugzilla Mozilla". Available at: <https://bugzilla.mozilla.org/>, June 2013.
- [12] Tuma Solutions. "Process Dashboard". Available at: <http://www.processdash.com/>, June 2013.
- [13] H. Shin, H. J. Choi, and J. Baik, "Jasmine: A PSP Supporting Tool," in *Internacional Conference on Software Process*, United States of America, pp. 73 - 83, 2007.
- [14] P. M. Johnson, H. Kou, M. G. Paulding, Q. Zhang, A. Kagawa, and T. Yamashita, "Improving software development management through software project telemetry," *IEEE Software*, vol. 22, n. 4, pp. 76 - 85, August 2005.
- [15] R. Sison, D. Diaz, E. Lam, D. Navarro, and J. Navarro, "Personal Software Process (PSP) Assistant," in *Proceedings of the 12th Asia-Pacific Software Engineering Conference*, United States of America, pp. 687 - 696, December 2005.
- [16] M. H. N. M. Nasir, and A. M. Yusof, "Automating a modified Personal Software Process," *Malaysian Journal of Computer Science*, vol. 18, n. 2, pp. 11 - 27, December 2005.
- [17] X. Yuan, P. Vega, H. Yu, and Y. Li, "A Personal Software Process tool for Eclipse environment," in *Proceedings of the International Conference on Software Engineering Research and Practice*, United States of America, pp. 717 - 723, 2005.
- [18] M. P. Huff, R. Cannon, A. T. Chick, J. Mullaney, and W. Nichols. *The Personal Software Process (PSP) Body of Knowledge*. Technical Report, CMU/SEI-2009-SR-018, v2.0, 2009.

# A Semantic Analyzer for Simple Games Source Codes to Programming Learning

Elanne Cristina Oliveira dos Santos<sup>1,2</sup>

Victor Hugo Vieira de Sousa<sup>1</sup>

<sup>1</sup>Instituto Federal de Educação, Ciência e Tecnologia do  
Piauí, IFPI  
Teresina, Brasil  
elannecristina.santos@ifpi.edu.br  
vhv.sousa@gmail.com

Gleison Brito Batista<sup>2</sup>

Esteban Walter Gonzales Clua<sup>2</sup>

Instituto de Computação  
<sup>2</sup>Universidade Federal de Fluminense, UFF  
Niterói, Brasil  
gbatista@ic.uff.br  
esteban@ic.uff.br

**Abstract**— The teaching of programming and algorithms consists in a big challenge, not only in universities but also in schools and training centers. Many proposals for stimulating this process were made in the last years. Previously to this work we proposed a semantic analyzer based on behaviors comparison between two programs, the model program and a student program. The programs compared are simple 2D games developed by JPlay framework. The JPlay was proposed and developed for teaching programming and consist on a framework that facilitates the teaching of programming. Thus, in order to compare the behaviors of two programs we developed a comparison algorithm between the classes of the model program and student program, resulting in similar pairs of classes. While in a previous work we proposed how to combine similar classes of two programs, in this paper we show how to analyze similar classes, based on a variable behavior strategy. These behaviors are identified based at an occurrence scope. Therefore, when behaviors differences are identified, our proposed system makes semantical suggestions about these behaviors differences found in the student program. The teacher adds and edits the suggestions using comments in the source code of the model. In this paper, we will also show some results of this comparison.

**Keywords**- *programming; JPlay; knowledge modelling; classification; code comparison; teaching; games*

## I. INTRODUCTION

The teaching of programming and algorithms consists in a big challenge, not only in universities but also in schools and training centers. Studies point to the difficulty of teaching and learning the disciplines related to algorithms and programming, resulting in high dropout rates in computer courses [13][5]. The main reason for this negligence is the difficulty in learning abstract concepts of programming [16]. Many proposals for stimulating this process were made in the last years [3][12][17][4].

In this sense, the JPlay framework was proposed and developed for teaching programming [9][11]. JPlay is a framework for facilitating the teaching of programming, providing an algorithmic learning process related with the logic of simple 2D game development. JPlay does not interfere with

the structure of basic programming necessary for a correct learning of algorithmic logic and does not introduce specific features of design patterns or stereotypes of games in the source code. The tool allows the students an easy way to draw and move images on a computer screen and provides methods and objects that help to create 2D games using the Java language.

Previously to this work we proposed a semantic analyzer based on behaviors comparison between two programs, the model program and a student program. The programs compared are simple 2D games developed by JPlay framework. Thus, in order to compare the behaviors of two programs we developed a comparison algorithm between the classes of the model program and student program, resulting in similar pairs of classes. Previously, we show how the algorithm combined the similar classes of the two programs [15]. In this paper we will show how the similar classes will be analyzed. This analysis will show more specifics results using a comparison algorithm between the variables of same type of each classes pair, analyzing its game context instead syntax details and based on variables behaviors. Behaviors are identified through the analysis of its occurrence scope at the source code, such as an assignment, a loop or a conditional usage. Therefore, when behaviors differences are identified, the system makes suggestions about these behaviors differences found in the student program and clues that may indicate a possible semantic error. The teacher adds the suggestions in the form of comments in the source code of the model program, which are automatically captured by our solution. In this paper, we will also show some results of this comparison.

## II. RELATED WORK

During the process of learning programming different techniques can be used for students in addition of learning to program with the purpose of acquiring good programming practices. The techniques are classified as follows: tests based programming, programming pattern, automatic evaluator, programs diagnostic systems [13].

Based on previous works [7][13], we classify the techniques related to programming learning context as follows: programming based unit testing, proposals of programming

environments, automatic evaluator, analysis of programming patterns and automatic deputation systems (intelligent tutoring systems and programs diagnostic systems).

In programming based unit the teacher provides a set of specific tests to solve a particular problem and the student must build a program that allows the achievement of expected results in the execution of all tests [13][17]. The proposals of programming environments consist on the fact that some development tools were created in order to assist students in introductory programming, such as BlueJ and DrJava [12][3].

The automatic evaluator is used to help the teachers with tasks of activities corrections. The teacher can define acceptance tests to be automatically executed after the students deliver their programming activities and results of the tests can be used to compose the final score the student [13]. We can quote the Web-Cat [4] as an automatic evaluation tool.

The analysis of programming patterns is based on research of programming learning suggestions that experienced developers solved when looking for previous solutions that are related to the new problem and that can be adapted to the ideal situation [7]. Thus, the concept of patterns is based on the fact that experienced programmers are able of solve new problems through the analysis of a previously solved problem. They can identify what structure to use, what types of data is involved, as well as other ways to solve the same problem, through previous experiences that identify solutions [2]. Previous experiments contain the basis for Programming patterns, which are solutions that often appear in solving computational problems [2]. Thus, patterns translating programming strategies created by experts can lead to good programming practices. We can quote the systems Proust [10] and PROPAT [7] as systems that use the strategy of analysis of programming patterns.

An automatic deputation system is a system that uses techniques in order to find and classify components from a program. Based on the type of technique used, this may be classified as a programming intelligent tutoring systems and program diagnostic system.

Laura is one of the first attempts to build a tutoring system for teaching programming and is written in Fortran [6]. Its strategy is a comparison between two programs, the model and the candidate. The comparison is possible through the representation of the model and candidate programs by graphs, and its heuristic strategy to identify step by step the elements of the graphs [1].

We have previously developed a knowledge modeling system for semantic analysis of Games [15] and is based at learning objectives. It aims to find and to classify possible errors that happen in the program. Therefore it can be used in order to guide the student about these errors. It has a function of interpreting semantically and architecturally a Java program developed that uses the JPlay and return results of this examination to the programmer. The process consists on a comparison between the student program and model program. For this, the programmer must select, in his integrated development environment (IDE) tool, the model program that he wants to use as reference, previously available in a repository. Thus, the analyzer is able to interpret semantically

the program that is being built by the student, may point out problems and suggest possible solutions.

The comparison is based on behaviors of the programs. Different behaviors between the model program and student program produce suggestions about possible errors in the student's source code. Thus, in order to compare the behaviors of two programs we developed a comparison algorithm between the classes of the model program and student program, resulting in similar classes pairs. Previously, we show how the algorithm combined the similar classes of the two programs [15].

After the selection of similar classes, variables of each pair are compared and similar variables pairs are selected. Then, at this stage, the analysis shows more specifics results using a comparison algorithm between the variables of the same type of the pairs of classes. The comparison is based on variables behaviors. Behaviors are identified through the occurrence, at the source code, of assignments, loops or conditionals statements. Therefore, when behavioral differences are identified at the student program, the algorithm makes suggestions about. The teacher adds these suggestions in the form of comments in the source code of the model program.

A difference is that our proposal is based on a design pattern oriented to simple 2D game, following the original purpose of JPlay [9][11]. One more important difference concerns the comparison between similar variables. In the case of Laura, for example, two graphs of the model program and candidate program are built and compared. In our work we can build a data structure (behavior tree) starting from the behaviors of the variable, and compare each of these structures, thus obtaining a higher level of granularity in this heuristic strategy in order to identify behaviors differences between programs.

### III. JPLAY

Our proposal is based in the JPlay framework. JPlay was previously developed with the purpose of teaching computer science and algorithms based in game development. In order to identify sequential patterns in JPlay architecture, we divide the JPlay diagram into three parts: the interaction between game and player, characters and output game.

The classes responsible for interaction between game and player are: Keyboard (define input data for the keyboard) and Mouse (define input data for mouse). The classes responsible for creating the characters of the game are: Animation (defines an animation. It must have a picture and their frames. A frame is a piece of the picture responsible for the movement of animation), Sprite (the Sprite class extends the Animation class. The Sprite class contains methods that can make the image move across the screen) and Body (the Body class extends the Animation class. Like Sprite, the Body class also contains methods that can move the image, and beyond these methods it adds methods to accelerate and decelerate the image across the screen). The classes responsible for outputs in the game are: Window (defines a window where all the game elements will be drawn), Time (defines a time counter), Sound (defines the sound that will be played in the game) and Collision (it is a static class, used to check if there was a

collision between two objects. The occurrence of a collision can be verified using this method in all classes, or by the Collision static class).

Our proposal is based on design patterns of sequence used at the JPlay framework. JPlay follows a typical game framework pattern: objects, also called as game objects, are initially defined. A loop is initiated (also called as a game loop) and each iteration corresponds to a frame being produced. In this loop all game objects are updated with their corresponding logic (coming from an AI algorithm, physic algorithm or even from the user interface sequence). Finally, all the elements of the game are drawn in the screen.

#### IV. SEMANTIC ANALYZER ARCHITECTURE

We propose in this work a semantic analyzer for the code being generated by the students. The system is composed by many stages and modules, which are illustrated in Fig. 1.

Basically, the system starts classifying pairs of classes that are taken from both the model class, which corresponds to the model program (teacher’s program), and student code (1). The analyzer then checks if the pairs of classes are standardized according to the properties defined by the teacher for each exercise, which corresponds to the model program (2).

Thus, in the standardization phase (2), the student’s code must be standardized according to the model program so that they can be compared later. Thus, a markers structure must be filled in each of the classes of the program model. Each marker indicates the characteristics that the class must have to implement their behavior. The teacher must inform the values of the markers in each of the classes of the program model. Then, the analyzer identifies, in the student code, the values of the markers of each class. Some of the markers that are used are: inheritance, constructor, movex, movey, keyboard, mouse, method, game object, main and game loop. Although we define basic markers, more specific markers can be defined by the teacher. If the standardization is correct for each pair of classes, the analyzer starts the comparison process (4), classifying variables pairs between pair of classes, otherwise the analyzer requests the student adaptations in the code and the process returns to the beginning.

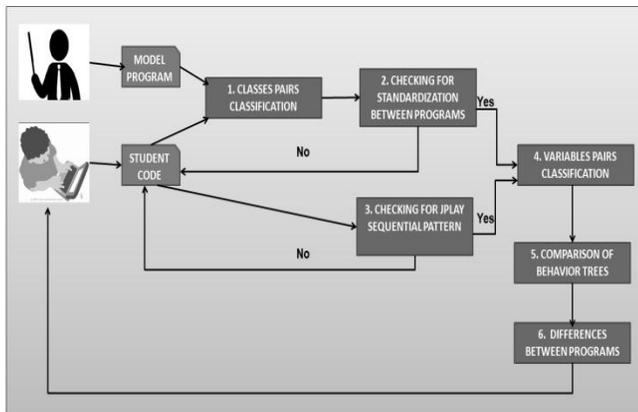


Figure 1. Stages of the Semantic Analyzer Process

In the case of the JPlay Sequential Pattern (3), if the code is correct (a sequence code pattern of the JPlay is a code sequence in the program based on JPlay that must always happen when the program is correct), the analyzer starts the process (4), otherwise the analyzer requests the student adaptations in the code and the process returns to the beginning. Thus, after process (4), the comparison of behavior trees of each variables pairs is performed (5) and the differences between the programs is returned to the student.

#### V. MODELING AND ANALYSIS OF PAIRS OF SIMILAR CLASSES BETWEEN PROGRAMS

Our proposal in [15] affirm that since Badros approach allows the preservation of the source code and our method needs a subsequent semantic analysis, we initially convert all classes from a developed program into a XML representation, based on the proposed JavaML method. Due the increase of tag’s representation from JavaML 2.0, we ignored this update. In order to semantically analyze a code under development, the programmer must select in a repository another program which will act as a model program for the comparison. The analysis consists in compare pairs of classes. Every class of the programmer code and from the program base will initially be transformed into XML by the parser. Each XML file will be read and interpreted by Java language using Document Object Model (DOM) [8].

After defining the pairs of classes, the comparisons between pairs are performed. The goal on this stage consists on identifying the pairs of classes that have higher similarity. The analysis of similarity between pairs of classes is based on rules previously established [15]. The matrix is first initialized with its similarity values, represented by the “weight” attribute, equal to 0 (zero) in all its elements. Other attributes are initialized with null.

There are specific rules to find the first and second pair. In order to classify the first pair of similar classes we establish as a single rule that the class contains the “main” method. When it finds two classes containing this method, the weight value is 2 and the “main” attribute value is true. After, in order to classify the second pair of similar classes we establish only one rule: if the class contains the game loop. When it finds two classes containing the game loop, the weight value is 2 and the “game loop” attribute value is true. The others attributes are calculated according to the rules defined in [15] in order to be performed more analysis of the student program.

After the first and second pair of similar classes be classified, the analyzer will take the rest of the classes of the program to be processed. This classification is performed in levels. First, the analyzer will search pairs of classes that extend from the same super class. In Fig. 2, for example, the “MyBall” class belongs to student program and the “Ball” class belongs to model program. During the classification the analyzer verifies that the “MyBall” and “Ball” classes extend from the same Sprite class. The analyzer verifies the list of variables of each one of the classes of the program that extend the same super class. In Fig. 3, for example, the “MyBall” and the “MyBar” class belong to the student program and extend Sprite class, the “Ball” and “Bar” class belong to model

program and also extend Sprite class. Then, all the combinations between these classes of the two programs will be analyzed with the goal of finding the pairs of classes more similar. In this step the combinations are performed by comparing the lists of variables of each class from the programs, as shown in Fig. 4. Each variable list contains the variable type and the number of variables of each type. The algorithm compares each pair of lists and calculates the difference between the values of variables of same type, after the similarity weight assigned the value of the sum of the results. Then, the pair having the lowest weight is rated as the most similar pair.

Generally the comparison between the lists of variables each class does not get full precision. Thus, the results may be close to reality, but not entirely correct. In order to obtain greater accuracy in the result, the algorithm performs a second comparison based on lists of behaviors. At the next comparison, the algorithm generates lists of behaviors containing the type of behavior and the value of that type of behavior. The algorithm compares each pair of lists again and then the pair containing the smallest weight is classified as most similar pair. At this level, all pairs of similar classes are defined, and the pairs classified should be compared.

At this point, with the aim of comparing the behavior of the pairs of classes, variables of the same type in each of the classes of the pair should be compared according to their behavior and then pairs of variables should be classified. The algorithm compares the variables of the same type using a list of behaviors and then the pair containing the smallest weight is rated as the most similar pair of variables.

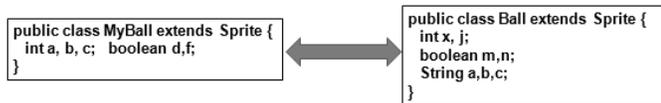


Figure 2. Example of MyBall and Ball classes code that extend the same Sprite super class

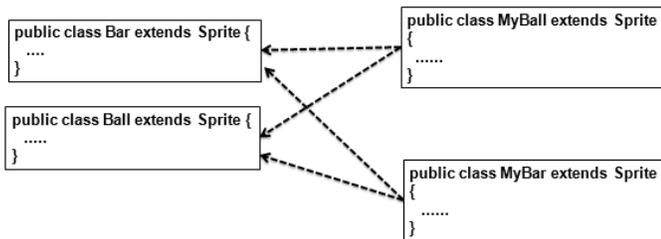


Figure 3. Example of classes from student program and model program that will be combined

MyBall			Ball	
Type	Number of variables of this type		Type	Number of variables of this type
Int	3	← 1	Int	2
Boolean	2	← 0	Boolean	2
		← 3	String	3
<b>Weight = 4</b>				

Figure 4. Example of the comparison between MyBall and Ball classes

After that, similar pairs of variables are defined and then analyzed. In order to compare these variables, we propose a tree structure, called behavior tree. This structure contains all the behaviors of a variable, which in our case we defined as “assignment”, “conditional” and “loop”. We build a behaviors tree to each variable, and the analyzer compares the behavior tree of a variable of a class that belongs to the student program with the behavior tree of its similar variable from model program. The behavioral trees of both the variables are compared, thus when behaviors differences occurs it is possible that a probable error of the student program occurs. In this point, when behaviors differences are identified in the trees, the analyzer makes suggestions about these behaviors differences found in the student program and clues that may indicate a possible semantic error. The suggestions are defined as comments in the source code of the program model. Each comment must be entered by the teacher before each behavior of variable, thus are automatically captured by our solution as suggestions to student.

An example of the comparison between variables behavior trees is shown in Fig. 5 and Fig. 6. The comparison between the behaviors trees of variables “b” and “d” show a difference in the while behavior. The analyzer must find the difference of the conditions between the programs. Finally, it must send suggestions to the student describing all the differences detected between the behaviors of the classes.

## VI. RESULTS

In this paper we propose a novel semantic analysis approach that checks a JAVA code and guide a student for a specific game development, giving clues of possible semantic failures, within a game oriented framework.

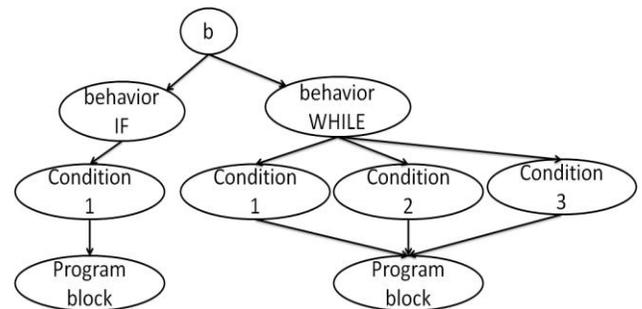


Figure 5. Example of the behavior tree of variable “b” in the model program

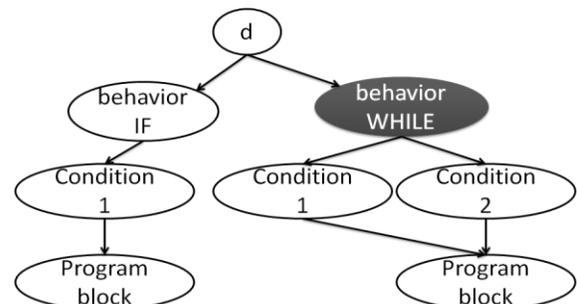


Figure 6. Example of the behavior of variable “d” in the student program

We developed a validation scenario in a classroom with 6 students of the integrated high school of the Informatics course. The students were proposed to develop a “BrickBreak” game, using JPlay framework. In this paper, we analyzed the “Ball” and “Bar” classes of the 6 codes developed, based on the model program developed by the teacher.

The results analysis presented here are related with two levels of analysis (according to Fig. 1): Checking for standardization between programs (2) and Comparison of the behavior trees (5). We evaluated the results according to behaviors expected for each class. The *Ball* class has two basic specifications: it must move in the x and y axis and collide with objects. The *Bar* class also has two specifications: it should move through the keyboard control, for both right and the left sides and collide with the *Ball*.

The results were marked as positive when the analyzer does not detect differences between the student program and the standardization model and negative when the analyzer detects differences between them. For the behavioral trees, we evaluated the results as false-positive when the analyzer does not detected differences, but the behavior of the object is not correct and false-negative when the analyzer detects differences but the behavior of the object is correct. It is considered, negative when the analyzer detects differences that really it exists and positive when the analyzer does not detect differences and the behavior of object is correct. Table I summarizes the results for the evaluation strategy for the *Ball* class example. Students 1 and 4 used one method more than requested in the standardization of model. The analyzer checks the difference between the students program and standardization model and prints the following suggestion for the student:

“You probably defined more methods than the necessary.”

About the comparison of behavior trees, there are differences in the program of the students 1, 3, 4, 5 and 6, according to Table I. The differences are defined as false-negative because these differences don’t modify the program behavior. Most of the differences in this class refer to assignment behaviors and are false-negative results.

TABLE I. TABLE EVALUATION FOR THE BALL CLASS

Student	Evaluation for the <i>Ball</i> class		
	Status program	Analysis according with the standardization between programs	Comparison of behavior trees
1	Incorrect (with the standard model)	Negative	False-negative
2	Correct	Positive	Positive
3	Correct	Positive	False-negative
4	Incorrect (with the standard model)	Negative	False-negative
5	Correct	Positive	False-negative
6	Correct	Positive	False-negative

Student 1 observations are related with the assignment behaviors of variables *left* and *right* from the model program and the suggestions that will be printed for him are:

- In the class constructor, initialize the attribute that controls the initial movement of the ball to the left or right;
- In the class constructor, initialize the attribute that controls the initial movement of the ball for up and down directions.
- Modify the movement of the ball if the ball position at the X axis is less than minimum limit of the game window and it is going to the left.
- Modify the movement of the ball if the ball position in the X axis is greater than maximum limit of the game window and it is going to the right. Consider this case the width of the ball. Example: maximum limit - width of the ball.
- Change the direction of the ball if the height of the ball in the y-axis is less than minimum limit of the game window and it is going to the left.
- Change the direction of the ball up if the height of the ball in the y-axis is greater than maximum limit of the game window and is going down.

The students 3, 5 and 6 show a similar situation to the student 1. For the student 4, the differences are related to the conditional behaviors of the variable “this.width” of the model program. The suggestion will be:

- Define the movement of the ball to the right side. Check which is the maximum right margin of the game window (maximum value of the x axis).

The results of the evaluation of the *Bar* class at the example is summarized by Table II:

TABLE II. TABLE EVALUATION FOR THE BAR CLASS

Student	Evaluation for <i>Bar</i> class		
	Status program	Analysis according with the standardization between programs	Comparison of behavior trees
1	Incorrect (with the standard model)	Negative	False-negative
2	Incorrect (with the standard model)	Negative	False-negative
3	Incorrect (with the standard model and with the behavior)	Negative	Negative
4	Incorrect (with the standard model)	Negative	False-negative
5	Incorrect (with the standard model)	Negative	False-negative
6	Incorrect (with the standard model)	Negative	False-negative

The analyzer did not find occurrence of the key words “Keyboard” and “keydown” in the programs of all the students, according Tab. II. The standardization model has defined JPlay objects and methods with these names with the purpose of being used to implement the movement of the bar through the keyboard control. Students 1, 2, 4, 5 and 6 implemented the keyboard control in another class program. Some of them even used another JPlay method (called `movex()`) to accomplish the same behavior, that led to the indication of the errors, although the behavior of the programs were correct. Then the analyzer prints the following comment for the student:

- *Please, check if you defined the movement of the bar through the keyboard control.*

In the analysis of the behavior trees of students 1, 2, 4, 5 and 6 there are differences in all the behaviors of the variables of the Bar class comparing with the model program, because the analyzer could not mount pairs of similar variables, thus all the comments associated will be printed as simple suggestions for the student.

In the special case of the student 3 (three), the program behavior is actually incorrect, because the bar is not moving properly. Thus, it is not possible compare variables, form pair of variables and compare the behaviors trees. Then the following comments are shown as suggestions:

- *“Define the movement of the bar to the right through the use of the keyboard using the `KeyDown()` method. Check which is, in the game window, the maximum value of the right margin.”*
- *“Increase the movement of the bar on the x axis, making the bar moves to the right.”*
- *“Define the movement of the bar to the left through the use of the keyboard using the `KeyDown()` method. Check which is, in the game window, the minimum value of the left margin.”*
- *“Decrement the movement of the bar on the x axis, making the bar move to the left.”*

## VII. CONCLUSION

This paper presents a novel heuristic strategy based in a analyzer that, interpreting semantically a JPlay code, guide a student for a specific game development process. Although our implementations and tests are related to JPlay framework, our proposal can easily be adapted to other program patterns.

The goals of the analyzer are to interpret semantically a Java program that uses JPlay and return results of this analysis to the student. Our proposal brings significant contributions to researchers working in the field of programming education and software engineering, specifically in relation to the knowledge modeling, having as main contributions the architecture for classification of similar classes and the definition of the data structure (behavior tree) starting from the behaviors of variables. Our paper also contributes in the sense that introduces a tool able to semantically interpret code built by students, returning results, pointing out problems and suggesting solutions.

The results of the comparison of behavior trees show that the analyzer is not able to be totally accurate, but is able to make a suggestion closer to the truth to the student. The results of analysis according with the standardization between programs are most accurate, how much more standardized the

student program, will be found less inaccuracy on comparison of the behavior trees.

As future work we intend to improve the efficiency of the algorithm and obtain more results comparisons. Also, as future work, we intend to develop a tutoring interface in order to manage the results received by the analyzer and the communication with the student.

## REFERENCES

- [1] Adam, A., Laurent, J., “LAURA, a system to debug student programs”. *Artificial Intelligence*, v.15, n.1, pp. 75-122, 1980.
- [2] Alexis, V.de A., Deller, J. F. “Aplicando Padrões de Seleção no Ensino de Programação de Computadores para Estudantes do Primeiro Ano do Ensino Médio Integrado”. In *X Encontro Anual de Computação – EnAComp*, 2013.
- [3] Allen, E., Cartwright, R. and Stoler, B. “Drjava: a lightweight pedagogic environment for java”. *SIGCSE Bull.*, 34(1):137-141, 2002.
- [4] Allowatt, A. and Edwards, S. “Ide support for test-driven development and automated grading in both java and c++”. In *eclipse ‘05: Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange*, pages 100-104, New York, NY, USA. ACM Press. 2005.
- [5] Barbosa, L. S., Fernandes, T.C.B., CAMPOS, A. M. C. “Takkou: Uma Ferramenta Proposta ao Ensino de Algoritmos”. In: *XXXI CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO - WEI XIX WORKSHOP SOBRE EDUCAÇÃO EM COMPUTAÇÃO*, 2011, Natal. *WEI XIX WORKSHOP SOBRE EDUCAÇÃO EM COMPUTAÇÃO*, 2011.
- [6] Botelho, C. A. “Sistemas Tutores no domínio da programação”. *Revista de Informática Aplicada/Journal of Applied Computing*, v.4, n. 1, 2010.
- [7] Delgado, K. V. “Diagnóstico baseado em modelos num sistema inteligente para programação com padrões pedagógicos”. Master’s dissertation, Institute of Mathematics and Statistics. 2005.
- [8] DOM, available in <http://www.w3.org/DOM/>. Accessed in November 2012.
- [9] Feijó, B., Clua, E., Da Silva, F.S.C. *Introdução à Ciência da Computação com Jogos: Aprendendo a Programar com Entretenimento*. Campos Elsevier. 1º ed. 2010.
- [10] Johnson, W. L., Soloway E.. “Proust: Knowledge-based program understanding”. In *ICSE 84: Proceedings of the 7th international conference on Software engineering*, pp. 369-380, Piscataway, NJ, USA, 1984. IEEE Press.
- [11] JPlay, available in <http://www.ic.ufr.br/jplay/>. Accessed in April 2012.
- [12] Kolling, M., Quig, B., Patterson, A., and Rosenberg, J. “The BlueJ system and its pedagogy”. *Journal of Computer Science Education*, Special issue on Learning and Teaching Object Technology, 13(4):249-268, 2003.
- [13] Pinheiro, W.R., Barros, L.N., Kon, F.. “AAAP: Ambiente de Apoio ao Aprendizado de Programação”. In *Workshop de Ambientes de Apoio à Aprendizagem de Algoritmos e Programação*, São Paulo, 2007.
- [14] Rapkiewicz, C. E. et al. “Estratégias pedagógicas no ensino de algoritmos e programação associadas ao uso de jogos educacionais”. *RENOTE*, v.4, n.2, 2006.
- [15] Santos, E.C.O., Batista, G.B., Clua, E.W.G. “A Knowledge Modeling System for Semantic Analysis of Games Applied to Programming Education”. In *SEKE 2013: Proceedings of the twenty-fifth International Conference on Software Engineering & Knowledge Engineering*, pp-668-673, Boston, June 27-29, 2013.
- [16] Santos, N.S.R.S., Rapkiewicz, C.E.. “Ensinando princípios básicos de programação utilizando jogos educativos em um programa de inclusão digital”. In: *SBGAMES - VI Simpósio Brasileiro de Jogos para Computador e Entretenimento Digital*, 2007, São Leopoldo - RS.
- [17] Traetteberg, H., Aalberg, T. “Jexercise: a specification-based and test-driven exercise support plugin for eclipse”. In *eclipse ‘06: Proceedings of 2006 OOPSLA workshop on eclipse technology eXchange*, pages 70-74, New York, NY, USA. ACM Press. 2006.

# Supporting Online Synchronous Education for Software Engineering via Web-based Operation Record and Replay

Dejian Chen, Yanchun Sun\*, Kui Wei, Zijian Qiao, Chao Xin

Institute of Software, School of Electronics Engineering & Computer Science, Peking University,  
Key laboratory of High Confidence Software Technologies, Ministry of Education,  
Beijing 100871, P.R.China

Email: {chendj12, sunyc, weikui13}@sei.pku.edu.cn, {qiaozijian, xincao}@pku.edu.cn

**Abstract**—Influenced by web 2.0 and cloud computing, web applications such as online modeling tools and web-based IDE develop rapidly. They are important for Software Engineering education because practice is crucial for students to get better understanding of the concepts introduced in class. However, most of these web applications are designed for individual usage, lacking support for real-time interactions. To solve this problem, we develop and demonstrate an Online Synchronous Education Plugin (OSEP), which is based on web-based operation record and replay. On one side, OSEP supports basic synchronous and interactive education on existing non-interactive web applications by high-fidelity record on the lecturer’s terminal and self-adaptive replay on the observer’s terminals. On the other side, OSEP ensures reliable, orderly synchronization and interaction by replay smoothing and latecomer controlling. In case study, we illustrate how OSEP is used in the real-time interactive education of online UML and web-based IDE teaching, which verifies the feasibility of the online synchronous education method for Software Engineering via web-based operation record and replay.

**Keywords**—Web applications; Software Engineering Education; Operation Record and Replay; Online Synchronous Education

## I. INTRODUCTION

With the development of web 2.0 and cloud computing, online education based on web browser develops rapidly. Software Engineering education also benefits from this trend, and more and more web-based platforms, tools and web applications are developed to aid the teaching process.

From 2012, MOOC (Massive Open Online Course) [1] has been increasing dramatically around the world such as edX, Coursera and Udacity. By connecting to the Internet with a browser, any student around the world can easily access resources on MOOCs, such as high-qualified video lectures, instant-feedback tests and interactive user forums. However, practice is a crucial factor in Software Engineering education,

and MOOCs are short of web-based supporting tools for interactive practice, such as interactive designing and coding.

With the development of HTML5 and CSS3, web applications are more and more complicated due to its graphical features. The concept “graphical” here has two implications: one directly refers to web graphics such as online UML modeling, and the other refers to Graphical User Interface (GUI) of web applications such as web-based IDE. These web applications are helpful for students to practice in Software Engineering course. However, most of them are designed for individual usage, so the lack of interaction makes them difficult to serve synchronous purpose. Re-development for interactive purpose is possible but costly.

From the introduction and analysis above, we can see that current online education platforms or web applications seldom support real-time interactive synchronization for education purpose. In this paper, we develop OSEP (Online Synchronous Education Plugin) to address this problem. Rather than re-development, OSEP adds synchronous feature into existing non-interactive web applications in an innovative way by recording each web-based operation of the instructor and replaying identical operation on the browsers of students in real time. Also, the instructor can authorize different students to display their operation to realize interaction among participants. Moreover, the whole process is recorded during the synchronization and it can be reviewed afterwards.

The rest of this paper is organized as follows. Section 2 introduces the related work. Section 3 demonstrates the architecture of OSEP. Section 4 presents the implementation of browser-side and server-side of OSEP. Section 5 introduces case study by two scenarios: online UML teaching and web-based IDE teaching. Section 6 presents concluding remarks and future work.

## II. RELATED WORK

Remote desktop technique, such as Virtual Network Computing (VNC) in the Linux environment and screen sharing system DisplayCast [2], enables a computer to share

---

\* Corresponding author

screen with or remotely control another computer over a network. Although remote desktop technique supports interactions in some degree by operating on the same computer among participants, it is not suitable for education purpose because: (a) students can only observe the process but not practice on them individually since the content cannot be copied automatically to their own computer; (b) latecomers may miss previous teaching process since they can only follow the screen of instructor in real time; (c) the process is not reusable since it can be only saved as a video, and if a student wants to reproduce the process and practice on the contents after class, he can only rebuild the contents on the browser step by step by watching the video.

Most of current collaborative web applications do not support particular graphical features in Software Engineering education. Google Docs [3] is a web-based office suite platform supporting collaborative editing on documents among different users in real time, but it does not support collaboration in complicated graphical operations. Shared white boards [4] support simple graphical drawing during brain-storm, but they are not suitable for more complex graphics and environments in education.

Our research team has put forward a method of operation record and replay technique and created an IDE-based interactive tutorial tool called SmartTutor [5, 6], which permits instructors and learners to edit tutorials on demand and create interactive tutorials. But it is based on Eclipse and does not support tutorials on web-based tools, so we further develop web-based operation record and replay techniques.

Web-based record and replay techniques are developed in some researches, but most of them are not suitable for online education purpose. CoScripser [7] is a system that allows users to capture, share, automate, and personalize business processes on the web. WaRR [8] is a tool that records and replays the interaction between users and modern web applications with high fidelity. However, these researches focus on form processing, such as buttons, check boxes and text fields, lacking support for graphical features such as capturing and automating of scalable vector graphics (SVG). Also, they do not support real-time interaction between learners in different places to finish a collaborative study.

Different from existing record and replay techniques, our approach features on “self-adaptive replay” and “real-time collaboration”. To the best of our knowledge, our approach is the first approach that supports online synchronous education for Software Engineering by using web-based record and replay technique.

### III. THE ARCHITECTURE OF OSEP

OSEP is composed of two modules: the graphical web operation record and replay module ensures that operations on graphical elements can be captured and reproduced during synchronization, and the interactive synchronization control module enables interactions among participants (See Fig. 1). Lecturer and observer are two kinds of roles in the system, where the lecturer is the one who gets the display authority and the observer is the one who observes the operations from the lecturer in real time. An instructor, initially acts as the

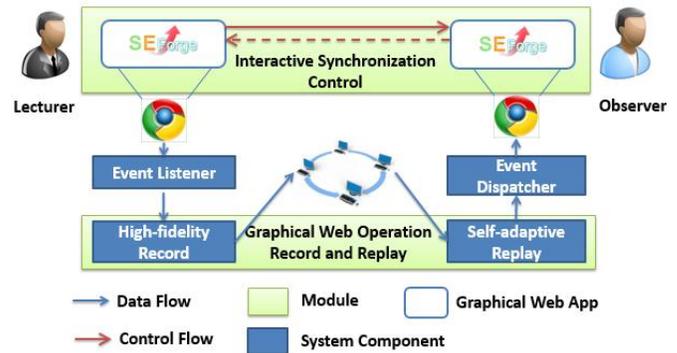


Figure 1. The Architecture of OSEP

lecturer, can designate different students as lecturers in different time to realize interactive education.

#### A. Graphical Web Operations Record and Replay

Since graphical web applications are more complicated than normal web pages due to moving, editing and resizing of web graphics and operations on GUI, more sophisticated record and replay technology is required.

##### 1) High-fidelity record

When recording users’ actions on normal web pages, we can only record crucial events such as “mouse click” and “keyboard input”, but for actions on graphical web pages, we need to capture more detailed events with high fidelity such as “mouse move” and “mouse over”. Raw events recorded, however, are abundant and non-semantic, so we need to encapsulate and optimize them.

There are three stages in high-fidelity record. (a) In capturing stage, raw events are captured by adding event listeners on browser. (b) In encapsulating stage, raw events are tailored to high-level operations, e.g. a combination of “mouse down”, “mouse move”, “mouse up” events on the same element can be encapsulated into a “drag and drop” operation. An operation can be described as  $\langle A, L, P \rangle$ , in which  $A$  means crucial attributes set,  $L$  means location sequence and  $P$  means operation type. In the previous example,  $A = \{a / a = \text{coordinate attributes of target element}\}$ ,  $L = [\text{DOM location sequence of target element}]$ ,  $P = \text{“drag and drop”}$ . (c) In optimizing stage, we filter redundant operations such as continuous duplicated “mouse move” events on the same target element.

##### 2) Self-adaptive replay

For reusable purpose, recorded operations should be replayed on different terminals and at different time, and this feature is termed as “self-adaptive”. Due to heterogeneous environments of different terminals, the replay should adapt to the differences of screen size, resources loading speed and even versions of browsers. Due to different time points of record and replay, the website may be updated and the replay should adapt to insignificant changes of the website.

There are three stages in self-adaptive replay. (a) In positioning stage, target elements are selected from the DOM tree according to crucial attributes set  $A$  and location sequence  $L$  from  $\langle A, L, P \rangle$ . (b) In resolving stage, high-level operations are reverted back into browser-level events

according to operation type  $P$  from  $\langle A, L, P \rangle$ . (c) In executing stage, appropriate event dispatch APIs are chosen to execute browser events according to current version of the browser.

To ensure “self-adaptive” feature, target elements are selected based on similarity metric aiming to adjust changes of website. The similarity is measured by two factors: content similarity and structural similarity. The process can be described as the following four steps.

a) *Filter candidate elements*

Given a recorded element description  $X = \langle A, L \rangle$ , element tag is obtained from attributes set  $A$ , and we select elements with identical tag from DOM tree as candidate elements. For each candidate element  $Y$ , it can be described as  $Y = \langle B, M \rangle$ , where  $B$  and  $M$  denote attributes set and location sequence respectively.

b) *Calculate content similarity score*

Crucial attributes set  $A$  of recorded element  $X$  can be described as a vector:  $A = (a_1, a_2, \dots, a_n)$ , where  $a_i (i=1,2,\dots,n)$  is an attribute of element. Also, for each candidate  $Y$ , its corresponding attribute vector is  $B = (b_1, b_2, \dots, b_n)$ . To calculate similarity score of  $A$  and  $B$ , we can use Minkowski distance to describe the difference, and then convert it to similarity. With dimension coefficient of 1, the distance is defined as

$$d(A, B) = \frac{1}{n} (\sum_{i=1}^n |a_i - b_i|)$$

The distance  $d(A, B)$  is standardized, if each of the attribute pair distance  $|a_i - b_i|$  is standardized with a range of  $[0, 1]$ . Three types of attributes are involved: nominal type, numeric type and string type.

For nominal type, the value is enumerable in DOM, such as “id”. Specially, a Boolean type attribute lies in this category whose value limits to “true” of “false”. The handling is straightforward that the attribute pair distance is 1 if both values are the same or 0 if they are different.

For numeric type, the value is a number with unlimited range. Distance can be measured by their absolute value of their difference, but it should be standardized within range  $[0, 1]$  by exponential or reciprocal conversion.

For string type, the value is an array of characters. To calculate the distance between two strings, we use Levenshtein distance (or editing distance) as measurement [9]. It is defined as the minimum number of single-character edits (insertions, deletions or substitutions) required to change one string into the other. With dynamic programming, it can be implemented with time complexity of  $O(n^2)$ . Also, to standardize the result, we divide the distance by the larger length of the strings to limit the range to  $[0, 1]$ .

Table 1 summarizes the standardized distance of three attribute types introduced above. After distance is computed for each attribute pair, we can get content distance of  $X, Y$ , and their content similarity:

$$Content-Similarity(X, Y) = 1 - d(A, B)$$

c) *Calculate structural similarity score*

TABLE 1. STANDARDIZED DISTANCE OF THREE ATTRIBUTE TYPES

Attribute Type	Nominal	Numeric	String
Value	Enumerable	$(-\infty, +\infty)$	Array of characters
Examples	id, checked, readonly	clientX, height	href, src, value, innerText
Attribute Pair Distance $d(a_i, b_i)$	$\begin{cases} 1 (a_i=b_i) \\ 0 (a_i \neq b_i) \end{cases}$	$e^{- a_i-b_i }$ OR $\frac{1}{1+ a_i-b_i }$	$\frac{LevDist(a_i, b_i)}{MaxLength(a_i, b_i)}$
Range	$[0, 1]$		

If the website updates in replay stage, some structural changes of DOM tree may occur. For example, in web-based IDE, some new projects may have been established after the record stage, so the project list in replay stage may be different. In this case, it is probable that the sub-tree structure of the selected project shifts from one node to another node, and the location sequence of target element should not change dramatically.

We calculate the structural similarity by measuring how two location sequences are alike, or namely the similarity of  $L$  and  $M$  from  $X$  and  $Y$ . Similar to content similarity, Levenshtein distance is used as the measurement. However, it differs that we are comparing DOM tree nodes rather than string characters. The key here is that how to judge whether two nodes are identical. Although we can use the content similarity model as above, it is time-consuming when DOM tree are large. Under insignificant changes assumption, we compare only tag names, node indices and the most important attributes such as “id” and “type”. In practice, it is much faster and the performance is very close to the content similarity model.

The structural similarity is calculated as:

$$Structural-Similarity(X, Y) = 1 - \frac{LevDist(L, M)}{MaxLength(L, M)}$$

d) *Select the most similar element*

For each candidate element  $Y$ , calculate content-similarity( $X, Y$ ) and structural-similarity( $X, Y$ ), which are denoted as  $C$  and  $S$  respectively. Since tag names of  $X, Y$  are the same, we can allocate specific weights  $W_C$  and  $W_S$  according to different tags and calculate the weight-average similarity score:

$$Similarity(X, Y) = W_C * C + W_S * S$$

Elements such as  $\langle input \rangle, \langle a \rangle, \langle img \rangle$  are more identifiable by their attributes, thus we allocate more weights on  $C$  than  $S$  ( $W_C > W_S$ ). Likewise, elements such as  $\langle div \rangle, \langle span \rangle, \langle td \rangle$  distinguish from others mainly based on their structural features, so we allocate more weights on  $S$  than  $C$  ( $W_C < W_S$ ). The candidate element with the highest score is regarded as the identical one to the recorded element.

3) *Applicable scope*

The technology of graphical web operation record and replay is applicable to websites with “manifest” DOM structure such as SVG (Scalable Vector Graphics). If a website

is developed using flash or HTML5 canvas whose internal structure is not manifested as DOM, the technology is unsuitable due to lack of adaptability among different terminals. Besides, the replay is “self-adaptive” only if the changes of websites are insignificant. If a website has changed dramatically since recorded, i.e. the layout and content both update to a brand new version, it is necessary for users to start a new record stage on this “new” website.

### B. Interactive Synchronization Control

In the process of interactive teaching, synchronous control mechanism is required to ensure that normal participants as well as latecomers can take part in the online class regardless of heterogeneous network environment.

Two issues are crucial to interactive synchronization control: server-push and disconnection.

Server-push techniques [10], such as polling, Comet, Server-sent event or HTML5 WebSocket, are required to ensure timely synchronization. There are some open-source frameworks that support these techniques, such as Pushlet for Servlet and Socket.IO for Node.js.

Disconnection occurs when networks or systems are unstable. Although it may be partially solved by some server-push frameworks using the store-and-forward strategy, some problems are unsolved in the particular education scenario. Based on graphical web operation record and replay techniques, two related disconnection issues must be addressed.

#### 1) Replay Smoothing

Ideally, whenever an instructor acts on the website, the operation is recorded on the record-side, broadcast via the server and replayed on the browsers of the replay-side in real time. However, delay or disconnection may block the broadcast and when it resumes, the recorded operations crowd in the replay-side. Without data buffering and replay speed control, problems of disordered replaying, unclear displaying or other unexpected errors (such as resources loading speed is slow in replay-side, and operations are executed before resources are loaded) may occur. Therefore, we should adjust differences in server-push speed and replay speed, and the process is termed as “replay smoothing”.

The problem is abstracted similar to the classical “producer-consumer” model in the Operating System field, where the producer is network-receiving module, the consumer is replay-executing module, and products are operations. However, there are four differences in our case: (a) operations (products) are ordinal and should be replayed in correct order; (b) network-receiving module (producer) should confirm the operations and send back requests if operations are missing; (c) the buffer is unlimited in size since JavaScript objects are extendable; (d) most importantly, producer and consumer cannot block each other in context of multithreading because JavaScript is non-blocking.

Although HTML5 WebWorker enables thread features in JavaScript, it is still in development and some browsers may not support it. Therefore, we develop a modified “Producer-Consumer” model under non-blocking and event-driven

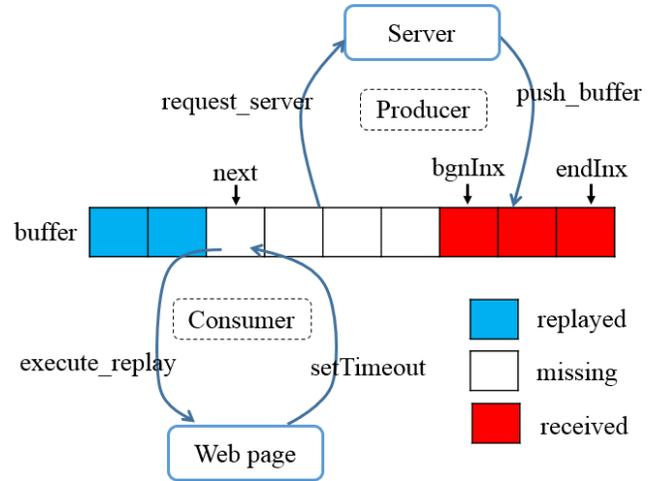


Figure 2. Schematic Diagram of Modified “Producer-Consumer” Model

assumptions of JavaScript. The producer and the consumer are independent, and they communicate by global variables. Global variables include *next*, denoting the index of next operation to be replayed, *buffer* of operations array, and *interval*, denoting polling interval of consumer which is introduced in the following.

The receiving module (producer) is driven by server-push events. When new operations are received, they are stored in buffer. If *buffer[next]* is still empty, a request is sent for fetching the missing operations due to disconnection or delay.

The replaying module (consumer) is driven by “setTimeout” events, where “setTimeout” is a mechanism provided by JavaScript that a callback function is triggered after a given time interval on non-blocking basis. By triggering itself, the function is actually under a polling process. To increasing efficiency, in practice, we start by setting the polling *interval* as 100 millisecond and scales up to a maximum of 1000 millisecond if there are no new operations. Otherwise, it resets to 100 millisecond and starts the polling process again.

The schematic diagram of modified “producer-consumer” model is shown as Fig. 2, and the algorithm is shown as algorithm 1.

---

#### Algorithm 1: Modified “Producer-Consumer” Model

```

// Global Variables
next ← 0
buffer [] ← ∅
interval ← 100 ms
// Producer procedure
PROCEDURE onReceive (opt, bgnIdx, endIdx)
  INPUT: Received operations opt, Beginning index bgnIdx
  and Ending index endIdx
  put_buffer(opt, bgnIdx, endIdx)
  IF buffer[next] is empty AND next < bgnIdx
    THEN request_server(next, bgnIdx -1)
// Consumer procedure
PROCEDURE tryReplay()
  IF Page is ready AND buffer[next] is not empty
    THEN execute_replay(buffer[next])
    next ← next +1

```

```

interval ← 100 ms
ELSE interval ← min ( interval *2, 1000 ms);
setTimeout ( tryReplay, interval )

```

## 2) Latecomer Problem

As mentioned in [11, 12], a latecomer can be treated as an observer that has been disconnected since the beginning of the session, and it can be handled by store-and-forward strategy. Two modes can be chosen by the latecomer to catch up with current progress: the fast replay mode enables latecomers to review all historical operations in an accelerated speed, while the content replication mode skips non-crucial operations by replicating content directly in forms of DOM sub-tree in order to catch up the teaching process as soon as possible. An algorithm of HTML tree-matching based on editing distance is introduced in [13], which is adopted to address the sub-tree matching issue. At present, the function “Latecomer control” is still under construction.

## IV. IMPLEMENTATION

### A. Browser Side

The browser side of OSEP is developed as a Chrome Extension (<http://developer.chrome.com/extensions/>), which allows developers to add some functions into Chrome without diving deeply into native code. The OSEP extension is composed of two parts: content-script and console. The content-script part, embedded in the original Chrome window, captures raw events in record stage and plays back reverted events in replay stage. The console part, shown as a separate window, provides user interface and connects to server. Both parts are implemented in JavaScript and jQuery (<http://jquery.com/>), and they communicate with each other by message channel APIs of chrome extension.

### B. Server Side

The server side of OSEP is developed as a HTTP server built on Node.js (<http://nodejs.org/>), a platform built on Chrome’s JavaScript runtime for building fast, scalable network applications. The event-driven, non-blocking I/O features makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices. To implement server-push and client-pull techniques, we use Socket.IO (<http://socket.io/>) which aims to make real-time web apps possible, blurring the differences between the different transport mechanisms. As built-in audio chatting tool is also required for better communication purpose, we use WebRTC (<http://www.webrtc.org/>) to support Real-Time Communications (RTC) capabilities on web browser via simple JavaScript APIs.

## V. CASE STUDY

In case study, we will present how OSEP is used to implement synchronous education in Software Engineering class. GUI of OSEP is shown in the left part of Fig. 3 and figure 3. Students can participate in synchronous education with teachers in the “Online Classes” page, access and load historical recorded classes in the “Resources” page, and replay the loaded class and reproduce teaching process in the “Scripts” page.

Imagine that a teacher is giving an online course to students in different branch schools of the same university, as well as students around the world. When confronted with some complicated and practical problems in software engineering class, the teacher may use OSEP to interact with students online. Two scenarios of software engineering teaching are illustrated as follows.

### A. Online UML Teaching

When teaching object-oriented method, the teacher may feel more straightforward and distinct if he uses online UML applications to display the modeling process. He chooses GWT UML (<https://code.google.com/p/gwtuml/>), an open source online UML modeling tool on Google Code. As GWT UML is implemented in HTML5 SVG techniques, OSEP is capable of supporting the synchronization on it.

The teacher starts by creating a class diagram, creates and renames several classes, and then adds relationships among them (See Fig. 3). While the teacher is operating on the graphics, students can observe identical graphical status by automatically replaying the same operations on their browsers. Some students are late when they join the class, and they can choose to synchronize the teaching process with: the fast replay mode where teachers operations are recurred step by step; or the content replication mode where the teacher’s current graphics are sent and replaced in the terminals of latecomers.

If the teacher wants to interact with students, he can pause and authorize a student to play the role of “lecturer”, and the student’s operations will be synchronized and displayed on other terminals simultaneously. Also, the teacher can designate another student to operate and display on the existing model to support interaction. Finally, when the synchronous teaching ends, a student can continue to practice on the online UML modeling application, which is not supported by remote desktop techniques.

### B. Web-based IDE Teaching

When teaching Computer-Aided Software Engineering (CASE), the teacher may use more concrete examples to illustrate the advantages of using IDE to develop software. He chooses Cloud 9 (<https://c9.io/>), one of the most popular free

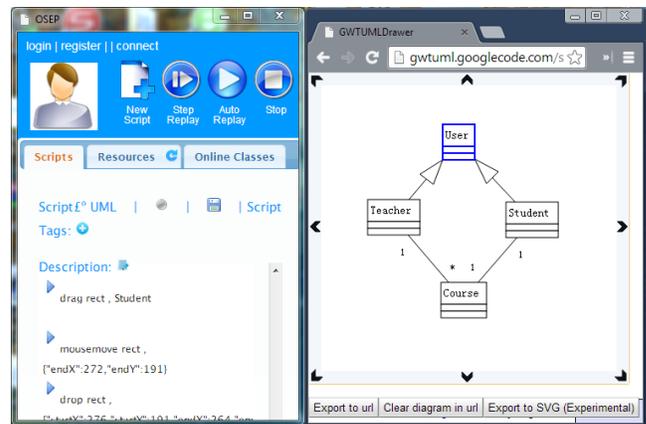
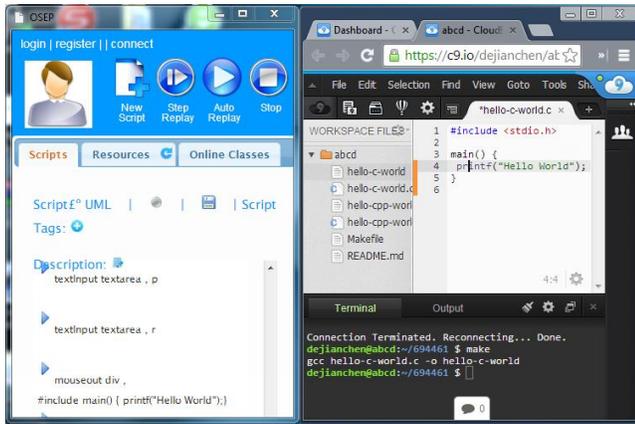


Figure 3. Online UML Teaching Scenario (As shown in both teacher-side and student-side)



**Figure 4. Web-based IDE Teaching Scenario  
(As shown in both teacher-side and student-side)**

online IDEs. Although cloud9 itself supports real-time collaborative editing in the editor among users just like Google Docs, the operations on other GUI elements such as menus and menu items cannot be synchronized and displayed. One of the similar work focusing on collaboration in a web IDE is introduced in [14], but it is customized and not extendable. Instead, OSEP enables the record and replay of GUI elements, and is suitable for education.

The teacher starts by showing how to develop a project using various tools on the IDE by simply operating on GUI. In the process of configuring, coding, compiling, testing and running the software, some operations can be complicated and time consuming. When the teacher clicks the menu, opens a new dialogue, edits parameters, submits the modification, creates a new file, right-clicks, chooses “rename”, edits new name and so on, students can observe all these operations so that they can learn how to develop software step by step (See Fig. 4).

When the teaching process is finishing, the teacher can save the recorded operations as a tutorial and share it to students as after-class study material. Also, students can use the tutorial to configure a new project automatically by replaying operations on GUI, which is not supported by remote desktop.

## VI. CONCLUSION

We can conclude that OSEP supports synchronous and interactive education for Software Engineering on graphical web applications, which verifies the feasibility of the synchronous education method based on graphical web operation record and replay. In future work, we would further improve latecomer control to make OSEP more suitable for usage of synchronous education.

## ACKNOWLEDGMENTS

This effort is sponsored by the National Basic Research Program of China (973) under Grant No. 2011CB302604, the Joint Fund of the National Natural Science Foundation of China under Grant No. U1201252, and the Science Fund for Creative Research Groups of China under Grant No. 61121063.

## REFERENCES

- [1] Mehran Sahami, Mark Guzdial, Fred G. Martin, Nick Parlante. The revolution will be televised: perspectives on massive open online education. In *Proceeding of the 44th ACM technical symposium on Computer science education*. 2013:457-458
- [2] Surendar Chandra, John Boreczky, Lawrence A. Rowe. High performance many-to-many intranet screen sharing with DisplayCast. In *ACM Transactions on Multimedia Computing, Communications, and Applications*. Vol.10, No.2, 2014
- [3] Dan R. Herrick. Google this!: using Google apps for collaboration and productivity. In *Proceedings of the 37th annual ACM SIGUCCS fall conference*. 2009:55-64
- [4] Stacy Branham, Gene Golovchinsky, Scott Carter, Jacob T. Biehl. Let's Go from the Whiteboard: Supporting Transitions in Work through Whiteboard Capture and Reuse. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '10)*. 2010:75-84
- [5] Ying Zhang, Gang Huang, Nuyun Zhang, Hong Mei. SmartTutor: Create IDE-based Interactive Tutorials via Editable Replay. In *Proceedings of the 31st International Conference on Software Engineering (ICSE '09), Formal Research Demo Track, 2009*, pp:559-562
- [6] Nuyun ZHANG, Gang HUANG, Ying ZHANG, Hong MEI. Towards Automated Synthesis of Executable Eclipse Tutorial. In *Proceedings of the 22nd International Conference on Software Engineering and Knowledge Engineering (SEKE '10)*, 2010, pp:591-598
- [7] Gilly Leshed, Eben M. Haber, Tara Matthews, Tessa Lau. CoScripter: automating & sharing how-to knowledge in the enterprise. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '08)*, 2008, pp:1719-1728
- [8] S Andrica, G Candea. WaRR: A Tool for High-Fidelity Web Application Record and Replay. In *proceedings of Ieee/Ifip 41St International Conference on Dependable Systems and Networks (DSN'11)*, 2011, pp. 403-410.
- [9] Liu Yujian, Liu Bo. A Normalized Levenshtein Distance Metric. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol.29, No.6, 2007, pp:1091-1095
- [10] Mikko Pohja. Server push with instant messaging. In *Proceedings of the 2009 ACM symposium on Applied Computing*. 2009, pp: 653-658
- [11] Carl Gutwin, T. C. Nicholas Graham, Chris Wolfe, Nelson Wong, Brian de Alwis. Gone but not forgotten: designing for disconnection in synchronous groupware. In *Proceedings of the 2010 ACM Conference on Computer Supported Cooperative Work (CSCW '10)*. 2010, pp:179-188
- [12] Banani Roy, T.C. Nicholas Graham, and Carl Gutwin. DiscoTech: a plug-in toolkit to improve handling of disconnection and reconnection in real-time groupware. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work (CSCW '12)*. 2012, pp:1287-1296
- [13] Yeonjung Kim. Web Information Extraction by HTML Tree Edit Distance Matching. In *Proceedings of the 2007 International Conference on Convergence Information Technology (ICCIT '07)*. 2007, pp:2455-2460
- [14] Max Goldman, Greg Little, and Robert C. Miller. Real-Time Collaborative Coding in a Web IDE. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*. 2011, pp:155-164

# A DIMENSIONALITY REDUCTION PROCESS TO FORECAST EVENTS THROUGH STOCHASTIC MODELS

Joaquim Assunção, Paulo Fernandes, Lucelene Lopes, Silvio Normey

PUCRS University – Computer Science Department – Porto Alegre, Brazil  
{joaquim.assuncao, paulo.fernandes, lucelene.lopes, silvio.gomez}@pucrs.br

## Abstract

This paper describes a dimensionality reduction process to forecast time series events using stochastic models. As well as the KDD process defines a sequence of common steps to achieve useful information through data mining techniques, we propose a sequence of steps in order to estimate the probability of future events through stochastic modeling. Our process focus on reduce the dimensionality of data, thus reducing the effect of the common problems involved in stochastic modeling, such as the state space explosion and the large modeling efforts to create such models.

## 1 Introduction

Stochastic modeling can be a natural option to predict system behavior. This system can be a process that represents basic situations or even complex nature events. Either way, a good model can deliver interesting probabilities about system events for its past, or even its future.

However, modeling is a non trivial task that requires a specialist on the domain that have a good knowledge about the scenario to be modeled. Other problem frequently faced, with stochastic models, is the *state space explosion* that causes a limit of states, thus, a limit of data to be modeled.

The amount of data in the world, in our lives, seems to go on and on increasing [32], and as the volumes of data increase, it increases the difficulty of handling these data. One of the problems most frequently faced is the difficult to detect information among such large amounts of data. These informations are lost because it is hard for a human being to see the patterns and relations among such abundant data.

To cope with this problem, specialized data mining algorithms retrieve information that are practically invisible to our perception. Although the information is revealed, it is not yet knowledge, since knowledge must be obtained by rationally processing the extracted information. Aiming to formalize the steps commonly used to reach the knowledge through data mining, there are some well defined processes in the literature, as for example the knowledge discovery in databases (KDD) described by Han *et al.* [17]. Although this process is well consolidated and often used as a guide for knowledge discovery, instead of describing the path to discover knowledge, it seems to have been created to cover the steps that usually come before and after data mining.

In the context of our paper we focus on a specific case of KDD, which is the forecast of future events in time series databases. Therefore, we propose the use of stochastic modeling techniques as steps of the proposed process, information about events is a common trade in stochastic models. Specifically, we simulate and analyze a formal state-based model of the system in order to retrieve probabilities for future events (and states) of the targeted system.

In a rough comparison. as well as KDD shows a set of steps commonly used to prepare data for data mining, here we describe a process that shows useful steps to prepare data for stochastic modeling. These steps aim to decrease the time spent to develop models, coping with state space problem and reducing the chance of human mistakes while manually developing complex models.

This paper is organized as follows: the next sections describe this paper background: Time series and Stochastic modeling formalisms; The fourth section puts this work in perspective with pre-existent techniques; The fifth section describes the proposed process; Section 6 illustrates the proposed process with some numerical result obtained applying it to pre-existent datasets; Finally, the conclusion summarizes our contribution and suggests future works.

## 2 Time Series

A popular method to represent and analyze data is time series (TS) [8, 21, 31]. Given the ease of use, modify, compare and analyze data; TS became very popular in many fields of sciences such as statistics, economy, biology, geography, seismology, engineering, communication, machine learning, *etc.* A TS is characterized as a collection of data spread in time. Formally, a TS  $S$  can be defined as follows:

$$(2.1) \quad S = [(p_1, t_1), (p_2, t_2), \dots, (p_k, t_k), \dots, (p_n, t_n)]$$

where  $(t_1 < t_2 < \dots < t_k < \dots < t_n)$

In this representation,  $p_i$  is the data corresponding to the  $i$ -th observation, and  $t_i$  is the time when  $p_i$  was observed. In such way, a TS can be visualized in two axis, one with the data representing  $p_i$ , and other with the time passage from

$t_1$  to  $t_n$ . Despite of that, since the data associated to each observation can be related to a complex information, TS are essentially high dimensional data [17]. In fact, TS can be employed to describe very complex structures like fossil shapes, molecules, geological materials, *etc.* [34, 18].

In order to enhance the ease of storage, to facilitate human analysis, and to be flexible to many domains, TS had became a global trending for representation of data. Consequently, the constant use of TS in different areas and the growth of machine learning, brought researchers to new challenges related to TS.

We can highlight such challenges according to the main goals of TS analysis: modeling and forecasting [17]. In modeling, researchers try to improve efficiency by reducing the impact of the dimensionality factor and by improving measurements techniques. In forecasting, researchers try to predict future situations, *e.g.*, weather forecast, traffic behavior, *etc.*

One of the big problems related to TS is how to improve modeling efficiency being TS generally a data streaming [31]. Aiming to solve this problem, in the last decade, many methods to represent TS have been created [25, 8, 20].

### 3 Stochastic Modeling

Stochastic modeling is the art of representing the behavior of a system by describing its states and transitions. One of the most common among such formalisms are the Markov chains. Given an initial state, even a simple discrete-time Markov chain can be used to forecast future states by achieving probabilities for each possible state for the next  $n$  steps [29]. These steps are given by the Chapman-Kolmogorov equation. Being  $P$  a probability matrix:

$$(3.2) \quad P_{ij}^{(m)} = \sum_{all\ k} P_{ik}^{(l)} P_{kj}^{(m-l)} \quad for\ 0 < l < m$$

Being a classic formalism, Markov chains are easy to handle and use to perform quantitative analysis. Unfortunately large volumes of data tend to generate huge models that are hard to handle using an ordinary Markov chain (state space explosion). In such cases, the use of a structured formalism as Stochastic Automata Networks (SAN) [24, 7] may reduce this problem impact.

SAN is a formalism that describes a complete system as a collection of subsystems that interact with each other in a Markovian behavior. The main advantage of SAN is that we can represent very large and complex system in a human understandable model that has an equivalent Markov chain model. Figure 1 exemplifies a SAN model with two automata and its equivalent Markov chain.

SAN formalism uses synchronized events and functional rates to build the system. Once a model is ready, with few automata it can be equivalent to a giant Markov chain. In

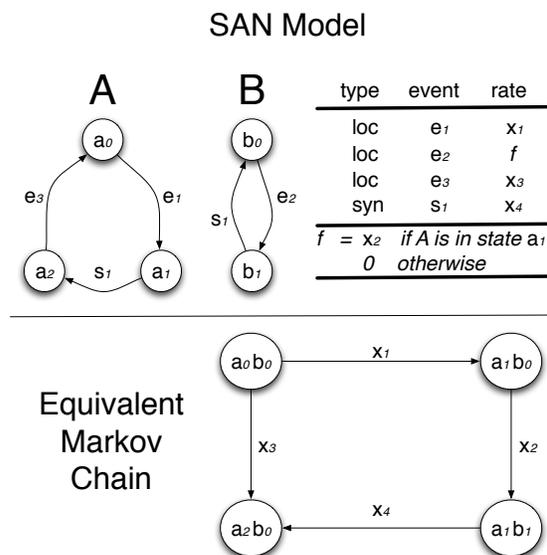


Figure 1: SAN model example with 2 automata and its equivalent Markov Chain.

addition, the tensor format of SAN makes possible the use of specialized algorithms [15, 10, 11] which are more efficient than those used for solving large Markov chains [10, 11]. A complete SAN model is represented by a SAN code that can be executed [6] to generate a set of probabilities, including those which may be generated in a Markov chain.

Recently we successfully created a SAN model to represent and predict classes in geological events [1]; a different approach that returns results much similar to a data mining classification. Due to the processes of analysis and data collection, this model took much time to be created. However, even high dimensional data can be transformed into TS, and then, into a string that can be transformed both into a SAN model or into a Markov chain.

### 4 Dimensionality Reduction and Forecast

Both research areas, TS and stochastic modeling, have an interesting thing in common; they both have many applications that are useful for knowledge discovery. The use of TS for data representation and forecasting is presented in a myriad of works [9, 26, 23, 22, 4]. Despite such abundant material, is also frequent to find the use of Markovian models to predict states in complex scenarios [13, 35, 28, 5, 16, 1, 14]. In this work, we use these both techniques trying to profit from the knowledge on TS modeling and stochastic prediction through Markovian models.

Using stochastic modeling techniques, we can retrieve accurate probabilities about a given system or event. However, the “state space explosion” makes it hard to scale models to real application size problems [30, 29]. In TS

there is a similar phenomenon, the well-known “curse of dimensionality”, which is the subject of many works in the area [25, 8, 21].

Actually, these two problems are different views of a same phenomenon, since the background limitation is the fact that it is easy to describe a large number of points of view (several dimensions), but the reality that needs to be handled is the Cartesian product of the data from all this points of view, *i.e.*, we model focusing on the parts, but the model must be handle as a whole. Among the approaches to tackle this problem, we are interested in this paper in dimensionality reduction [21] and structured modeling formalisms [7].

Several techniques have been developed to represent TS in order to reduce the difficulty of processing high dimensional data. Here, we are interested in techniques that have two characteristic: 1) the representation must be symbolic; 2) the representation must be flexible in length. Among this techniques, for experiment purposes, we use Symbolic Aggregate approXimation (SAX) [21].

SAX is a solution to reduce the dimensionality of a TS both in length (it reduces the number  $n$  of observations) and in data space (it aggregates the values of  $p_i$ ). Also, it generates symbolic data that can be used to generate stochastic models on an automated way. The next section presents the process, and the basic phases to obtain the stochastic model, and then, the data forecasting through its transient and steady states analysis.

## 5 Proposed Process

Instead of work directly in mathematical models to slight improve some stochastic formalism, we focus on a process that reduces the data dimensionality before these data goes into the model. Instead of use a limited set of algorithms to forecasting in TS, we focus on adapt these data to work with advanced stochastic algorithms.

In order to achieve it, we carefully select the most useful techniques to represent and reduce the dimensionality of a given dataset, as well as, powerful methods that allow us to perform quantitative analysis under the generated data. Thus, this process aims to achieve a set of useful probabilities directly from a dataset using the minimum human effort and minimizing the effects of “state space explosion”.

Figure 2 shows a simplified flow diagram of the proposed process. Each picture represented at the bottom part of this figure is a set of data; and each term written at the top part of this figure is a task of the process. Next, we describe each step of the process, which is basically composed by a task and its resultant data .

**5.1 Data Selection** Given a database, the first step begins with Data Selection. The Data Selection described here is very similar to the data mining *Data Selection* step; the data

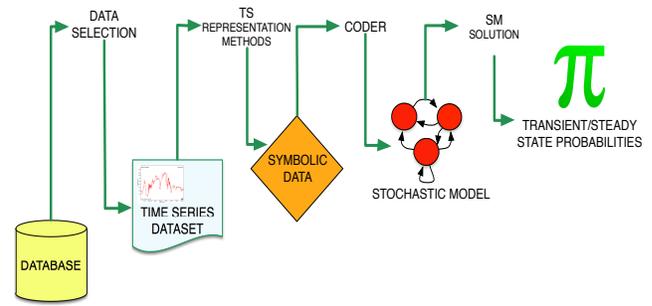


Figure 2: Process Phases

must be analyzed aiming to retrieve the desired TS. After this step, a set of TS is achieved.

**5.2 TS Representation** This step performs the dimensionality reduction of the dataset. One by one, the TS goes to SAX which performs a dimensionality reduction using Piecewise Aggregate Approximation [8]. Then, SAX generates the symbolic data.

**5.3 Coder** In big data, coding is the process of tagging terms using an identifier that is assigned to every synonymous term in the data [3]. This step has a similar objective; although, instead of grouping terms, symbolic data are grouped into states. Thus, this step is the connection between TS output and the solution of the SM. In other words, Coder is the step responsible to get the symbolic data generated by TS representation and transform it into a SM. As seen in the Figure 3, the symbolic data is a collection of  $N$  symbols that represents  $N$  levels of a given TS; thus, represented by:  $S = [a, b, c, d, \dots, \Omega]$  the symbolic data generated ( $D$ ) can be formalized as:  $D_i = \{s \in S\}$ .

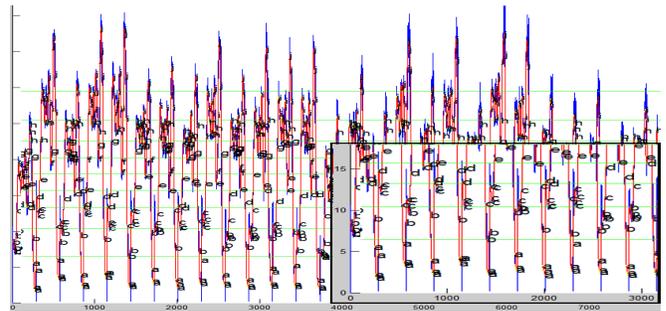


Figure 3: Classes and symbolic data generated for Coffee dataset, an example of TS dataset.

Once we have the symbolic data, it is possible to automatically create the SMs. Basically it is performed by getting the frequencies of changes for each class in a symbolic data.

Given three hypothetically TS,  $Q, S$  and  $T$  and its representing symbolic data  $w(Q), w(S)$  and  $w(T)$ , each one composed by classes from  $a$  to  $j$ . The coder calculates the frequency of changes for each possible combination, e.g.:

$$\begin{aligned} a \rightarrow b, a \rightarrow c, \dots, a \rightarrow j \\ b \rightarrow a, b \rightarrow c, \dots, b \rightarrow j \\ \dots, \dots, \dots, \dots \\ j \rightarrow a, j \rightarrow b, \dots, j \rightarrow j \end{aligned}$$

Each frequency is turned into a percentage value that is used as a transition rate for the automata. Also, each set of symbolic data turns in one automaton. In this scenario, we will have a SAN composed by three automata created from  $w(Q), w(S), w(T)$ . Having each one 10 states, e.g.,  $a, b, \dots, j$ , our SAN model will be limited to 1000 reachable states, which is almost instantaneously to solve with current SAN algorithms.

**5.4 Stochastic model solution** In this step, the code generated is used to create and solve a stochastic model. These stochastic models can be either, Markov chains, SAN or any other Markovian formalism. We develop a tool to make automatically create Markovian models from a symbolic data; furthermore, there is a work in progress to automatically generate SAN code [2]. Once that we have a SAN model, SAN solvers like PEPS [6] and SAN Lite-Solver [27] can deliver steady state and transient solutions through specialized algorithms [15, 10, 11].

Despite the high complexity of the stochastic model solution algorithms, it is possible to understand the way those solutions deliver their results considering the following simple example. Let us consider a stochastic model with states  $A, B$  and  $C$ . Given the probabilities:

	A	B	C
A. to	0.3	0.1	0.4
B. to	0.2	0.8	0.4
C. to	0.5	0.1	0.2

A transient solution can be derived applying Chapman-Kolmogorov Equation [29] we shall achieve the following probabilities to be in each state for the next step:

	A	B	C
A. to	0.31	0.42	0.27
B. to	0.15	0.70	0.15
C. to	0.24	0.48	0.28

Finally the steady state solution is computed considering an infinite number of steps. In our simple example it results in the following values:

	A	B	C
A. to	0.2	0.6	0.2
B. to	0.2	0.6	0.2
C. to	0.2	0.6	0.2

## 6 Experiments and Results

The main goal of the experiments is to measure model accuracy. Thus, aiming to avoid data bias, we chose different kinds of public datasets to test our process. A sample of four synthetic datasets were randomly picked from [19]: *Coffe*, *Symbols*, *WordsSynonyms*, and *ItalyPwrDemand*. A sample of four, real world, economy datasets were collected from Brazilian market activity stock [33]: *petr3*, *bbas3*, *brfs3*, and *vale3*. *Paris Temp* was collected from [12]; as a real world dataset, it represents the Paris monthly temperature from Le Bourge station. Finally, for a more controllable scenario, we made a synthetic dataset that describes a sound-wave reducing its amplitude over time: *WaveDS*.

For each dataset, we reserved 80% to training, and 20% to test the output. The probabilities is given by two basic analysis: transient states and steady state. We use 42 transient states to predict next values; we also use 10 symbols for each dataset and a variable value to the dimensionality reduction. This is specially useful because many datasets have a high density of values, in the TS Representation step, many near points in the  $X$  axis for a light change in the  $Y$  axis.

Considering the high impact of external factors, we need to create a controllable experiment aiming to check if the model is generating coherent results. For such task, we had to set an input that we already had an expected output. This dataset, called *WaveDS*, construction is depicted in Figure 4.

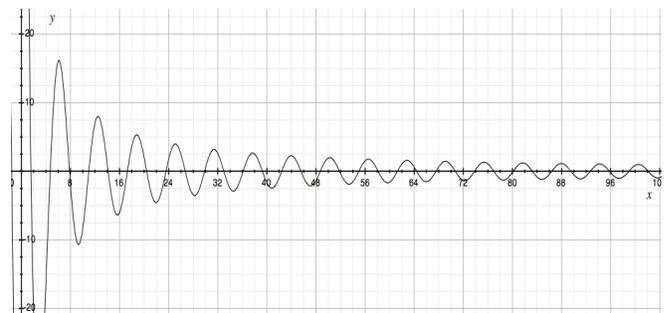


Figure 4: Synthetic dataset, *WaveDS*.

Clearly, we can see that tending to infinity the value of the wave should be 0. The expected flow for this experiment is the sequence:

1. SAX reads the TS and returns the corresponding symbolic data, being “a” the class close to  $-20$  and “j” the class close to  $20$ .
2. The Coder reads the symbolic data, calculate the probabilities and creates the stochastic model.
3. The model must return a very high steady state percentage chance for the classes close to 0.

The three steps had worked as expected; the experiment returns a high percentage values for the steady state in classes “e” and “f” that represents values close to zero, 48.63% and 48.03%, respectively. This experiment shows that the expected sequence was performed and the process has worked as expected.

The same approach was performed for all other datasets. For the synthetic datasets the output match with the expected classes, although 2 in market datasets the expected class was the second and third with higher percentages. This was expected, because there are many fluctuations on the market datasets that are very difficult to predict using only the historical data.

From all datasets tested, we divided nearly 80% to run in our process and 20% to test the output. For the datasets *Symbols* and *WordsSynonyms*, we had no significant changes, *i.e.*, the steady state for all classes was achieved in the very beginning, and the value became close to the original. In *Paris Temp* dataset, we found two high scores for non adjacent classes, *i.e.*, classes that cannot be agglutinated into one because they are non contiguous. Although, for the other datasets our quantitative analysis was accurate with the test part of the dataset.

All experiments results are show in Table 1. In this table is indicated the name of the dataset, the probabilities of the more frequent classes, and a general indication if the experiment prediction was accurate ("✓" indicates an experiment with a correct prediction, *i.e.*, the predicted class, "†" indicates otherwise).

Dataset	Predicted probabilities	Accuracy
<i>Coffee</i>	j: 94.28%	✓
<i>Symbols</i>	f: 12.51%; g: 11.94%	✓
<i>WordsSynonyms</i>	i: 29.51%; j: 29.01%	✓
<i>ItalyPowerDemand</i>	i: 40.21%; j: 42.54%	✓
<i>petr3</i>	b: 18.68 %	✓
<i>bbas3</i>	b: 45.37%	✓
<i>brfs3</i>	b: 55.80%	✓
<i>vale3</i>	a: 37.30%	†
<i>Paris Temp</i>	c: 21.55%; i: 25.08%	†
<i>WaveDS</i>	e: 48.63%, f: 48.03%	✓

Table 1: Probabilities and classes found for each dataset ("✓" indicates accurate and "†" indicates inaccurate).

For dataset *Paris Temp* the problem seems related to the bimodal characteristic of the data, since two non adjacent classes (c and i) were the more frequent ones. Dataset *vale3*, however, seems to be a more complex case and it deserves a deeper analysis in the future.

## 7 Conclusion

We successfully created a process that drives us into a next step to forecast data with high dimensionally. Our process is capable of making stochastic predictions through Markovian models coping with the curse of dimensionality. Our tests shows that the new process created is useful to reduce the data dimensionality and perform forecasting through stochastic models. Also connecting the best techniques in the literature we were able to automate the steps that usually demands time and effort from specialists.

More than forecasting, this process provides a new way to perform data mining tasks in high dimensional data. Now we are able to use Markovian models to knowledge discovery with a large amount of data, due to the curse of dimensionality it was infeasible before. This brings us to new challenges, once that we have a large amount of algorithms to solve problems using stochastic models.

In the implementation front, the next step is to generate the SAN code through the *Coder*, thus we can save time performing multiples executions due to the SAN format and the large variety of the algorithms already implemented in the SAN tools [6] [27]. This certainly will bring us more performance to make more tests and use the model into new scenarios, such as big data.

Other possible application is the real time prediction for streaming data. A dimensionality reduction can be applied online generating SAN code to be solved in real time, thus predicting next states.

## References

- [1] J. ASSUNÇÃO, L. ESPINDOLA, P. FERNANDES, M. PIVEL, AND A. SALES, *A structured stochastic model for prediction of geological stratal stacking patterns*, in 6th Practical Applications of Stochastic Modelling (PASM'12), London, UK, September 2012, pp. 1–15.
- [2] J. ASSUNÇÃO, P. FERNANDES, T. FISCHER, AND A. SALES, *Unsupervised model generation for geological events*, 2014. Accepted to 46th Annual Simulation Symposium.
- [3] J. BERMAN, *Principles of Big Data: Preparing, Sharing, and Analyzing Complex Information*, Elsevier Science, 2013.
- [4] G. E. BOX, G. M. JENKINS, AND G. C. REINSEL, *Time series analysis: forecasting and control*, Wiley. com, 2013.
- [5] L. BRENNER, P. FERNANDES, J.-M. FOURNEAU, AND B. PLATEAU, *Modelling Grid5000 point availability with SAN*, Electronic Notes in Theoretical Computer Science (ENTCS), 232 (2009), pp. 165–178.
- [6] L. BRENNER, P. FERNANDES, B. PLATEAU, AND I. SBEITY, *PEPS2007 - Stochastic Automata Networks Software Tool*, in Proceedings of the 4th International Conference on Quantitative Evaluation of SysTems (QEST 2007), Edinburgh, UK, September 2007, IEEE Computer Society, pp. 163–164.

- [7] L. BRENNER, P. FERNANDES, AND A. SALES, *The Need for and the Advantages of Generalized Tensor Algebra for Kronecker Structured Representations*, International Journal of Simulation: Systems, Science & Technology (IJSIM), 6 (2005), pp. 52–60.
- [8] K. CHAKRABARTI, E. KEOGH, S. MEHROTRA, AND M. PAZZANI, *Locally adaptive dimensionality reduction for indexing large time series databases*, ACM Trans. Database Syst., 27 (2002), pp. 188–228.
- [9] C. CHATFIELD, *Time-series forecasting*, Chapman and Hall/CRC, 2002.
- [10] R. M. CZEKSTER, P. FERNANDES, J.-M. VINCENT, AND T. WEBBER, *Split: a flexible and efficient algorithm to vector-descriptor product*, in International Conference on Performance Evaluation Methodologies and tools (ValueTools'07), vol. 321 of ACM International Conferences Proceedings Series, ACM Press, 2007, pp. 83–95.
- [11] R. M. CZEKSTER, P. FERNANDES, AND T. WEBBER, *Efficient vector-descriptor product exploiting time-memory trade-offs*, ACM SIGMETRICS Performance Evaluation Review, 39 (2011), pp. 2–9. doi: 10.1145/2160803.2160805.
- [12] DATAMARKET!, *The open portal to thousands of datasets from leading global providers*. <http://datamarket.com/>, 2013.
- [13] C. ENGEL, *Can the markov switching model forecast exchange rates?*, Journal of International Economics, 36 (1994), pp. 151–165.
- [14] P. FERNANDES, M. O'KELLY, C. PAPADOPOULOS, AND A. SALES, *Analysis of exponential reliable production lines using kronecker descriptors*, Int. Journal of Production Research, 51 (2013), pp. 2511–2528.
- [15] P. FERNANDES, B. PLATEAU, AND W. J. STEWART, *Efficient descriptor-vector multiplication in Stochastic Automata Networks*, Journal of the ACM, 45 (1998), pp. 381–414.
- [16] P. FERNANDES, A. SALES, A. R. SANTOS, AND T. WEBBER, *Performance evaluation of software development teams: a practical case study*, Electronic Notes in Theoretical Computer Science, 275 (2011), pp. 73 – 92.
- [17] J. HAN, M. KAMBER, AND J. PEI, *Data Mining, Second Edition: Concepts and Techniques*, The Morgan Kaufmann Series in Data Management Systems, Elsevier Science, 2006.
- [18] E. KEOGH, L. WEI, X. XI, S.-H. LEE, AND M. VLACHOS, *Lb\_keogh supports exact indexing of shapes under rotation invariance with arbitrary representations and distance measures*, in Proceedings of the 32nd international conference on Very large data bases, VLDB '06, VLDB Endowment, 2006, pp. 882–893.
- [19] E. KEOGH, Q. ZHU, B. HU, Y. HAO, X. XI, L. WEI, AND RATANAMAHATANA, *The UCR Time Series Classification/Clustering Homepage*. [www.cs.ucr.edu/~eamonn/time\\_series\\_data/](http://www.cs.ucr.edu/~eamonn/time_series_data/), 2011.
- [20] 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.
- [21] 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.
- [22] Z. LINJESSICA, Y. HUANG, H. WANG, AND S. MCCLEAN, *Neighborhood counting for financial time series forecasting*, in Evolutionary Computation, 2009. CEC '09. IEEE Congress on, 2009, pp. 815–821.
- [23] J. PERALTA, G. GUTIERREZ, AND A. SANCHIS, *Shuffle design to improve time series forecasting accuracy*, in Evolutionary Computation, 2009. CEC '09. IEEE Congress on, 2009, pp. 741–748.
- [24] B. PLATEAU, *On the stochastic structure of parallelism and synchronization models for distributed algorithms*, ACM SIGMETRICS Performance Evaluation Review, 13 (1985), pp. 147–154.
- [25] K. PONG CHAN AND A. W.-C. FU, *Efficient time series matching by wavelets*, in Proceedings of the 15th International Conference on Data Engineering, M. Kitsuregawa, M. P. Papazoglou, and C. Pu, eds., ICDE '99, Washington, DC, USA, 1999, IEEE Computer Society, pp. 126–133.
- [26] R. REYHANI AND A.-M. MOGHADAM, *A heuristic method for forecasting chaotic time series based on economic variables*, in Digital Information Management (ICDIM), 2011 Sixth International Conference on, 2011, pp. 300–304.
- [27] A. SALES, *San lite-solver: a user-friendly software tool to solve san models*, in Proceedings of the 2012 Symposium on Theory of Modeling and Simulation - DEVS Integrative M&S Symposium, TMS/DEVS '12, San Diego, CA, USA, 2012, Society for Computer Simulation International, pp. 44:9–16.
- [28] T. STEFFENS AND P. HUGELMEYER, *Real-time prediction in a stochastic domain via similarity-based data-mining*, in Simulation Conference, 2007 Winter, 2007, pp. 1430–1435.
- [29] W. J. STEWART, *Probability, Markov Chains, Queues, and Simulation*, Princeton University Press, USA, 2009.
- [30] A. VALMARI, *The state explosion problem*, in Lectures on Petri Nets I: Basic Models, W. Reisig and G. Rozenberg, eds., vol. 1491 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 1998, pp. 429–528.
- [31] X. WANG, A. MUEEN, H. DING, G. TRAJCEVSKI, P. SCHEUERMANN, AND E. KEOGH, *Experimental comparison of representation methods and distance measures for time series data*, Data Mining and Knowledge Discovery, 26 (2013), pp. 275–309.
- [32] I. H. WITTEN, E. FRANK, AND M. A. HALL, *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition*, The Morgan Kaufmann Series in Data Management Systems, Elsevier Science, 2011.
- [33] YAHOO!, *Brazilian market stock indices*. <http://br.financas.yahoo.com/indices?e=bovespa>, 2013.
- [34] L. YE AND E. KEOGH, *Time series shapelets: a new primitive for data mining*, in Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, KDD '09, New York, NY, USA, 2009, ACM, pp. 947–956.
- [35] G. YU, J. HU, C. ZHANG, L. ZHUANG, AND J. SONG, *Short-term traffic flow forecasting based on markov chain model*, in Intelligent Vehicles Symposium, 2003. Proceedings. IEEE, 2003, pp. 208–212.

# Choosing the Best Classification Performance Metric for Wrapper-based Software Metric Selection for Defect Prediction

Huanjing Wang  
Western Kentucky University  
huanjing.wang@wku.edu

Taghi M. Khoshgoftaar  
Florida Atlantic University  
khoshgof@fau.edu

Amri Napolitano  
Florida Atlantic University  
amrifau@gmail.com

**Abstract**—Software metrics and fault data are collected during the software development cycle. A typical software defect prediction model is trained using this collected data. Therefore the quality and characteristics of the underlying software metrics play an important role in the efficacy of the prediction model. However, superfluous software metrics often exist. Identifying a small subset of metrics becomes an essential task before building defect prediction models. Wrapper-based feature (software metric) subset selection uses a classifier to discover which feature subsets are most useful. To the best of our knowledge, no previous work has examined how the choice of performance metric within wrapper-based feature selection will affect classification performance. In this paper, we used five wrapper-based feature selection methods to remove irrelevant and redundant features. These five wrappers vary based on the choice of performance metric (Overall Accuracy (OA), Area Under ROC (Receiver Operating Characteristic) Curve (AUC), Area Under the Precision-Recall Curve (PRC), Best Geometric Mean (BGM), and Best Arithmetic Mean (BAM)) used in the model evaluation process. The models are trained using the logistic regression learner both inside and outside wrappers. The case study is based on software metrics and defect data collected from a real world software project. The results demonstrate that BAM is the best performance metric used within the wrapper. Moreover, comparing to models built with full datasets, the performances of defect prediction models can be improved when metric subsets are selected through a wrapper subset selector.

## I. INTRODUCTION

Software metrics collected are often associated with defects found during pre-release as well as post-release. Software defect prediction models are built based on these software metrics and fault data. Subsequently, the quality of currently under-development program modules is estimated, e.g., fault-prone (*fp*) or not-fault-prone (*nfp*). The models help to reduce software development effort and produce highly reliable software. Software defect prediction has been a high-focus area of research in the software engineering community [1], [2], [3], [4]. Defect predictors are often used by software quality practitioners in guiding them to intelligently allocate limited project resources toward program modules that are likely to have poor quality and reliability. A practical challenge faced by these practitioners is the selection of the right software metrics prior to building a defect predictor. Selecting the right static code metrics prior to building a defect prediction model has many benefits, including removing noisy attributes, reducing the set of software metrics to use, and perhaps even improving defect prediction performance [5]. Feature (software metric) selection algorithms (which select a subset of features from the original dataset) are often used to reduce the original feature set down to a subset containing only the most important features.

In this paper, we evaluate the wrapper-based feature subset selection method: classification models are built with logistic regression (LR) and feature subsets are selected according to their predictive capability, which are measured by a performance metric. We used

five different performance metrics: Overall Accuracy (OA), Area Under ROC (Receiver Operating Characteristic) Curve (AUC), Area Under the Precision-Recall Curve (PRC), Best Geometric Mean (BGM), and Best Arithmetic Mean (BAM). In total, we had 5 subset evaluators each based on the LR learner and using one of the five performance metrics. We assessed these wrapper-based feature subset selection methods by building classification models with the selected feature subsets and evaluating them with the five performance metrics (OA, AUC, PRC, BGA, BAM). When building a defect prediction classifier, we used ten runs of five-fold cross-validation, along with five-fold cross-validation within the wrapper process.

The experiments of this study were carried out on nine datasets from a real world project. These datasets are further divided to three groups with different levels of imbalance. There are three datasets in each group. We compared model performance on the smaller subsets selected by the wrapper-based feature subset selector. We also compared the classification performance on the smaller subsets of attributes with those on the original dataset. Results demonstrate that when BAM is used inside the wrapper, models built with the selected feature subset have the best performance, while PRC gives the worst performance.

The key contributions of this research are:

- Implementation and investigation of the wrapper-based feature subset selection technique. Although five performance metrics and one learner are used in the study, other performance metrics and learners can be used.
- The use of imbalanced data from real-world software systems. Such an extensive range of wrapper-based feature selection along with imbalanced data for software quality prediction is unique to this study.

The rest of the paper is organized as follows. We review relevant literature on feature selection in Section II. Section III provides detailed information about the wrapper-based feature selection, performance metrics, learner, and cross-validation methods used in our study. Section IV provides a description of the software measurement datasets used and presents empirical results of our study. Finally, in Section V, the conclusion is presented and the suggestions for future work are indicated.

## II. RELATED WORK

Feature (attribute) selection is an important preprocessing step in the area of data mining and machine learning. Researchers and practitioners are interested in which feature subset is important in the model building process when high-dimensional datasets are considered. Specifically, feature selection seeks to reduce the number of features by targeting an optimum subset of features and removing

the rest of the features from further consideration during subsequent analysis. The idea is that the features that are not being considered are either irrelevant to the problem at hand or redundant when compared to the features within the optimum subset. Guyon and Elisseeff [6] outlined key approaches used for attribute selection, including feature construction, feature ranking, multivariate feature selection, efficient search methods, and feature validity assessment methods. Hall and Holmes [7] investigated six attribute selection techniques that produce ranked lists of attributes and applied them to several datasets from the UCI machine learning repository. Liu and Yu [8] provided a comprehensive survey of feature selection algorithms and presented an integrated approach to intelligent feature selection.

Feature selection techniques can be separated into two categories: filter and wrapper. Filters are algorithms in which a feature subset is selected without involving any learning algorithm. Wrappers are algorithms that use feedback from a learning algorithm to determine which feature(s) to include in building a classification model. Wrapper methods differ from filter-based feature selection in that they use a learner when evaluating the features, either separately or as subsets. In this study we focused on wrapper-based feature selection. Typically, wrapper-based feature selection uses subset evaluation, which looks at the possible subsets of features and tests them for their ability to differentiate between the classes. It should be noted that in order to test each possible subset, one would have to perform  $2^n - 1$  tests (there is no point in testing an empty set) where  $n$  is the number of features. Search algorithms can reduce this space significantly, they do not resolve the problem of necessitating many evaluations, however.

Very limited research exists on wrapper-based feature subset selection in the software quality and reliability engineering domain. Chen et. al. [9] have studied the applications of wrapper-based feature selection in the context of software cost/effort estimation. They conclude that the reduced dataset improved the estimation. Rodríguez et al. [5] applied attribute selection with three filter models and two wrapper models to five software engineering datasets. It was stated that the wrapper model was better than the filter model; however, that came at a very high computational cost. Their conclusions were based on evaluating models using cross-validation. Their work is very limited since the same performance metric was used both inside and outside the wrapper. Although they performed ten runs of ten-fold cross validation for final classification, when performing wrapper feature selection (within the cross-validation process), the goodness of the wrapper learners was evaluated against the exact same data used to train them. In our work, we evaluate five wrapper-based subset evaluators, with these wrappers differing based on the choice of performance metric used in the model evaluation process inside the wrapper. We used five-fold cross-validation to build the learners within the wrapper, *in addition to* ten runs of five-fold cross-validation used to evaluate our overall classification models: within every run and fold of external (classification) cross-validation, we perform separate folds of internal cross-validation to evaluate subsets of features inside the wrapper. We also evaluate our final classification models using the five different performance metrics, and our choice of datasets enables us to study three different levels of class imbalance. An extensive comparative study of feature subset selection techniques is very unique to this paper, especially within the software engineering community.

### III. METHODOLOGY

In this study, we focus on wrapper-based feature subset selection techniques and apply these techniques to software engineering

datasets. Five performance metrics are used to evaluate the models built during the wrapper selection process giving a total of five different wrappers. The final classification models are also evaluated using these same five performance metrics. Logistic Regression is used to build classification models both inside and outside the wrapper.

#### A. Classifier

Classification is used to build a model that can accurately classify instances. The goal is to build a model which minimizes the number of classification errors. The first step of classification is to build a classification model that can describe the predetermined set of data classes, and in the second step, the classification model is evaluated using an independent test dataset. In this study, the software quality prediction models are built with logistic regression [10] both inside and outside the wrapper. Logistic Regression (LR) [10] is a statistical technique that can be used to solve binary classification problems. Based on training data, a logistic regression model is created which is used to decide the class membership of future instances. LR was selected because of its common use in software engineering and other application domains, and also because it does not have a built-in feature selection capability. We use default parameter settings for the learner as specified in WEKA [11].

#### B. Wrapper-based Feature Selection

Filter-based feature selection techniques have been studied extensively in the past [12]. By comparison, very little research has been focused on wrapper-based feature selection. The wrapper-based feature selection methods employ some predetermined learning algorithms (classifiers or learners) to evaluate the goodness of the subset of features being selected. The performance of this approach relies on three factors: (1) the strategy to search the feature space for possible optimal feature subsets; (2) the learner; and (3) the criterion to evaluate the classification model built with the selected subset of features.

Suppose a large set of  $n$  features is given, we need to find a small subset of features for future model building. Inspecting all candidate subsets ( $2^n$ ) is impractical. There are some strategies that can solve the problem. One way is to use a search algorithm to generate the possible feature subsets. Based on preliminary experimentation, we chose the Greedy Stepwise approach, which uses forward selection to build the full feature subset starting from the empty set. At each point in the process, the algorithm creates a new family of potential feature subsets by adding every feature (one at a time) to the current best-known set. The merit of all these sets are evaluated, and whichever performs best is the new known-best set. The wrapper procedure terminates when none of the new sets outperform the previous known-best set.

During the search process, classification models are built using a potential feature subset and using the performance of this model as a score for the merit of that subset [13]. For our experiments the wrapper process uses five-fold cross-validation: the training set is divided into five equal folds (partitions), a classifier is trained on four folds, then tested on the last (fifth) fold. This process is repeated five times, and the results are averaged to give the merit of the potential feature subset. In this study, logistic regression is used within the wrapper-based feature subset selector. The classification model is assessed on the performance of the model based on five different performance metrics (Overall Accuracy (OA), Area Under ROC (Receiver Operating Characteristic) Curve (AUC), Area Under the Precision-Recall Curve (PRC), Best Geometric Mean (BGM),

and Best Arithmetic Mean (BAM)). All five performance metrics associated with each potential feature subset are obtained.

### C. Performance Metrics

In a two-group classification problem, such as fault-prone and not fault-prone, there can be four possible prediction outcomes: true positive (TP), false positive (FP), true negative (TN) and false negative (FN), where positive represents fault-prone and negative represents not-fault-prone. The numbers of cases from the four sets (outcomes) form the basis for several other performance measures that are well known and commonly used for classifier evaluation. We used five of them for the wrapper-based feature subset selection in this study. They are:

- 1) Overall Accuracy (OA): It provides a single value range from 0 to 1. It can be obtained by  $\frac{|TP|+|TN|}{N}$ , where N is the total number of instances in the dataset.
- 2) Area Under ROC (Receiver Operating Characteristic) Curve (AUC): It has been widely used to measure classification model performance [14]. AUC is a single-value measurement that ranges from 0 to 1. The ROC curve is used to characterize the trade-off between true positive rate (defined as  $\frac{|TP|}{|TP|+|FN|}$ ) and false positive rate (defined as  $\frac{|FP|}{|FP|+|TN|}$ ). A classifier that provides a large area under the curve is preferable over a classifier with a smaller area under the curve. A perfect classifier provides an AUC that equals 1. It has been shown that AUC has lower variance and is more reliable than other performance metrics (such as precision, recall, or F-measure) [15].
- 3) Area Under the Precision-Recall Curve (PRC): It is a single-value measure that originated from the area of information retrieval. Precision and Recall are two widely used performance metrics in the context of classification tasks. The closer the area is to one, the stronger the predictive power of the attribute [16]. The PRC graph depicts the trade off between recall and precision. A classifier that is near optimal in AUC space may not be optimal in precision/recall space.
- 4) Best Geometric Mean (BGM): The Geometric Mean (GM) is a single-value performance measure that ranges from 0 to 1, and a perfect classifier provides a value of 1. GM is defined as the square root of the product of true positive rate and true negative rate (defined as  $\sqrt{\frac{|TN|}{|FP|+|TN|}}$ ). It is a useful performance measure since it is inclined to maximize the true positive rate and the true negative rate while keeping them relatively balanced. Such error rates are often preferred, depending on the misclassification costs and the application domain. The Best Geometric Mean (BGM) is the maximum Geometric Mean value that is obtained when varying the threshold between 0 and 1 [17].
- 5) Best Arithmetic Mean (BAM): The Arithmetic Mean is just like geometric mean but using the arithmetic mean of the true positive rate and true negative rate instead of the geometric mean. It is also a single-value performance measure that ranges from 0 to 1. The *Best Arithmetic Mean* (BAM) is just like the BGM, but using the maximum arithmetic mean that is obtained when varying the threshold between 0 and 1, instead.

### D. Cross-Validation

In the experiments, we use ten runs of five-fold cross-validation to build and test the final classification models. That is, the datasets are partitioned into five folds, where four folds are used to find the appropriate feature subset and train the classification model, and the model (using the chosen feature subset) is evaluated on the fifth. This

TABLE I  
SOFTWARE DATASETS CHARACTERISTICS

Project	Data	#Metrics	#Modules	%fp	%nfp
Eclipse-I	E2.0-10	208	377	6.1%	93.9%
	E2.1-5	208	434	7.83%	92.17%
	E3.0-10	208	661	6.2%	93.8%
Eclipse-II	E2.0-5	208	377	13.79%	86.21%
	E2.1-3	208	434	11.52%	88.48%
	E3.0-5	208	661	14.83%	85.17%
Eclipse-III	E2.0-3	208	377	26.79%	73.21%
	E2.1-2	208	434	28.8%	71.2%
	E3.0-3	208	661	23.75%	76.25%

is repeated five times so that each fold is used as hold out data once. In addition, we perform ten independent repetitions (runs) of each experiment to remove any bias that may occur during the random selection process. In total, 50 training subsamples are generated for each dataset. Note, feature selection is performed inside of this cross-validation procedure. That means, wrapper feature selection is applied to each of the training subsamples. Furthermore, the one run of five-fold cross-validation discussed in Section III-B is in addition to these ten runs of five-fold cross-validation: within every fold of external (classification) cross-validation, we perform separate folds of internal (wrapper) cross-validation, to evaluate the quality of our feature subsets. In addition to the models built during the wrapper process, a total of 2700 (9 datasets  $\times$  5 wrappers  $\times$  10 runs  $\times$  5 folds cross-validation + 9 datasets  $\times$  10 runs  $\times$  5 folds cross-validation) final classification models are trained and evaluated during the course of our experiments. The classification results reported in the next section represent the average across these ten runs of five-fold cross-validation.

## IV. EXPERIMENTS

### A. Experimental Datasets

The software metrics and fault data for this case study were collected from a real-world software project, the Eclipse project [18]. We consider three releases of the Eclipse system, where the releases are denoted as 2.0, 2.1, and 3.0. In particular, we use the metrics and defects data at the software package level. We transform the original data by: (1) removing all nonnumeric attributes, including the package names, and (2) converting the post-release defects attribute to a binary class attribute: fault-prone (*fp*) and not fault-prone (*nfp*). Membership in each class is determined by a post-release defects threshold  $t$ , which separates *fp* from *nfp* packages by classifying packages with  $t$  or more post-release defects as *fp* and the remaining as *nfp*. In our study, we use  $t \in \{10, 5, 3\}$  for release 2.0 and 3.0, while we use  $t \in \{5, 4, 2\}$  for release 2.1. These values are selected in order to have datasets with different levels of class imbalance. All nine derived datasets contain 208 independent attributes (software metrics). Releases 2.0, 2.1, and 3.0 contain 377, 434, and 661 instances respectively. A different set of thresholds is chosen for release 2.1 because we wanted to maintain relatively similar class distributions for the three datasets in a given group. Table I presents key details about the different datasets used in our study. The three groups of datasets exhibit different class distributions with respect to the *fp* and *nfp* modules, where Eclipse-I is relatively the most imbalanced and Eclipse-III is the least imbalanced.

### B. Experimental Design

We first used five wrapper-based feature subset selectors (LR + one of five performance metrics) to select the subsets of attributes.

TABLE II  
PERFORMANCE SUMMARIZATION FOR ECLIPSE-I

Wrapper Metric	Classifier Metric									
	OA		AUC		PRC		BGM		BAM	
	Mean	Stdev	Mean	Stdev	Mean	Stdev	Mean	Stdev	Mean	Stdev
OA	<i>0.9352</i>	0.0055	0.8388	0.0278	<i>0.4467</i>	0.0485	<i>0.7915</i>	0.0244	<i>0.7951</i>	0.0228
AUC	0.9368	0.0063	<i>0.8274</i>	0.0410	0.4746	0.0560	0.8054	0.0331	0.8086	0.0311
PRC	0.9355	0.0065	0.8374	0.0408	0.4561	0.0472	0.8033	0.0328	0.8078	0.0296
BGM	0.9392	0.0066	0.8471	0.0343	0.4970	0.0503	0.8192	0.0211	0.8221	0.0202
BAM	<b>0.9395</b>	0.0071	<b>0.8528</b>	0.0315	<b>0.5018</b>	0.0433	<b>0.8209</b>	0.0232	<b>0.8234</b>	0.0230

TABLE III  
PERFORMANCE SUMMARIZATION FOR ECLIPSE-II

Wrapper Metric	Classifier Metric									
	OA		AUC		PRC		BGM		BAM	
	Mean	Stdev	Mean	Stdev	Mean	Stdev	Mean	Stdev	Mean	Stdev
OA	0.9065	0.0082	0.8885	0.0172	0.6723	0.0250	0.8344	0.0171	0.8356	0.0168
AUC	0.9028	0.0095	<i>0.8574</i>	0.0278	0.6343	0.0404	0.8193	0.0230	0.8214	0.0213
PRC	<i>0.8965</i>	0.0095	0.8575	0.0242	<i>0.6160</i>	0.0382	<i>0.8168</i>	0.0231	<i>0.8184</i>	0.0226
BGM	0.9069	0.0065	<b>0.8911</b>	0.0199	0.6759	0.0253	0.8426	0.0158	0.8432	0.0156
BAM	<b>0.9071</b>	0.0064	0.8896	0.0202	<b>0.6766</b>	0.0263	<b>0.8438</b>	0.0176	<b>0.8450</b>	0.0174

TABLE IV  
PERFORMANCE SUMMARIZATION FOR ECLIPSE-III

Wrapper Metric	Classifier Metric									
	OA		AUC		PRC		BGM		BAM	
	Mean	Stdev	Mean	Stdev	Mean	Stdev	Mean	Stdev	Mean	Stdev
OA	0.8367	0.0087	0.8762	0.0130	0.7597	0.0194	0.8110	0.0103	0.8122	0.0097
AUC	0.8348	0.0092	0.8644	0.0129	0.7467	0.0169	0.8065	0.0116	0.8076	0.0118
PRC	<i>0.8295</i>	0.0113	<i>0.8519</i>	0.0158	<i>0.7284</i>	0.0229	<i>0.7923</i>	0.0151	<i>0.7940</i>	0.0151
BGM	<b>0.8387</b>	0.0060	0.8783	0.0128	0.7654	0.0146	0.8159	0.0128	0.8170	0.0122
BAM	0.8385	0.0072	<b>0.8789</b>	0.0108	<b>0.7655</b>	0.0127	<b>0.8160</b>	0.0103	<b>0.8170</b>	0.0096

Subsequently, the reduced dataset with the selected feature subset was used to build the respective final classification model. Note that the same learner (LR) is used within and outside wrapper. The experiments were conducted to discover the impact of (1) wrappers with different performance metrics; (2) different combinations of performance metrics used inside and outside the wrappers; and (3) three different groups of software metrics datasets with different class-imbalance levels from the software quality prediction domain. We implemented wrapper-based feature subset selection in WEKA and used it for the defect prediction model building and testing process. In the experiments, ten runs of five-fold cross-validation were performed. The five results from the five folds were then combined to produce a single estimation.

### C. Results and Analysis

Tables II to IV list the mean value and standard deviation for every classification model constructed over ten runs of five-fold cross-validation for each dataset group with different level of imbalance. Each column represents one choice of performance metric used to evaluate the final classification model, while each row is one choice of performance metric used within the wrapper. Within each column, the row which shows the best classification performance is indicated in **boldfaced** print, while the worst performance is *italic*.

The first notable observation from these experimental results is that BAM is the best wrapper metric for all of the classification metrics. This was true across all different levels of imbalanced datasets for 13 out of 15 cases. Typically, BGM was the second-place wrapper metric, except for two cases when it was the best metric. Nonetheless, based on these results, we recommend BAM as the wrapper metric which will produce the best classification performance.

TABLE V  
CLASSIFICATION MODEL PERFORMANCE ON FULL DATA

Dataset	OA	AUC	PRC	BGM	BAM
Eclipse I	0.8811	0.6346	0.1975	0.6428	0.6638
Eclipse II	0.8178	0.7129	0.3282	0.7099	0.7168
Eclipse III	0.7261	0.7142	0.4817	0.6738	0.6804

In addition to finding the best wrapper metrics, we also observed that PRC is the worst wrapper metric for the datasets with a moderate amount of class imbalance (that is, the Eclipse-II and Eclipse-III collections) regardless of which classification performance metric is used and was true for nine out of ten choices. For the most imbalanced datasets (the Eclipse-I collection), accuracy is the least favorable wrapper metric for four of five choices where the only exception is in the case of AUC. When AUC is used as both wrapper, and classification metric, it shows the worst performance.

We also performed experiments on the original datasets and results were shown in Table V. From these tables, we can conclude that the reduced feature subsets can have better prediction performance compared to the complete set of attributes (original dataset). This implies that software quality classification and feature selection were successfully applied in this study.

It is noted that Accuracy does not take class imbalance into account, while all four other metrics do so. So from the tables we can observe that accuracy suffers a much larger drop between the Balanced/Slightly Imbalanced datasets and the Imbalanced datasets than other performance metrics, such as AUC, have. We performed an ANalysis Of VAriance (ANOVA) F test to statistically examine the various effects on the performance metrics used inside the wrapper.

TABLE VI  
ANOVA ANALYSIS: OA

Eclipse-I	Source	Sum Sq.	d.f.	Mean Sq.	F	Prob>F
	Wrapper	0.00048	4	0.00012	1.13	0.3467
Error	0.01556	145	0.00011			
Total	0.01604	149				

Eclipse-II	Source	Sum Sq.	d.f.	Mean Sq.	F	Prob>F
	Wrapper	0.00243	4	0.00061	6.61	6.49E-05
Error	0.01335	145	0.00009			
Total	0.01578	149				

Eclipse-III	Source	Sum Sq.	d.f.	Mean Sq.	F	Prob>F
	Wrapper	0.0017	4	0.00042	1.26	0.2883
Error	0.0488	145	0.00034			
Total	0.0505	149				

TABLE VII  
ANOVA ANALYSIS: AUC

Eclipse-I	Source	Sum Sq.	d.f.	Mean Sq.	F	Prob>F
	Wrapper	0.01131	4	0.00283	1.21	0.3078
Error	0.33809	145	0.00233			
Total	0.3494	149				

Eclipse-II	Source	Sum Sq.	d.f.	Mean Sq.	F	Prob>F
	Wrapper	0.0377	4	0.00943	7.21	2.53E-05
Error	0.18957	145	0.00131			
Total	0.22728	149				

Eclipse-III	Source	Sum Sq.	d.f.	Mean Sq.	F	Prob>F
	Wrapper	0.01632	4	0.00408	5.73	0.0003
Error	0.10324	145	0.00071			
Total	0.11956	149				

An n-way ANOVA can be used to determine if the means in a set of data differ when grouped by multiple factors. If they do differ, one can determine which factors or combinations of factors are associated with the difference. We built our one-way ANOVA model for final classification model performance metrics OA and AUC (due to space consideration, we only represents results for these two performance metrics) on three different group of datasets separately. The factor A represents five wrappers. The ANOVA model can be used to test the hypothesis that the OA (or AUC) for the main factor A are equal against the alternative hypothesis that at least one mean is different. If the alternative hypothesis (i.e., that at least one mean is different) is accepted, multiple comparisons can be used to determine which of the means are significantly different from the others. In this study, we performed the multiple comparison tests using Tukey's honestly significant difference criterion. All tests of statistical significance utilize a significance level  $\alpha = 0.05$ .

The ANOVA results for the OA and AUC metrics are presented in Tables VI and VII, respectively. From the tables, we can see that for Eclipse-I-OA, Eclipse-I-AUC, and Eclipse-III-OA, the  $p$ -values (last column of the tables) for the main factor is greater than a typical cutoff value of 0.05, which implies no significant difference exists between the five wrappers across all 3 datasets in each group. The  $p$ -values for Eclipse-II-OA, Eclipse-II-AUC, and Eclipse-III-AUC are less than the typical cutoff value of 0.05, indicating that for the classification performance, the alternate hypothesis is accepted, namely, at least two group means are significantly different from each other for at least one pair of groups in the corresponding factors or terms.

Additionally Tukey's HSD test based multiple comparisons for the main factor were performed to investigate the difference among the respective groups (levels). The test results are shown in Figure 1 (for OA) and Figure 2 (for AUC), where each sub-figure displays graphs with each group mean represented by a symbol ( $\circ$ ) and 95%

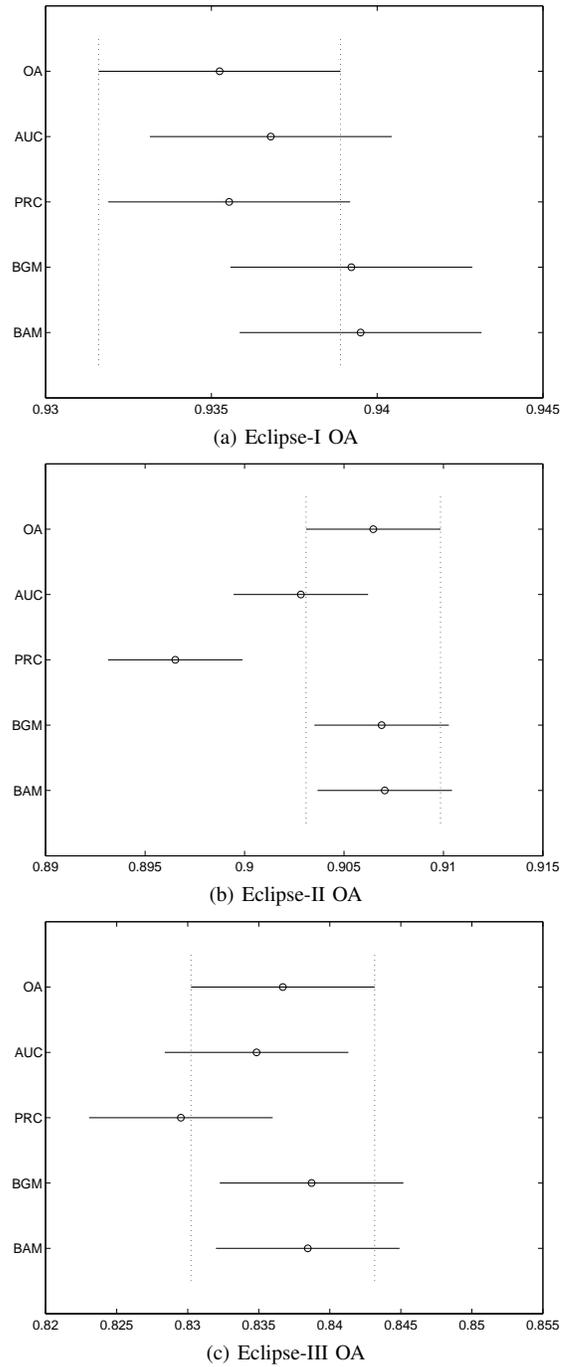


Fig. 1. Multiple Comparisons for OA

confidence interval. The results show the following facts: In general, wrapper with PRC performed worst among the 5 wrappers and BAM and BGM performed best.

## V. CONCLUSION

Identifying a small set of software metrics for building high accuracy software defect predictors can help reduce software development costs and produce a more reliable system. In this study, wrapper-based feature subset selection techniques have been used to find small subsets of attributes to build software defect prediction models. We used logistic regression as our learner both inside and outside

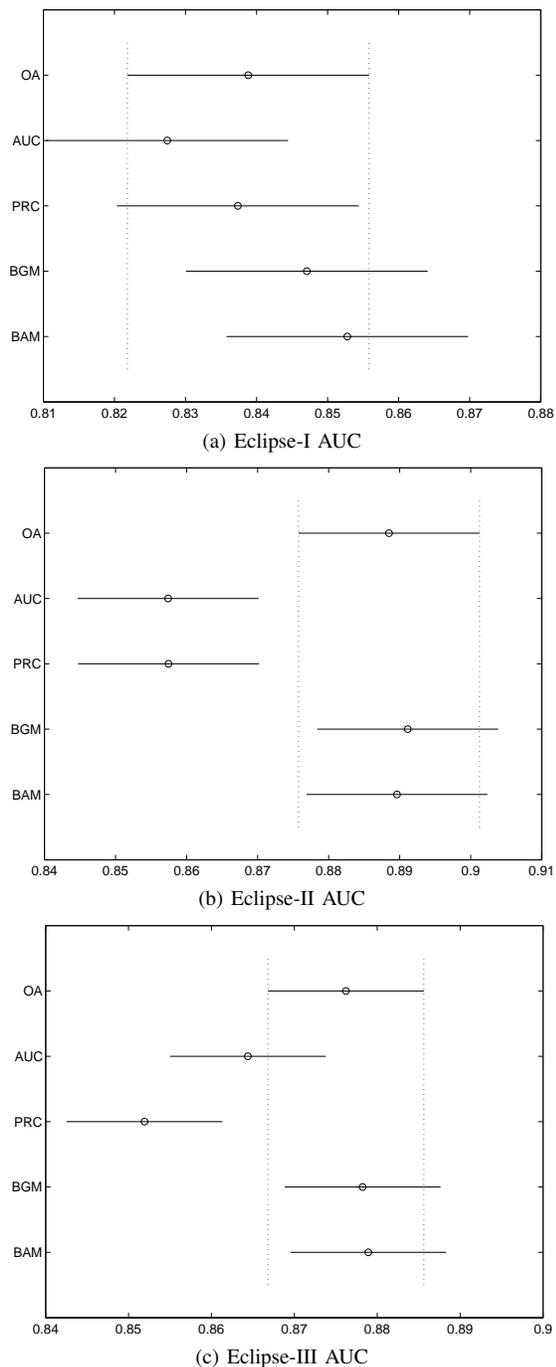


Fig. 2. Multiple Comparisons for AUC

the wrapper. Five performance metrics are used to evaluate models within the wrapper and also were used to evaluate the respective final classification models.

Our findings show that BAM is very reliable as a performance metric to evaluate models built within the wrapper. This is true for datasets with different degrees of imbalance.

Future work may include experiments using more classifiers, other feature subset search methods, and additional software metrics datasets from the software engineering domain. Given the imbalanced nature of the datasets studied in this paper, the use of sampling can also be considered to determine whether it will have any effects on

model performance with feature selection.

## REFERENCES

- [1] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 485–496, July–August 2008.
- [2] H. Wang, T. M. Khoshgoftaar, J. V. Hulse, and K. Gao, "Metric selection for software defect prediction," *International Journal of Software Engineering and Knowledge Engineering*, vol. 21, no. 2, pp. 237–257, 2011.
- [3] M. J. Meulen and M. A. Revilla, "Correlations between internal software metrics and software dependability in a large population of small C/C++ programs," in *Proceedings of the 18th IEEE International Symposium on Software Reliability Engineering, ISSRE 2007*, Trollhattan, Sweden, November 2007, pp. 203–208.
- [4] K. Gao, T. M. Khoshgoftaar, and A. Napolitano, "Improving software quality estimation by combining boosting and feature selection," in *ICMLA (1)*. IEEE, 2013, pp. 27–33.
- [5] D. Rodriguez, R. Ruiz, J. Cuadrado-Gallego, and J. Aguilar-Ruiz, "Detecting fault modules applying feature selection to classifiers," in *Proceedings of 8th IEEE International Conference on Information Reuse and Integration*, Las Vegas, NV, August 2007, pp. 667–672.
- [6] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of Machine Learning Research*, vol. 3, pp. 1157–1182, March 2003.
- [7] M. A. Hall and G. Holmes, "Benchmarking attribute selection techniques for discrete class data mining," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 6, pp. 1437 – 1447, Nov/Dec 2003.
- [8] H. Liu and L. Yu, "Toward integrating feature selection algorithms for classification and clustering," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 4, pp. 491–502, 2005.
- [9] Z. Chen, T. Menzies, D. Port, and B. Boehm, "Finding the right data for software cost modeling," *IEEE Software*, no. 22, pp. 38–46, 2005.
- [10] S. Le Cessie and J. C. Van Houwelingen, "Ridge estimators in logistic regression," *Applied Statistics*, vol. 41, no. 1, pp. 191–201, 1992.
- [11] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd ed. Morgan Kaufmann, 2005.
- [12] K. Gao, T. M. Khoshgoftaar, and H. Wang, "Exploring filter-based feature selection techniques for software quality classification," *IJIDS*, vol. 4, no. 2/3, pp. 217–250, 2012.
- [13] R. Kohavi and G. H. John, "Wrappers for feature subset selection," *Artificial Intelligence*, vol. 97, no. 1-2, pp. 273–324, Dec. 1997. [Online]. Available: [http://dx.doi.org/10.1016/S0004-3702\(97\)00043-X](http://dx.doi.org/10.1016/S0004-3702(97)00043-X)
- [14] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861–874, June 2006.
- [15] Y. Jiang, J. Lin, B. Cukic, and T. Menzies, "Variance analysis in software fault prediction models," *Software Reliability Engineering, International Symposium on*, vol. 0, pp. 99–108, 2009.
- [16] N. Seliya, T. M. Khoshgoftaar, and J. Van Hulse, "A study on the relationships of classifier performance metrics," in *Proceedings of the 21st IEEE International Conference on Tools with Artificial Intelligence (ICTAI'09)*. Newark, NJ: IEEE Computer Society, November 2009, pp. 59–66.
- [17] J. Van Hulse, T. M. Khoshgoftaar, A. Napolitano, and R. Wald, "Feature selection with high dimensional imbalanced data," in *Proceedings of the 9th IEEE International Conference on Data Mining - Workshops (ICDM'09)*. Miami, FL: IEEE Computer Society, December 2009, pp. 507–514.
- [18] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse," in *ICSEW '07: Proceedings of the 29th International Conference on Software Engineering Workshops*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 76–85.

# Synthetic Minority Over-sampling TEchnique (SMOTE) for Predicting Software Build Outcomes

Russel Pears & Jacqui Finlay  
School of Computing & Mathematical Sciences  
Auckland University of Technology  
Auckland, New Zealand  
russel.pears@aut.ac.nz

Andy M. Connor  
Colab  
Auckland University of Technology  
Auckland, New Zealand  
andrew.connor@aut.ac.nz

**Abstract—** In this research we use a data stream approach to mining data and construct Decision Tree models that predict software build outcomes in terms of software metrics that are derived from source code used in the software construction process. The rationale for using the data stream approach was to track the evolution of the prediction model over time as builds are incrementally constructed from previous versions either to remedy errors or to enhance functionality. As the volume of data available for mining from the repository was limited, we synthesized new data instances through the use of the SMOTE oversampling algorithm. The results indicate that a small number of the available metrics have significance for prediction software build outcomes. It is observed that classification accuracy steadily improves after approximately 900 instances of builds have been fed to the classifier. At the end of the data streaming process classification accuracies of 80% were achieved, though some bias arises due to the distribution of data across the two classes.

**Keywords-** SMOTE, Data Stream Mining, Jazz, Software Metrics, Software Repositories.

## I. INTRODUCTION

The Mining Software Repositories (MSR) field analyses the rich data available in software repositories to uncover interesting and actionable information about software systems and projects. This enables researchers to reveal interesting patterns and information about the development of software systems. MSR has been a very active research area since 2004 [4]. Until the emergence of MSR as a research endeavor, the data from software repositories were mostly used as historical records for supporting development activities. Analysis of MSR research has shown that the approach of extracting knowledge from a repository has the potential to be a valuable method for analyzing the software development process for many domains [5]. However there are a number of data related challenges, one of which is how to deal with repositories that contain insufficient or imbalanced data because the project is still immature or because data is discarded over time.

Our previous work [21] developed an approach for applying data stream mining techniques to overcome the challenge of discarded data utilizing the Jazz repository [1]. The Jazz repository stores data, including source code, related to each software build attempt and retains the build outcome, categorized as success or failure. As the volume of data associated with each build is large only a limited number of

build instances are actually stored in the repository on a first-in, first-out basis. Traditional data mining methods are tailored to static data environments where the data is retained. The first major challenge in mining software repositories is dealing with dynamic data that arrives on a continuous basis. Our previous work [21] addressed this challenge by modeling the development process as a data stream to deal with software project data that is produced continuously and accumulated over a period of time before being discarded. The data stream mining approach was shown to be effective at maintaining knowledge related to the project even after data is discarded.

This paper attempts to extend our work to address the second challenge, namely to deal with the limited volume of data and improve the accuracy of the prediction event. In order to boost the training power of the limited quantity of available data the SMOTE [2] oversampling algorithm was applied to synthesize new data instances from the available instances prior to inducing a decision tree model implemented via the Hoeffding [3] tree method. For the simulation to be realistic the naturally occurring distribution of successful and failed build instances occurring in the original population was maintained in the oversampling process. Future work will consider applying SMOTE to alter the balance of the dataset to increase the representation of the minority class.

The dynamic nature of software and the resulting changes in software development strategies over time causes changes in the patterns that govern software project outcomes. This phenomenon has been recognized in many other domains and is referred to as concept drift. Changes in a data stream can evolve slowly or quickly and rates of change can be queried within stream-based tools. This paper describes an attempt to improve build outcome prediction accuracies for the Jazz project by synthetically creating data to boost the training power of the data stream mining approach while taking into account concept drift that occurs as part of the stream.

## II. BACKGROUND AND RELATED WORK

This research draws from multiple areas to inform the inquiry, in particular it is placed in the context of other Mining Software Repositories research, specifically in the context of the Jazz repository. In addition, it uses experience gained applying data stream mining and synthetic data generation in other domains to improve the current prediction models.

### A. Mining the Jazz Repository

The Jazz development environment has been recognized as offering new opportunities in terms of MSR research because it integrates the software source code archive and bug database by linking bug reports and source code changes with each other [6]. Whilst this provides much potential in gaining valuable insights into the development process of software projects, such potential is yet to be fully realized. To date, much of the work focused on the Jazz repository is related to predicting build success, either through social network analysis [7] or source code metrics [21, 22]. As is common with much MSR research, the goal of working with the Jazz repository is in line with a key direction identified in the field [23], which is the transformation of software repositories from static record-keeping ones into active repositories in order to guide decision processes in modern software projects.

### B. Data Stream Mining

The mining of data streams has arisen as a necessity due to advances in hardware/software that have enabled the capture of different measurements of data in a wide range of fields [24]. Data streams are typically generated continuously and have very high fluctuating data rates. The storage, querying and mining of such data sets are computationally challenging tasks [24]. Research problems and challenges that have been arisen in mining data streams can be solved using well-established statistical and computational approaches that can be categorized as either data-based or task-based ones. In data-based solutions, only a subset of the whole dataset is examined or the data is transformed to an approximate smaller size representation. Task-based solutions involve applying techniques from computational theory to achieve time and space efficient solutions. Data-based solutions include Sampling, Load Shredding, Sketching and Aggregation. Task-based solutions include Approximation Algorithms and Sliding Window approaches, all of which have received considerable attention by researchers [28]. The discarding of data from the Jazz environment and the relatively low data rate lends itself to a Sliding Window solution.

In addition, various data mining approaches can be applied to mining data streams, including clustering, frequency counting and classification. The nature of the data, which includes a classifiable attribute in terms of build outcome, lends itself to a classification method. In this work, we have applied the Hoeffding tree incremental learner in conjunction with the Adaptive Sliding Window (ADWIN) concept drift detector. ADWIN is a parameter-free adaptive sliding window drift detector that compares all adjacent sub-windows in given data window in order to detect concept drift [29]. This method is recognized to produce high true positive and low false positives rates while, having low detection delay times in comparison to other drift detectors proposed in the data mining literature [29].

### C. Synthetic Data Generation

Many of the challenges associated with data stream mining are related to dealing with high volumes of data in relatively short timescales. It therefore seems counter-intuitive to deploy synthetic data generation techniques in conjunction with a data stream mining approach. However, the discarding of data from Jazz does not encourage the use of static classification

approaches in practice, even though such approaches can be deployed on any given snapshot of the repository [22]. Deploying a data stream method in conjunction with synthetic data generation allows a consistent approach to be used in practice for new projects. Synthetic data can be generated from the limited quantity of actual data that is available in the early stages of development, and a gradual phasing out of such synthetic data can be carried out when larger volumes of real data become available. Such consistency is important if data mining approaches are to be useful to software practitioners.

Synthetic data generation has been a research area for some time, with the literature containing many examples of random or pseudo-random data generation [25]. However, the goal of our research is such that synthetic data must be representative of the real data and therefore a more refined generation approach is required. Such approaches include DataBoost-IM [26], ADASYN [27] and SMOTE [2] to name but a few. Many of these approaches are based on similar sampling algorithms and in this work we have elected to apply the standard SMOTE algorithm as it has been effectively applied in many domains.

## III. THE JAZZ DATASET

IBM Jazz is a fully integrated software development tool that automatically captures software development processes and artifacts. The Jazz repository contains real-time evidence that allows researchers to gain insights into team collaboration and development activities within software engineering projects [1, 7]. The Jazz repository artifacts include work items, build items, change sets, source code files, authors and comments. A work item is a description of a unit of work, which is categorized as a task, enhancement or defect. A build item is compiled software to form a working unit. A change set is a collection of code changes in a number of files. In Jazz a change set is created by one author only and relates to one work item. A single work item may contain many change sets. Source code files are included in change sets and over time can be related to multiple change sets.

One of the challenges associated with working with the Jazz repository is that the data contains holes and misleading elements which cannot be removed or identified easily. This is because the Jazz environment has been used within the development of itself; therefore many features provided by Jazz were not implemented at early stages of the project. This sparseness of the data has driven the decision to focus on using software metrics as the predictor attributes. Whilst features of the Jazz environment may not have been present during early phases of development, there has always been source code and therefore a consistent set of data can be created.

## IV. THEORETICAL FOUNDATIONS

Software metrics have been generated in order to deal with the sparseness of the data. Metric values can be derived from extracting development code from software repositories. Such metrics are commonly used within model-based project management methods. Software metrics are used to measure the complexity, quality and effort of a software development project [8-12]. In the Jazz repository each software build contains change sets that indicate the actual source code files that are modified during the implementation of the build.

Source code metrics for each file are computed using the IBM Software Analyzer tool. The builds after state was utilized in order to ensure that the source code snapshot represented the actual software artifact that either failed or succeeded.

The Jazz repository consists of various types of software builds. Included in this study were continuous builds (regular user builds), nightly builds (incorporating changes from the local site) and integration builds (integrating components from remote sites). As a result the following basic, average basic, dependency, complexity, cohesion and Halstead software metrics were derived from the source code files for each build:

- Basic Software Metrics:
  - Number of Types Per Package, Number of Comments, Lines of Code, Comment/Code Ratio, Number of Import Statements, Number of Interfaces, Number of Methods, Number of Parameters, Number of Lines, Average Number of Attributes Per Class, Average Number of Constructors Per Class, Average Number of Comments, Average Lines of Code Per Method, Average Number of Methods, Average Number of Parameters.
- Dependency Metrics:
  - Abstractness, Afferent Coupling, Efferent Coupling, Maintainability index, Instability, Normalized Distance.
- Complexity Metrics:
  - Average Block Depth, Average Cyclomatic Complexity.
- Cohesion Metrics:
  - Lack of Cohesion 1 (LCOM1), Lack of Cohesion 2 (LCOM2), Lack of Cohesion 3 (LCOM3).
- Halstead Metrics:
  - Number of Operands, Number of Operators, Number of Unique Operands, Number of Unique Operators, Program Volume, Difficulty Level, Effort to Implement, Number of Delivered Bugs, Time to Implement, Program Length, Program Level, Program Vocabulary Size.

#### A. Synthetic Minority Over-sampling TEchnique (SMOTE)

When working with real world data it is often found that data sets are heavily comprised of “normal” instances with only a small percentage representing interesting findings. As a result the “abnormal” instances have a negative impact on a models' performance as they have a greater probability of misclassification using data mining methods [2, 13]. Data instances that introduce noise within the data are often found within the minority class [14, 15]. In order to overcome this limitation synthetically under-sampling the majority class may improve a classifiers' performance. However, in doing so valuable data may be lost and model over-fitting may occur, resulting in majority instances being wrongly classified as minority instances when new, unseen data is presented to the classifier model that was induced [14]. Another solution is to provide the classifier with more complete regions within the feature space via creation of new instances that are synthesized from existing data instances.

SMOTE enables a data miner to over sample the minority class to achieve potentially better classifier performance without loss of data [2, 13]. While other over-sampling methods exist, such as Rippers Loss Ratio and Naive Bayes methods, SMOTE provides better levels of performance as it generates more minority class samples for a classifier to learn from, thereby allowing broader decision regions and coverage [13]. SMOTE has been utilized within the software research community and compared with other sampling techniques in software quality modeling (random under-sampling, random

oversampling, cluster-based oversampling and Borderline-SMOTE) and has yielded encouraging results [5, 8]. SMOTE has also been applied as a sampling strategy for software defect prediction where data sets from NASA software project data sets [10,16-18] and fault-prone module detection using the MIS telecommunication systems [24]. For this work SMOTE is applied as a supervised instance filter using the Weka [19] machine learning workbench.

In order to avoid the over-fitting problem while expanding minority class regions SMOTE generates new instances by operating within the existing feature space. New instance values are derived from interpolation rather than extrapolation, so they still carry relevance to the underlying data set. For each minority class instance SMOTE interpolates values using a k-nearest neighbor technique and creates attribute values for new data instances [8]. For each minority data a new synthetic data instance (I) is generated by taking the difference between the feature vector of I and its nearest neighbor (J) belonging to the same class, multiplying it by a random number between 0 and 1 and then adding it to I. This creates a random line segment between every pair of existing features from instances I and J, resulting in the creation of a new instance within the data set [13]. This process is repeated for the other k-1 neighbors of the minority instance I. As a result SMOTE generates more general regions from the minority class and decision tree classifiers are able to use the data set for better generalizations.

#### B. Hoeffding Tree

The Hoeffding tree is an incremental decision tree induction method. Using the Hoeffding bound, it ascertains the number of instances that are needed to split a given (decision) node of a tree and operates within a certain precision that can be predetermined [3]. This method has potential in terms of predicting future outcomes of software builds with high accuracy while working with real-world data. Rather than using training and test sets, instances are represented as streams. The Hoeffding tree is commonly used for classifying high speed data streams. The algorithm that it uses generates a decision tree from data incrementally by inspecting each instance within a stream without the need to store instances for later retrieval. The tree resides in memory during each iteration and stores information in its branches and leaves, potentially growing from “learning” every new instance. The decision tree itself can be inspected at any time during the streaming process. The quality of the tree itself is comparable to that used by traditional mining techniques, even though instances are introduced in an incremental manner.

Just as with traditional decision tree learners, the Hoeffding tree is easy to interpret, making it easier to understand how the model works. In addition to this, decision tree learners have proven to provide accurate solutions to a wide range of problems that are based on multi-dimensional data. For Hoeffding trees each node of a decision tree undergoes a test which may result in it being split into two or more child nodes and sending each instance down a relevant branch to its destination child node, depending on the values of its attributes. The split test is implemented through the use of the Hoeffding bound which is expressed in equation 1.

$$\epsilon = \sqrt{\frac{R^2 \ln\left(\frac{1}{\delta}\right)}{2n}} \quad (1)$$

The Hoeffding bound expressed in (1) above states that with confidence  $(1 - \delta)$ , the population mean of  $R$  lies in the interval,  $[\bar{R} - \epsilon, \bar{R} + \epsilon]$ , where  $\bar{R}$  is the sample (observable) mean of the random variable  $R$ . In the context of decision tree induction  $R$  refers to information gain. The Information gain function ranges in value from 0 to  $\log_2 c$ , where  $c$  is the number of classes. Since  $c=2$  in the mining problem that we undertake (since only the outcomes, *success* and *failure* are possible),  $R$  reduces to 1. The variable  $n$  refers to the number of data instances seen up to the point that the test was carried out. The bound holds is true irrespective of the underlying data distribution generating the values and only depends on a range of values, number of observations made and a split confidence level. The Hoeffding tree uses the Hoeffding bound to determine whether an existing (leaf) node should be split as follows. Suppose that after  $n$  data instances have arrived, the difference in information gain between the two highest ranking attributes  $X_a$  and  $X_b$  with  $\Delta \bar{G} = \bar{G}(X_a) - \bar{G}(X_b) > \tau$  (i.e.  $X_a$  is the attribute with the highest information gain), then with confidence  $(1 - \delta)$ , the Hoeffding bound guarantees that the correct choice to split the given leaf node is attribute  $X_a$  if  $\Delta \bar{G} > \epsilon$ , where  $\tau$  is a tie threshold parameter.

In this research we use the Hoeffding tree implementation from MOA [13], a real time analytics tool for data streams was used for mining data streams.

## V. EXPERIMENTAL STUDY

The original software metric data set consists of 199 Jazz build instances. From these instances there are 127 successful builds and 72 failed builds. Build instances are sorted by date to ensure accurate simulation of a development team working over time. SMOTE is then applied twice at 900%, increasing the number of instances to 1,990 (1270 successful builds and 720 failed builds). The first application increases the number of minority class instances (failed builds) and the second application increases the temporarily “new” number of minority class instances (successful builds). The instances are then encoded into data streams which are utilized by the Hoeffding tree for the data mining process. Three parameters were set for the tree induction. The Hoeffding tree uses a grace period parameter which stipulates the frequency with which checks for leaf node splits are carried out, the greater the value the higher the efficiency of the process. We use a setting of 200 for the grace period parameter. The tie threshold parameter,  $\tau$  that controls the degree of splitting, was set to 0.05.

Figure 1 presents the classification accuracy obtained with the use of after state metrics for builds. The classification accuracy at the start of the time series was 65.2% and at the end of the stream the accuracy grew to 80.25%. The average overall accuracy over the entire time series was 70%. This indicates that there is potential for the accuracy of prediction to improve as more real data emerges.

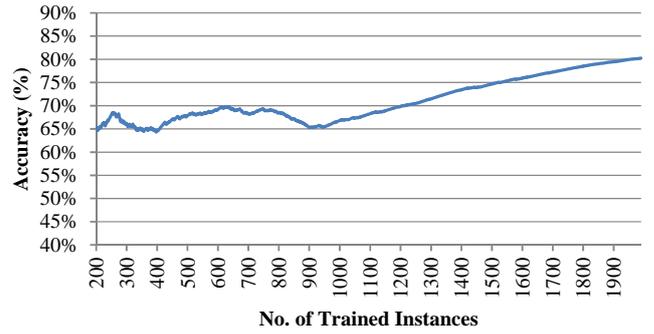


Figure 1. Hoeffding Tree Overall Classification Accuracy.

The initial instability in classification accuracy is an interesting phenomenon, given the initial grace period of 200 builds is intended to provide stability in the emerging model. Upon examination of the synthetic data it can be observed that the data maintains comparable instances of each class up until 900 builds. After 900 builds, the data contains an increasing proportion of successful builds. This is shown in Figure 2.

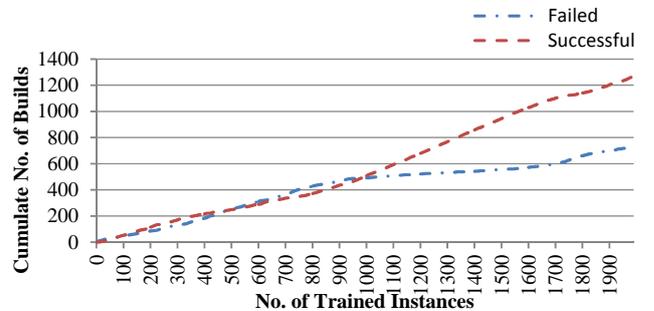


Figure 2. Build Distribution over Time

Figure 3 presents the classification accuracies of successful builds. It is observed that the general trend for classifying success initially declines to reach a minimum at approximately 900 instances, after which there is a gradual improvement that appears to be trending towards a stable value of around 80%. Figure 4 displays the sensitivity ratings for successful builds over time. For successful builds the accuracy at the beginning of the data stream time series was 66.38% and ended with 79.1% (with an average of 64%).

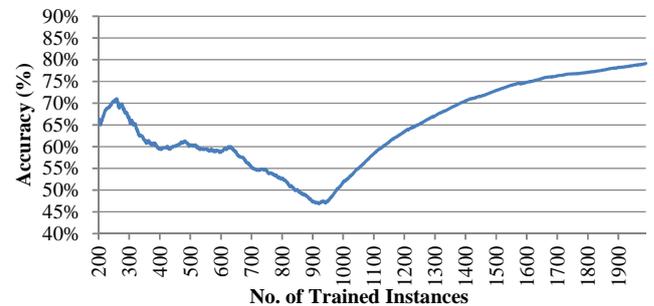


Figure 3. Hoeffding Tree Classification Accuracy for Successful Builds.

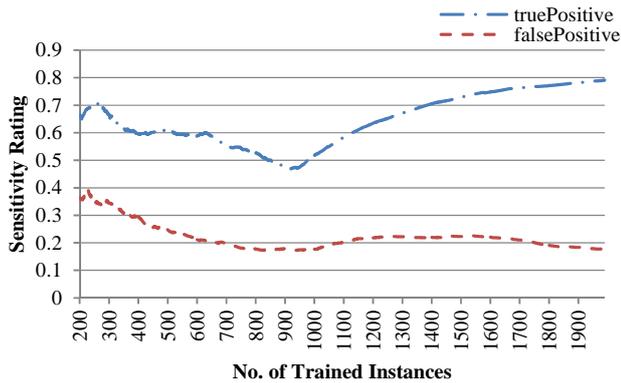


Figure 4. Hoeffding Tree Sensitivity Measurements for Successful Builds.

Figure 5 shows the classification accuracy over time for failed builds and the corresponding sensitivity ratings for failed builds are shown in Figure 6. Accuracy for failed builds started at 63.5% and at the end of the time series was 82.2% (with an average of 78%). The false positive values between 700 to 1000 trained instances appear to peak when classifying failed builds, due to over-fitting the model at earlier time segments. The false positive value then proceeds to decrease over time

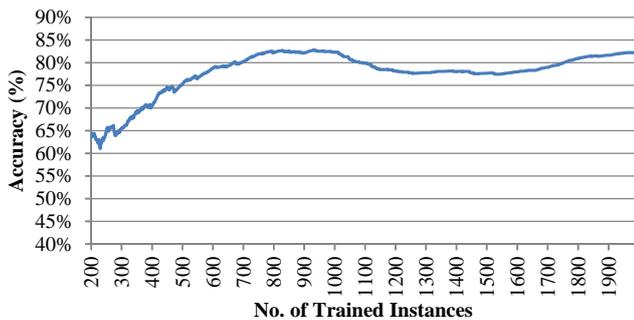


Figure 5. Hoeffding Tree Classification Accuracy for Failed Builds.

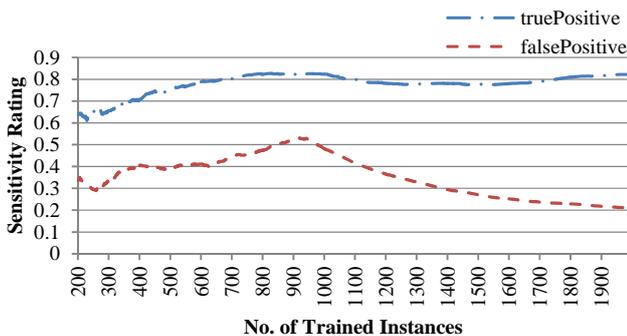


Figure 6. Hoeffding Tree Sensitivity Measurements for Failed Builds.

Interestingly, the distribution of build instances across the two classes has marginal impact on the overall classification accuracy when compared to the impact it has on the individual classes themselves. When the distribution of classes in the synthetic data is roughly equal there is an increase in the classification accuracy of failed builds that is accompanied by a decrease in classification accuracy of successful builds. This seems at odds with the observation of previous work [21, 22, 30] that suggests that failed builds are harder to classify than

successful builds. This work suggests that failed builds may be harder to classify when there is a significantly larger number of successful builds that dominate the classification model.

Figure 7 illustrates the final decision tree using the Hoeffding Tree stream mining technique on the extended RSA after state software metrics data set. In this case the tree is larger than the previous software metric based Hoeffding tree, with a depth of 7. Upon inspecting the tree there are common sense classifications being made, for example a higher number of interfaces tend to be associated with failure. This is intuitive because if there are too many Java interfaces it can become tedious when debugging an error as the actual implementation of the error may be in an obscure location. Interfaces also add to the collection of files within the system and if an interface is “dead” (not used) and not removed it leads to a less elegant system design. The number of interfaces has a direct influence on dependency metrics, i.e. Abstractness.

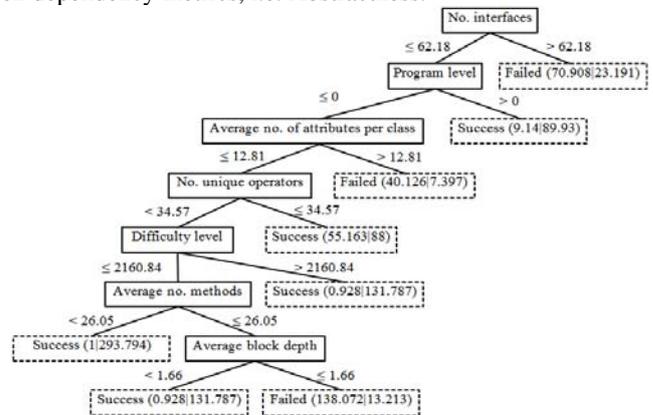


Figure 7. Final Hoeffding Tree for After State Software Metrics.

In comparison with the results obtained without applying SMOTE [21], the overall accuracy has increased as has the complexity of the classification tree, though the metric “Average number of attributes per class” is still significant.

## VI. LIMITATIONS AND FUTURE WORK

The Jazz repository contains holes and misleading elements which cannot be removed or identified easily. There is a great challenge in dealing with such inconsistency and the methodology has adopted an approach that delves further down the artifact chain than most previous work using Jazz. It is a premise that the early software releases were functional, so whilst the project “meta-data” may be missing details (such as developer comments) the source code should represent a stable system that can be analyzed to gain insight regarding the development project. Even when comparing to other Jazz studies there are concerns over validity that arise from the possibility of different extraction techniques being applied. However, the approach for creating a predictive model by mining data streams that relate to software data can be applied to other repositories and as such is a generalizable process. Similarly, the process of using the predictive model to identify build outcome risk and proactively manage the build scope and activities is equally applicable to other projects. The actual prediction models are likely to be different for other projects, but the techniques for developing them are generic. Other

limitations from this study are products of the relatively small sample size of build data combined with the sparseness of the data itself. For example, the ratio of metrics (42) to builds (199) is such that it is difficult to identify significant metrics. Even though a SMOTE is applied to increase the number of instances, it is not possible to assess the extent to which the generated data reflects real-world data as there is the likelihood of unpredictable events in software development projects.

## VII. CONCLUSIONS

The goal of synthetically generating data was to explore what might happen if there was more data available for mining and specifically to see if classification of builds. While the use of SMOTE may not be a “true” representation of future real world data, it does however interpolate values between existing instances to generate new data that may be considered at representative of existing data. This provides insights into what may occur if there were no “new” anomalies encountered during the project. This may not be entirely realistic given that the causes of failure are not predictable and that new failure modes are likely to appear over time. From previous data mining experiments it was observed that build failure metrics were often overlapped in value with those of successful builds, thus challenging the ability of a classifier to distinguish between these two types of build outcomes. This indicates that if more data is available accuracy for classifying builds may improve over time. The results obtained support other studies where software build outcome prediction accuracy and stability both increased when adopting the use of SMOTE [10, 11, 20] on other project software metrics. While this research has gone some way to addressing the challenges associated with mining software repositories, there is still much potential for future work in understanding success and failure patterns.

## REFERENCES

- [1] Nguyen, T., Schröter, A., & Damian, D. “Mining Jazz: An experience report”. Proceedings of the Infrastructure for Research in Collaborative Software Engineering Conference, 2008
- [2] Chawla, N., Bowyer, K., Hall, L. and Kegelmeyer, W. “SMOTE: Synthetic Minority Over-sampling TEchnique.” Journal of Artificial Intelligence Research, vol. 16, pp. 341-378, 2002.
- [3] Giannella, C., Han, J., Pei, J., Yan, X., and Yu, P. S. “Mining frequent patterns in data streams at multiple time granularities”. Next generation data mining, pp. 191-212, 2003.
- [4] Kagdi, H., Collard, M., and Maletic, J. “A survey and taxonomy of approaches for mining software repositories in the context of software evolution”. Journal of Software Maintenance and Evolution: Research and Practice, vol. 19, pp. 77 - 131, 2007.
- [5] Poncin, W., Serebrenik, A., and Brand, M. “Process mining software repositories”. 15th European Conference In Software Maintenance and Reengineering (CSMR), pp. 5-14, 2011.
- [6] Herzig, K., and Zeller, A. “Mining the Jazz Repository: Challenges and Opportunities”. Mining Software Repositories MSR '09. 6th IEEE International Working Conference, pp. 159-162, 2009.
- [7] Wolf, T., Schroter, A., Damian, D., and Nguyen, T. “Predicting build failures using social network analysis on developer communication”. Proceedings of the IEEE International Conference on Software Engineering (ICSE), 2009.
- [8] Drown, D. J., Khoshgoftaar, T.M. Seliya, N. “Evolutionary Sampling and Software Quality Modeling of High-Assurance Systems.” Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions, vol. 39, pp. 1097-1107, 2009.
- [9] Manduchi, G. and Taliercio, C. “Measuring software evolution at a nuclear fusion experiment site: a test case for the applicability of OO and reuse metrics in software characterization”, Information and Software Technology, vol. 44, pp. 593-600, 2002.
- [10] Pelayo, L. and Dick, S. “Applying novel resampling strategies to software defect prediction.” In Fuzzy Information Processing Society, 2007. NAFIPS'07. Annual Meeting of the North American IEEE, pp. 69-72, 2007.
- [11] Shatnawi, R. “Improving software fault-prediction for imbalanced data.” Paper presented at the Innovations in Information Technology (IIT), International Conference, 2012.
- [12] Kamei, Y., Monden, A., Matsumoto, S., Kakimoto, T., and Matsumoto, K. I. “The effects of over and under sampling on fault-prone module detection”. First International Symposium on Empirical Software Engineering and Measurement, pp. 196-204, 2007.
- [13] Bifet, A., Holmes, G., Kirkby, R. and Pfahringer, B. “MOA: Massive Online Analysis.” In Journal of Machine Learning Research, vol. 11, pp. 1601-1604, 2010.
- [14] Haibo, H. and Garcia, E. A. “Learning from Imbalanced Data”. IEEE Transactions on Knowledge and Data Engineering, vol. 21, 1263-1284, 2009.
- [15] Jeatrakul, P., Kok Wai, W., Chun Che, F. and Takama, Y. “Misclassification analysis for the class imbalance problem”. Paper presented at the World Automation Congress (WAC), 2010.
- [16] Gray, D., Bowes, D., Davey, N., Sun, Y. and Christianson, B. “Using the Support Vector Machine as a Classification Method for Software Defect Prediction with Static Code Metrics. Engineering Applications of Neural Networks”, pp. 223-234, 2009.
- [17] Jiang, Y., Li, M., and Zhou, Z. H. “Software defect detection with ROCUS”. Journal of Computer Science and Technology, vol. 26, pp. 328-342, 2011.
- [18] Seliya, N., Khoshgoftaar, T. M. and Hulse, J. V. “Predicting Faults in High Assurance Software”. Paper presented at the Proceedings of the 2010 IEEE 12th International Symposium on High-Assurance Systems Engineering, 2010.
- [19] Hall, M., Frank, E. Holmes, G., Pfahringer, B., Reutemann, P. and Witten, I. H. “The WEKA Data Mining Software: An Update.” SIGKDD Explorations, vol. 11, pp. 10-18, 2009.
- [20] Kehan, G., Khoshgoftaar, T. M., and Napolitano, A. “Impact of Data Sampling on Stability of Feature Selection for Software Measurement Data”. Paper presented at the Tools with Artificial Intelligence (ICTAI), 2011 23rd IEEE International Conference, 2011
- [21] Finlay, J., Pears, R. & Connor, A.M. “Data stream mining for predicting software build outcomes using source code metrics”, Information & Software Technology, 56(2), 183-198, 2014.
- [22] Finlay, J., Connor, A.M. & Pears, R. “Mining software metrics from Jazz”, Software Engineering Research, Management and Applications 2011, Springer Berlin / Heidelberg. 377: 95-111, 2011.
- [23] Hassan, A.E., “The road ahead for Mining Software Repositories” Frontiers of Software Maintenance, 2008. FoSM 2008. pp.48,57, Sept. 28 2008-Oct. 4 2008
- [24] Mohamed Medhat Gaber, Arkady Zaslavsky, and Shonali Krishnaswamy. 2005. Mining data streams: a review. SIGMOD Rec. 34, 2 (June 2005), 18-26.
- [25] Jim Gray, Prakash Sundaresan, Susanne Englert, Ken Baclawski, and Peter J. Weinberger. “Quickly generating billion-record synthetic databases”. SIGMOD Rec. 23(2), 243-252, 1994
- [26] Hongyu Guo and Herna L. Viktor. “Learning from imbalanced data sets with boosting and data generation: the DataBoost-IM approach”, SIGKDD Explorations Newsletter 6(1), 30-39, 2004
- [27] Haibo He; Yang Bai; Garcia, E.A.; Shutao Li, “ADASYN: Adaptive synthetic sampling approach for imbalanced learning,” IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence), 2008
- [28] Aggarwal, C. C. (Ed.). “Data streams: models and algorithms” (Vol. 31). Springer, 2007.
- [29] Bifet, A. “Adaptive learning and mining for data streams and frequent patterns.” SIGKDD Explorations Newsletter, 11(1), 55-56, 2009.
- [30] Connor, A.M., Finlay, J.A. and Pears, R. “Mining Developer Communication Data Streams”. Proceedings of the Fourth International Conference on Computer Science and Information Technology, 2014

# Building Empirical Software Engineering Bodies of Knowledge with Systematic Knowledge Engineering

Stefan Biffl<sup>a</sup>, Marcos Kalinowski<sup>b</sup>, Fajar Ekaputra<sup>a</sup>, Estefanía Serral<sup>a</sup>, Dietmar Winkler<sup>a</sup>

<sup>a</sup> CDL-Flex, Inst. Software Technology and Interact.Sys.  
Vienna University of Technology  
Vienna, Austria  
<firstname>.<lastname>@tuwien.ac.at

<sup>b</sup> NEnC, Núcleo de Engenharia do Conhecimento  
Federal University of Juiz de Fora  
Juiz de Fora, Brazil  
kalinowski@ice.ufjf.br

**Abstract**—[Context] Empirical software engineering (EMSE) researchers conduct systematic literature reviews (SLRs) to build bodies of knowledge (BoKs). Unfortunately, valuable knowledge collected in the SLR process is publicly available only to a limited extent, which considerably slows down building BoKs incrementally. [Objective] In this paper, we introduce the Systematic Knowledge Engineering (SKE) process to support building up BoKs from empirical studies efficiently. [Method] SKE is based on the SLR process and on Knowledge Engineering (KE) practices to provide a Knowledge Base (KB) with semantic technologies that enable reusing intermediate data extraction results and querying of empirical evidence. We evaluated SKE by building a software inspection EMSE BoK KB from knowledge acquired by controlled experiments. We elicited relevant queries from EMSE researchers and systematically integrated information from 30 representative research papers into the KB. [Results] The resulting KB was effective in answering the queries, enabling knowledge reuse for analyses beyond the results from the SLR process. [Conclusion] SKE showed promising results in the software inspection context and should be evaluated in other contexts for building EMSE BoKs faster.

**Keywords**—Empirical software engineering, systematic knowledge engineering, systematic review, software inspection.

## I. INTRODUCTION

Researchers in Empirical Software Engineering (EMSE) collaborate on research topics, such as defect detection methods for software inspection [1], to build up a body of knowledge (BoK). An EMSE BoK includes theory models [2], hypotheses derived from the theory models, and results from empirical studies that test those hypotheses [3], to explain and/or predict EMSE phenomena.

A consequence of the growing number of EMSE studies is the need to adopt systematic approaches for aggregating research outcomes in order to provide an objective summary of evidence on a particular topic [4]. In this context, systematic literature reviews (SLRs) have become a widely used research method [5]. However, the main public result of a SLR is, in general, a specific research synthesis report [6], while the accumulated knowledge in the SLR working material, generated from heterogeneous empirical study data sources [7], is not available to other researchers. Therefore, each new SLR project has to overcome knowledge sharing issues and rebuild large parts of existing knowledge from previous SLRs, making building a BoK

considerably less efficient than necessary. Moreover, meta-analyses are limited to the presented research synthesis, not allowing other researchers to explore the underlying extracted information in different ways in order to answer questions related to their specific research goals.

Fig. 1 illustrates EMSE BoK building research challenges with key stakeholders, artifacts, and technologies. EMSE BoK researchers produce SLR and empirical study reports, available to general readership through digital libraries. However: (1) data extracted during the SLR process usually stays in a local archive and (2) the SLR report contains a specific research synthesis and there is seldom a way for other researchers to access the extracted data to apply different analyses and research synthesis methods, or to extend the data in the BoK.

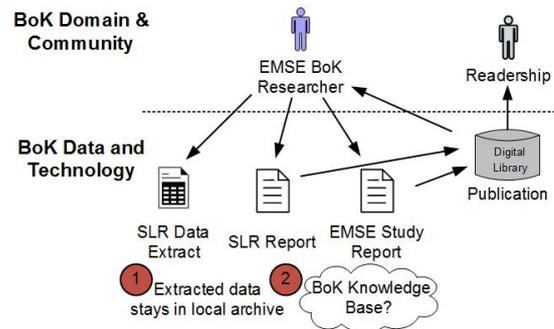


Figure 1. EMSE BoK research challenges.

In this paper, we address these challenges by introducing the Systematic Knowledge Engineering (SKE) process, which enables systematically building up EMSE BoKs from empirical studies. SKE builds on the SLR process [8], improving data management by storing knowledge on BoK domain concepts typically used for theory building [2] from EMSE studies [3] in a Knowledge Base (KB). The semantic technology used in SKE facilitates researchers to identify relevant knowledge in the BoK through semantic queries (e.g., on synonyms and related concepts). Thus, SKE allows truly building up knowledge incrementally as researchers can access and reuse knowledge from past studies and integrate new contributions. Also, the KB enables researchers to explore knowledge aggregated from EMSE study reports through an extensible set of queries.

For evaluation we instantiated SKE to build a “Software Inspection” EMSE BoK KB based on a specific type of empirical study: controlled experiments. First, we elicited a set of relevant

query candidates in a survey with EMSE BoK researchers. Then, we modeled BoK topics related to software inspection theory and a common data model for controlled experiment data on those topics, defining the relevant information to extract. Following SKE's search process, 102 software inspection experiments were identified. An independent team extracted information from the 30 most recent papers for data import and query resolution by a knowledge engineer.

Major findings were: (1) SKE was effective in identifying and characterizing inspection experiments and their results, (2) it was possible to build the software inspection KB from published experiments, and (3) the KB was effective in providing answers to the required queries. Additionally, we compared the SKE and SLR processes, when applied to building EMSE BoKs. In this context, the SKE KB facilitates reusing and exploring the extracted data based on semantic search capabilities not available for SLR reports. While the data extraction effort is comparable, SKE requires a knowledge engineer for data modeling, mapping, and providing query facilities. This overhead is offset by benefits of the KB, which can be used among researchers in and beyond a work group to incrementally build EMSE BoKs.

The remainder of this paper is organized as follows. Section 2 summarizes related work. Section 3 motivates the research issues. Section 4 introduces SKE. Section 5 presents the evaluation. Section 6 discusses results and lessons learned. Section 7 concludes the paper.

## II. RELATED WORK

The quality and speed of building up a body of knowledge (BoK) in an EMSE research area depend on the ability of researchers to discover the existing content in a BoK, e.g., empirical studies investigating a set of hypotheses or variables. Currently, online searching for content is supported by syntactic full-text search on specialized databases, such as digital libraries. However, support for semantic searching is limited and researchers may not discover all relevant content.

Although some effort has been spent on repositories for empirical studies (e.g., *CeBASE* [9] and *ViSEK* [10]), they did not show significant progress since their introduction. To our knowledge, there is no related work on using SLR-based study identification and KB integration for bottom-up EMSE BoK building to facilitate reuse and semantic search. Therefore, this section describes work related to the theoretical foundations of this research's main constructs: Systematic Literature Reviews (Section II.A) and Knowledge Base Design & Population (Section II.B). Further, we describe the foundations on Software Inspections (Section II.C) as basis for the evaluation use case.

### A. Systematic Literature Reviews

Kitchenham and Charters [8] developed guidelines for performing Systematic Literature Reviews (SLRs) in the software engineering (SE) domain. Those guidelines state that the main reasons for conducting SLRs are (a) summarizing existing evidence concerning a treatment or technology; (b) identifying gaps in current research; and (c) providing background to appropriately position new research activities. The first reason is directly related to building BoKs by gathering evidence-based knowledge. In the context of this research, the main advantage of using SLRs is allowing systematically summarizing

knowledge on a specific BoK scope and enabling incremental updates on top of previous SLRs. An example of such updates is available in [11], where four independent SLR trials were conducted to incrementally build evidence-based defect causal analysis guidelines. Lessons learned from applying SLRs to the SE domain are reported by Brereton et al. [4].

The SLR guidelines [8] summarize three main phases of a systematic review: Planning the Review, Conducting the Review, and Reporting the Review. The stages related to each of those phases are: (a) Planning the Review: identification of the need for a review, commissioning a review, specifying the research questions, and developing a review protocol; (b) Conducting the Review: identification of research, selection of primary studies, study quality assessment, data extraction and monitoring, and data synthesis; and (c) Reporting the Review: specifying dissemination mechanisms, formatting the main report, and evaluating the report. The PICO (population, intervention, comparison, outcome) strategy [12] is suggested [8] for detailing the research question elements in order to support developing the review protocol.

SLRs have become a widely used and reliable research method [5]. However, the main public result of a SLR is, in general, a specific research synthesis report by the authors. Unfortunately, reusable SLR packages that include the working material, which holds the accumulated knowledge, are available only seldom. The working material includes the data extracted from the primary studies (commonly stored in spreadsheets). Therefore, new SLRs have to rebuild large parts of existing knowledge, making the addition of knowledge less efficient than necessary. Even if such knowledge were available, data integration mechanisms to enable making the knowledge available for further use by other EMSE BoK researchers have not been discussed in this context. In summary, the reuse value of SLR knowledge to help building EMSE BoKs is limited.

### B. Knowledge Base Design and Population

The process of building a KB may be seen as a modeling activity [13]. For creating a KB, it is essential to capture domain knowledge through content-specific agreements, so both human and knowledge-based systems can access and use the information [13]. For this purpose, formal ontologies have been successfully used since the 1990s [14]. Ontologies can provide standard terminologies and rich semantics to facilitate knowledge sharing and reuse [13]. OWL DL (*Web Ontology Language - Description Logic*) is the most used language for ontologies as it has the capability of supporting semantic interoperability to exchange and share context knowledge between different systems, and keeps a balance between expressiveness and automated processing. In addition, ontology enhances searching mechanisms, which may refer to precise semantic concepts rather than simple syntactic keywords, facilitating the use of the knowledge stored in the ontology [15].

Many methodologies have been proposed to design ontologies [16]. However, only a few of them consider collaborative and distributed construction of ontologies, such as the Collaborative Design Approach [17], which addresses the issue of collaborative construction of the ontologies by identifying and involving a diverse panel of participants.

Once the ontology or the data model of the KB is defined, it is necessary to capture the extracted data from information resources in accordance to the KB. This process is called KB population, and involves the creation, transformation and integration of individuals (instances) into the KB. In our case, the information resources for creating the KB are empirical study reports. The KB population process may face integration problems if the information resources use heterogeneous representations of the same concepts. The Interchange Standard Approach has been recommended as one of the best solution options for semantic integration [18]. Currently available tools to manage ontologies usually require ontology experts. Therefore, ontology non-experts need effective and efficient interfaces for both, importing and exporting knowledge, and for querying.

### C. Software Inspections

Software inspections (SI) improve software quality by the analysis of software artifacts, detecting defects for removal before these artifacts are delivered to following software life cycle activities [1]. The traditional SI process by Fagan [19] involves a moderator planning the inspection, inspectors reviewing the artifact, a team meeting to discuss and register defects, passing the defects to the author for rework, so they can be corrected, and a final follow-up evaluation by the moderator on the need of a new inspection.

Over the years, many contributions on SI have been proposed, including alternative SI processes, SI methods to improve the effectiveness and efficiency of inspectors in detecting defects, models and guidelines supporting tasks of the inspection process that involve decision making, and tool support [20]. Much knowledge on those contributions has been acquired by conducting empirical studies and can be considered part of a BoK in the SI area. However, such knowledge is currently not organized in the context of a BoK. Therefore, it still takes considerable expertise and effort to identify studies and study results relevant for a given topic in the scope of SI. To support the SI community in building up their BoK, specific BoK topics can help to define the scope of knowledge. The IEEE Software Engineering BoK (SWEBoK) [21] breaks down the Testing BoK into the following topics: *Fundamentals*, *Test Levels*, *Test Techniques*, *Test-Related Measures*, *Test Process*, and *Software Testing Tools*. SI relates to *Test Techniques* in the IEEE SWEBoK. Nevertheless, they can be seen as a similar topic of interest and a hierarchical structure for them could be useful to facilitate organizing knowledge.

A fixed BoK topic structure may be limiting, since it is possible to apply several variant options of SI. Laitenberger and DeBaud [22] provide some parameters that help define SI variant options based on an early literature review on SI experiments: SI artifacts (e.g., requirements, design or code), SI process (e.g., SI with or without a group meeting), and SI methods (e.g., reading techniques). Such a list of parameters could be used as a starting point for a SI EMSE BoK topic structure.

## III. RESEARCH ISSUES

The overall SKE goal is to enable incrementally building up an EMSE BoK by providing a process for knowledge acquisition and querying. Therefrom we derive three research issues.

*Research Issue RI-1: SKE Requirements Analysis.* Which queries to knowledge on empirical studies are most relevant to EMSE BoK stakeholders?

EMSE researchers want to synthesize EMSE BoKs but tend to focus on conducting empirical studies and seem to spend much less effort on considering data management to provide their EMSE BoK community with suitable access to the created knowledge [7]. Building on typical hypothesis patterns reported by Sjöberg et al. [2] we conducted an informal survey with EMSE BoK researchers from six active work groups on queries to knowledge on empirical studies that they need for their work. We are aware of the limitations of this survey and see the survey outcome as a preliminary working result, which is still useful to drive the SKE research at this stage, and can be extended by a future more formal survey in a wider scope.

*Research Issue RI-2: SKE Process and Data Modeling.* How can the traditional SLR process be adapted to support incrementally building EMSE BoKs? Which data elements are necessary to address the most relevant queries of EMSE BoK researchers?

The SKE process builds on the traditional SLR process [8] and on the Collaborative Design Approach (CDA) [17] for knowledge engineering. Key idea is to loosen the tight connection between SLR data extraction and data synthesis in order to allow collecting knowledge from EMSE studies in a KB, as input to a range of research synthesis methods in a BoK community. Second key goal is to enable incrementally building up knowledge in the context of an EMSE BoK. We evaluate the resulting SKE process regarding effort and added benefits.

SKE aims at designing a common KB data model that captures both, concepts from the BoK domain (e.g., software inspection), and from the selected types of EMSE studies (e.g., controlled experiments). We evaluate the SKE data model regarding the effectiveness to answer the queries of EMSE BoK researchers identified in RI-1.

*Research Issue RI-3: SKE Tool Support.* Which functions are necessary to automate key steps in the SKE process, i.e., efficient data integration and querying?

For automating SKE a knowledge base (KB) is a major element to provide the desired semantic capabilities. The SKE KB is based on semantic technology with ontologies. Using ontologies makes the KB model extensible and facilitates semantic search [15]. Based on the requirements coming from RI-1 and RI-2 and discussions with SLR researchers the user interface for data import should be based on spreadsheet technology and for querying based on web technologies. We evaluate the tool support regarding the effectiveness and efficiency of data import and querying when applying the SKE process.

## IV. SYSTEMATIC KNOWLEDGE ENGINEERING

This section presents the SKE process and its application to build a Software Inspection EMSE BoK KB based on contributions from controlled experiments. The following subsections provide details on how each research issue (requirements analysis, SKE process and data modeling, and SKE tool support) was addressed.

### A. SKE Requirements Analysis (RI-1)

Fig. 2 shows how SKE addresses the challenges posed in Fig. 1 by introducing a KB and the role of a knowledge engineer. In this context, researchers extract data from empirical studies published in digital libraries and have that extracted data integrated into the KB by the knowledge engineer. Therefore, the knowledge collected is then available for semantic querying from the KB also to the general readership, including other researchers and practitioners.

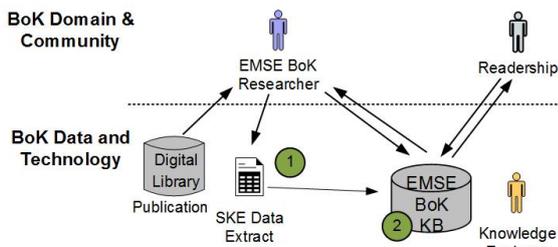


Figure 2. EMSE BoK stakeholders and technology with SKE.

To identify the most relevant query candidates to be answered by the software inspection EMSE BoK KB, we focused on EMSE BoK researchers as main stakeholders, who conduct meta-analyses on study reports or conduct empirical studies and need to be aware on relevant research in their area.

Based on an informal survey with software inspection EMSE BoK researchers in six research groups (located in Austria, Brazil, Chile, Ecuador, and Spain), we identified a set of query candidates. The selection of most relevant queries was based on a limited budget of value points, which each stakeholder could spend on the query candidates. Then the query candidates were sorted in descending order by the total number of points of each query. Overall, 10 researchers from the 6 research groups contributed to this selection process. The six most relevant stakeholder queries were:

- Q1: Which inspection methods were effective (or efficient) in finding defects in requirements artifacts?
- Q2: What are the results of experiments that report on a given BoK Topic Parameter <BTP>, e.g., inspection method PBR?
- Q3: Which experiments were conducted with the response variable <RV>, e.g., number of defects?
- Q4: Which hypotheses include the domain concept <DC> (and its synonyms), e.g., effectiveness?
- Q5: Which synonyms have been used for domain concept <DC>, e.g., effectiveness?
- Q6: Who are researchers working on topics with response variables in their experiments similar to the domain concept <DC>, e.g., efficiency?

### B. SKE Process and Data Modeling (RI-2)

Fig. 3 compares the phases and stages of the SLR [8] and SKE processes. The key innovation of the SKE process comes from decoupling data extraction from data synthesis and integrating extracted data into a KB designed for EMSE BoK building, rather than using the data to apply a particular synthesis method for answering a specific research question in the format of a SLR report (keeping the extracted data from the BoK research community). Thus, the approach allows the community

to extend the knowledge gathered during data extraction and reusing it, with the KB’s semantic search facilities, as building blocks for a variety of analyses.

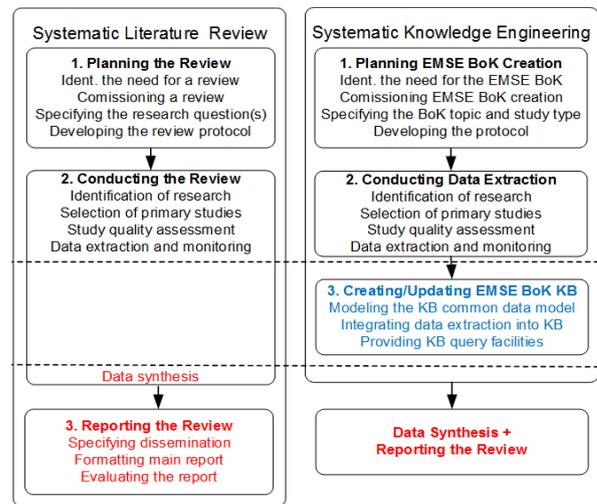


Figure 3. Comparison of the SLR and the SKE processes.

Details on each of the three SKE process stages and how they could be applied to build the software inspection EMSE BoK KB follow.

1) *Planning EMSE BoK Creation.* Similar to conducting a SLR, the first SKE phase starts with identifying the need and commissioning the creation of the EMSE BoK. Since SKE has a pre-defined purpose of building an EMSE BoK, instead of specifying research questions, SKE just needs to specify the BoK (topics) and the types of empirical studies of interest. In the case of building the inspection EMSE BoK, the BoK was software inspection and the empirical studies of interest were controlled experiments. Then, the SKE protocol is built based on a specific configuration of the PICO (population, intervention, comparison, outcome) strategy [12] to derive search strings that can be applied to digital libraries in the “P and I and C and O” format. In this configuration, the population represents the specified BoK or some of its topics, in our case Software Inspection. The intervention represents the specified empirical study types, in our case Controlled Experiments. The comparison is blank and the outcome represents the elements to extract from the empirical studies (e.g., hypotheses, findings), in our case experimental study results.

As in SLRs, the protocol includes the search strategy with the definition of sources of primary studies (e.g., digital libraries), the study selection criteria and procedure, the quality assessment procedures, and the data extraction strategy. In our case a single digital library was chosen: Scopus, which claims to be the largest database of abstracts [8] and seems sufficient for our study purpose. The study selection and quality assessment criteria were: the study should be an experiment published in a peer-reviewed publication medium. The search string to be applied on Scopus was derived from the PICO synonyms, adding two specific operators: (i) TITLE-ABS-KEY, avoiding searching in the reference metadata; and (ii) W/2, allowing a distance

of up to two words between keywords. The resulting search string was:

*TITLE-ABS-KEY ((software W/2 (inspection OR "defect detection" OR "reading technique")) AND ("experimental study" OR "experimental evaluation" OR "experiment" OR "empirical study" OR "empirical evaluation")) AND ("hypothesis" OR "evidence" OR "finding" OR "result"))*

If there is already a BoK KB to build on, the search concepts, including synonyms, can be derived from the BoK KB.

For data extraction, a spreadsheet was prepared to gather relevant experiment data, according to the information to be loaded into the KB common data model on inspection experiments. Once the protocol is defined, the next SKE phase, Conducting Data Extraction, can be accomplished.

2) *Conducting Data Extraction.* This phase consists of following the protocol's search, selection, and assessment strategies for extracting relevant data. In our case, we executed the search string in Scopus. Overall 156 papers were retrieved. After filtering by title and abstract, a set of 102 papers containing experiments on software inspections were identified, ranging from 1985 to 2013.

A sample consisting of the 30 most recent papers (ranging from 2006 to 2013) was chosen as criteria for data extraction. Six independent local EMSE experts extracted information from those papers into spreadsheets, with an extra expert acting as a data checker. Data extraction took on average about 2 person hours per paper. Data checking took additional 0.5 person hours per paper. As data synthesis is in SKE decoupled from data extraction, the goal is integrating the data into a KB so the BoK community can reuse the knowledge. Thus, the next SKE phase concerns creating the EMSE BoK KB.

3) *Creating/Updating EMSE BoK KB.* In this phase, the knowledge engineer (KE) has to design (or update) the KB common data model and then integrate the extracted data. This role is also responsible for providing query facilities.

For building the software inspection EMSE BoK KB, EMSE experts and the KE designed the data model applying CDA [17] and derived the data extraction spreadsheets. Then the KE provided the KB query facilities to address the most relevant stakeholder queries. More details on the KB common data model for hosting information on software inspection experiments follow.

*Data Modeling for an EMSE BoK KB.* Fig. 4 shows a high-abstraction view on the context in which empirical studies are conducted. Empirical studies have data and artifacts, contribute to a Body of Knowledge (BoK) on specific topics, and are performed by researchers who provide publications. Fig. 5 shows the entities for hosting data on experiments in UML (see [23] for UML-to-OWL transformation), based on these areas and on experimental concepts described by Wohlin et al. [3].

To organize the aggregated knowledge in a flexible way and link knowledge from the empirical studies (in this case, experiments) to inspection BoK topics, each topic was designed as relating to a set of inspection parameters, extended from the list of parameters by Laitenberger and Debaud [22]. For instance,

knowledge on requirements (artifact) inspections applying PBR (inspection method) during individual preparation (activity) when conducting Fagan inspections (process). This tailoring is shown in Fig. 6. Note that this final step has to be redone for BoKs in other research topics.

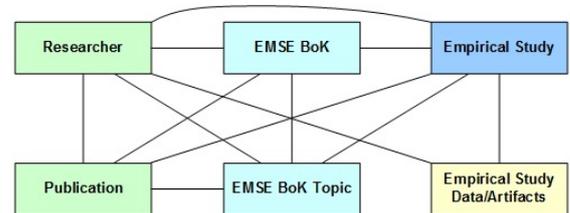


Figure 4. Major areas of the SKE data model.

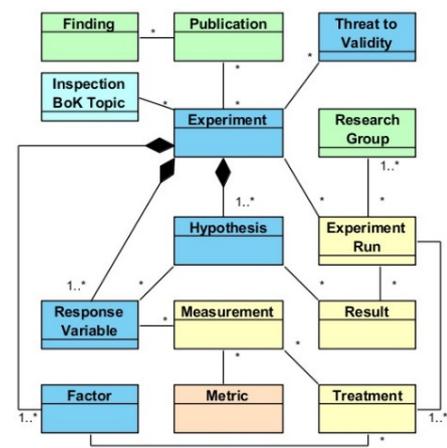


Figure 5. KB data model overview.

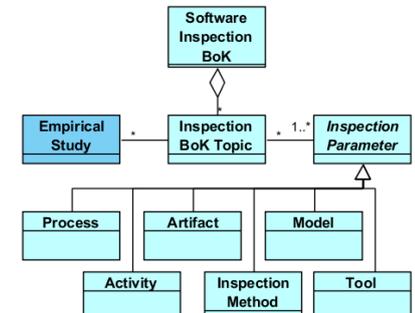


Figure 6. Empirical studies linked to inspection BoK topics.

The resulting model allows querying knowledge acquired from experimental studies. Such queries can list hypotheses of experiments related to specific BoK topics parameters (or their synonyms), the results for each hypothesis in the available experiment runs (confirmed/rejected) and information on their statistical confidence. Moreover, measurements that led to each of those results can also be obtained.

C. *SKE Tool Support (RI-3)*

The KB was implemented using the Protégé framework and uses semantic technology with ontologies to facilitate semantic searches. Besides the KB, the tool support comprises a spreadsheet data contribution interface and a web prototype for querying. The data contribution interface was automated in Java by using a spreadsheet reader library (Apache POI) and an ontology library (Apache Jena). The Interchange Standard Approach [18]

can be applied for integrating heterogeneous data. The queries of the web prototype were implemented using SPARQL query language. Using ontology-specific features, the knowledge engineer enhanced the KB by implementing semantic search functions (e.g., searching on domain concepts, their synonyms and related concepts). Additionally, a glossary tool to facilitate identifying and defining domain concepts was also designed and implemented.

The material related to this Section, including the list of queries, the complete data model, the SKE online query prototype and the glossary tool is available online<sup>1</sup>.

## V. EVALUATION RESULTS

This section reports on the evaluation of the results regarding RI-2 and RI-3 based on the required queries coming from RI-1.

### A. SKE Process Evaluation

For evaluation, we applied and measured the SKE process steps with tool support to build a software inspection EMSE BoK KB based on knowledge acquired from controlled experiments. Information was extracted from 30 typical research papers and integrated into the KB. While the authors of this paper provided the SKE process and data model design, the data extraction was conducted by an independent expert team.

1) *SKE Process Feasibility.* The feasibility of applying each SKE phase (planning, data extraction, and building the EMSE BoK KB) was evaluated. The planned SKE PICO configuration effectively supported the identification of relevant experiments on software inspections. The accuracy of the derived search string was evaluated by comparing its results against the software inspection review described in [24]. The experiments reported in this review and indexed in Scopus were successfully retrieved. Therefore, we see the identified papers also representative for new inspection experiments.

Data extraction into spreadsheets containing information on software inspection experiments based on the common KB data model was successfully achieved. Thus, the common data model built for software inspection experiments was effective in characterizing the inspection experiments (modeling similarities and variations) and their results.

Finally, concerning EMSE BoK KB creation, it was possible to create the software inspection KB from published experiment reports. A knowledge engineer conducted the integration of the data from the local spreadsheets into the common KB data model. The resulting EMSE BoK KB was effective and efficient in providing answers to the required queries.

2) *SKE Process Effort.* Based on our previous SLR experiences, we found the overall effort of applying the SKE process comparable to the effort of conducting SLRs (around 90 person hours). However, the effort of extending the SKE results is likely to be considerably lower (especially if done by other researchers, by allowing directly reusing and extending the extracted data, currently in many SLRs not publicly available). Table 1 shows the effort spent on each SKE phase to

build the software inspection KB and an informal effort comparison to the corresponding SLR phase. Planning takes slightly less effort, since SKE applies a predefined PICO configuration. Although in SKE data extraction from primary studies probably takes somewhat more effort than for the SLR, the overall conduct phase takes about the same effort, since SKE does not apply data synthesis in this phase. Finally, creating the EMSE BoK KB is not considered in SLRs.

TABLE I. SKE INSPECTION KB PROCESS EFFORT.

SKE Effort (person hrs)	Effort Description	SKE vs SLR	
Planning EMSE BoK Creation	8	Building the protocol.	↓ (-10% to -5%)
Conducting Data Extraction	80	Filtering primary studies and data extraction.	↔ (-5% to +5%)
Creating EMSE BoK KB (Population)	< 0.05	KB data integration (automated).	N/A

3) *SKE Process Added Value.* We are aware that SKE and SLR processes have different purposes (SLRs usually seek for evidence-based answers to research questions, while SKE focuses on building EMSE BoKs) and that one does not replace the other. However, since SLRs are widely used [5], we ascertain that, given the similar effort, it makes sense to apply SKE for building EMSE BoKs. Moreover, in this specific context, when compared to SLRs, SKE presents the following benefits:

- SKE integrates extracted data into a KB and facilitates the reuse of the aggregated knowledge by other researchers according to their specific goals. SLRs usually focus on specific research syntheses and extracted data is mostly stored in local spreadsheets, seldom publicly available.
- SKE facilitates building up knowledge incrementally by integrating new extracted data into the KB.
- The SKE KB allows exploring the gathered empirical evidence using semantic search capabilities that cannot be performed on SLR reports. For instance: “Which results were obtained for hypotheses investigated in BoK topic <BT> using the response variable <RV> (or any of its synonyms)?”

Note that SKE required some setup effort by the knowledge engineer for creating the KB’s ontology model, creating the spreadsheet importer, providing the query facilities, and developing a suitable user interface. Table 2 shows this setup effort.

TABLE II. KNOWLEDGE ENGINEER EFFORT.

Know. Eng. Effort (person hrs)	Effort Description	
Creating EMSE BoK KB ontology model	16	Specifying the ontology model based on the common data model.
Spreadsheet Importer Creation	40	Developing the spreadsheet importer tool based on Apache Jena and POI.
Query creation	32	Translating queries into SPARQL.
UI Development	40	Developing a querying user interface.

### B. SKE Data Model Evaluation

Having evaluated the feasibility of applying the SKE process, interest sprouts in evaluating the resulting software inspection EMSE BoK KB and its underlying common data model against the required queries coming from RI-1. Therefore, the

<sup>1</sup> SKE Prototype: <http://cdflex.org/prototypes/ske>

knowledge engineer formulated the queries in the KB language so that results for them could be obtained. To evaluate these results, an independent researcher built a set of query test cases based on the 30 papers included in the KB. Queries Q2 to Q6 could be directly formulated, listing experiment results, experiments, hypotheses, synonyms, and research groups. The related test cases passed successfully with the query results, which can be accessed in the online prototype<sup>1</sup>.

However, query Q1: “Which inspection methods were effective (or efficient) in finding defects in requirements artifacts?,” addressing a practitioners’ need could not be answered directly and had to be translated into terms of the underlying data model. The first decision was to focus on experiments that reported on effectiveness or efficiency (or synonyms) in their hypotheses or response variables and that were related to BoK Topics associated to inspection methods and to the artifact type requirements. For allowing semantic search on synonyms the additional domain concepts KB technology and the glossary were used. The test case built for this query was related to identifying the right set of experiments and passed successfully. Then, to answer the query, it was split into two separate queries that would provide an overview on the knowledge of interest out of those experiments. Q1.1 was to list the hypotheses and their results in all experiment runs. Q1.2 was to show the findings of all papers related to them.

Fig. 7 shows results of Q1.1 enabling stakeholders to see which inspection methods were reported as effective. Stakeholders can also see other focused and interesting information. For instance, that defects detected with PBR are more evenly distributed over the document, when compared to using checklists. Regarding Q1.2, it provided an insightful overview on findings of the papers on effectiveness of inspection methods. Thus, the evaluation of all required queries passed successfully in the scope of the SKE research prototype.

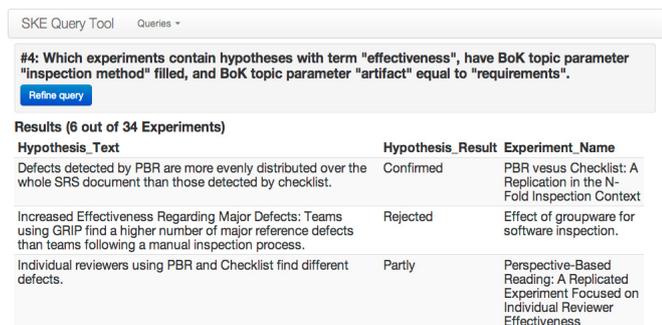


Figure 7. Prototype screenshot answering the query Q1.1.

### C. SKE Tool Support Evaluation

This evaluation concerned the support for data importing and querying. The spreadsheet data contribution interface was effective in enabling contributions from researchers and efficient by requiring low effort and little time for such contributions to be imported. Data from 6 spreadsheets, containing data on 30 experiments and including over 6,000 data elements (cells) were integrated into the KB in less than 3 minutes. The simple web interface prototype allows stakeholders to easily retrieve query results within a few seconds.

## VI. DISCUSSION

In this section, we discuss for each research issue the evaluation results, possible limitations, and lessons learned.

### A. RI-1: SKE Requirements Analysis

The SKE process and its resulting KB support users well in locating empirical evidence and reusing it according to specific needs. However, it is important to state that support for meta-analyses and for applying specific research syntheses methods are beyond the scope of SKE. Moreover, the relevant queries of our evaluation were chosen focusing on a survey with a specific type of stakeholder, the EMSE BoK researchers. Other stakeholders may have different information needs. Jedlitschka et al. [25], for instance, gathered empirical study information needs from 175 managers from industry. According to their findings, there are three categories of information needs for these stakeholders from experiment reports: technology, context, and impact. We believe that these information needs could be provided by semantic queries applied to the SKE KB. However, addressing practitioners’ needs still has to be evaluated.

### B. RI-2: SKE Process and Data Modeling

The SKE process evaluation showed the feasibility of building a software inspection EMSE BoK KB from controlled experiments. Moreover, the overall effort was comparable to the effort of conducting a SLR and, for the specific purpose of building EMSE BoKs, several advantages of using SKE could be confirmed. However, it is important to state that the SKE and SLR processes have different purposes. The suitability of applying one or the other will depend on the specific research goals. In fact, if an EMSE BoK is built following SKE, the resulting KB may be used as input to an SLR protocol to identify relevant research on a specific topic, exploring semantic search facilities usually not available in digital libraries. SLR data extraction sheets, on the other hand, can be used as input for extending the knowledge contained in the SKE KB.

### C. RI-3: SKE Tool Support

The simple user interface for data importing and querying enables researchers to contribute building up additional knowledge, and to query for knowledge with low effort. Using ontologies facilitates extensions of the underlying KB common data model and semantic search. The querying capabilities were found efficient in the evaluation on answering the EMSE BoK researchers’ most relevant queries.

### D. Threats to Validity and Lessons Learned

The performed evaluation faces some threats to validity. A major threat is related to the decision of applying the approach on building a software inspection EMSE BoK based on study reports from controlled experiments. Software inspections are widely spread in academia and industry and many empirical studies were conducted in this area, which may have facilitated building up the EMSE BoK. Additionally, SKE can also be applied to gather knowledge from other types of empirical studies but the feasibility of this process needs to be investigated. Besides, the relevant queries used to evaluate the KB were obtained from a limited and informal survey.

Main lessons learned relate to success factors for applying SKE. A major success factor is properly involving a knowledge

engineer. The overhead of this new role is likely to be offset soon by benefits obtained by the established KB. Another success factor is getting the EMSE BoK community involved for long-term collection and use of data. Therefore, incentives and benefits of contributing should be clarified.

## VII. CONCLUSIONS

In this paper we introduced the Systematic Knowledge Engineering (SKE) process, which supports building up EMSE BoKs from empirical studies. SKE builds on the Systematic Literature Review (SLR) process and provides a Knowledge Base (KB) as storage for extracted data. By decoupling data extraction from data synthesis and providing a KB, SKE allows the community to extend gathered knowledge (even if the original authors are no longer available) and reusing it with semantic search facilities, as building blocks for a variety of analyses.

SKE was evaluated by building a software inspection EMSE BoK from knowledge acquired through controlled experiments. Information was extracted from 30 research papers and integrated into the KB. The resulting software inspection EMSE BoK is available online<sup>1</sup>. Main evaluation results were:

- SKE's suggested PICO configuration was successful in identifying relevant experiments on software inspections.
- Data extraction of inspection experiments into spreadsheets based on the common data model was successful.
- It was possible to create the software inspection EMSE BoK KB from published experiment reports.
- The KB was effective and efficient in answering the most relevant stakeholder queries.
- SKE enables knowledge reuse (by applying queries) for analysis and meta-analysis purposes. Moreover, new knowledge, i.e., new data from literature, can be included in the KB as a foundation for a growing EMSE BoK.

The SKE process showed promising results in the software inspection context and should also be evaluated in other contexts. The overall effort of applying SKE is comparable to the effort of conducting SLRs and, for the specific purpose of building EMSE BoKs, several advantages of using SKE could be identified. As future work we propose: (a) investigating to address different EMSE BoK stakeholder needs; (b) extending the set of empirical studies on software inspections in the KB; (c) instantiating SKE in other contexts; and (d) setting up a platform to allow building on the collective intelligence of the research community for quality assurance & recommendation.

## ACKNOWLEDGMENT

This work was supported by the Christian Doppler Forschungs-gesellschaft, the Federal Ministry of Economy, Family and Youth and the National Foundation for Research, Technology and Development - Austria, and CNPq - Brazil.

## REFERENCES

[1] B. Boehm and V. R. Basili, "Software Defect Reduction Top 10 List," *IEEE Computer*, vol. 34, no. 1, pp. 135–137, 2001.

[2] D. I. K. Sjøberg, T. Dybå, B. C. D. Anda, and J. E. Hannay, "Building Theories in Software Engineering," in: *Guide to Adv. Emp. Soft. Eng.*, F. Shull, J. Singer, and D. K. Sjøberg, Eds. Springer, pp. 312–336, 2008.

[3] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, "Experimentation in Software Engineering," Springer, 2012.

[4] P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, and M. Khalil, "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, Apr. 2007.

[5] S. MacDonell, M. Shepperd, B. Kitchenham, and E. Mendes, "How Reliable Are Systematic Reviews in Empirical Software Engineering?," *IEEE Trans. on Soft. Eng.*, vol. 36, no. 5, pp. 676–687, Sep. 2010.

[6] D. S. Cruzes and T. Dybå, "Research synthesis in software engineering: A tertiary study," *Inf. Soft. Tech.*, vol. 53, no. 5, pp. 440–455, May 2011.

[7] S. Biffl, E. Serral, D. Winkler, O. Dieste, N. Juristo, and N. Condori-Fernández, "Replication Data Management: Needs and Solutions - An evaluation of conceptual approaches for integrating heterogeneous replication study data," *Int. Symp. on Emp. Soft. Eng. and Meas.*, 2013.

[8] B. Kitchenham and S. Charters, "Guidelines for performing Systematic Literature Review in Software Engineering," *Keele University Technical Report - EBSE-2007-01*, 2007.

[9] B. Boehm and V. Basili, "The cebase framework for strategic software development and evolution," in: *Int. Workshop on Economics-Driven Software Engineering Research (EDSER)*, 2001.

[10] B. Hofmann and V. Wulf, "Building Communities among software engineers: the VISEK approach to intra-and inter-organizational learning," *Advances in Learning Soft. Org.*, pp. 25–33, 2003.

[11] M. Kalinowski, D. N. Card, and G. H. Travassos, "Evidence-Based Guidelines to Defect Causal Analysis," *IEEE Software*, vol. 29, no. 4, pp. 16–18, Jul. 2012.

[12] M. Pai, M. McCulloch, J. D. Gorman, N. Pai, W. Enanoria, G. Kennedy, P. Tharyan, and J. M. Colford, "Systematic reviews and meta-analyses: an illustrated, step-by-step guide," *The National Medical Journal of India*, vol. 17, no. 2, pp. 86–95, 2004.

[13] R. Studer, V. R. Benjamins, and D. Fensel, "Knowledge engineering: Principles and methods," *Data & Knowledge Engineering*, vol. 25, no. 1–2, pp. 161–197, Mar. 1998.

[14] T. R. Gruber, "Toward principles for the design of ontologies used for knowledge sharing?," *International Journal of Human-Computer Studies*, vol. 43, no. 5–6, pp. 907–928, Nov. 1995.

[15] F. J. Ekaputra, E. Serral, D. Winkler, and S. Biffl, "An Analysis Framework for Ontology Querying Tools," in *9th International Conference on Semantic Systems*, 2013, pp. 1–8.

[16] O. Corcho, M. Fernández-López, and A. Gómez-Pérez, "Methodologies, tools and languages for building ontologies. Where is their meeting point?," *Data & Kn. Eng.*, vol. 46, no. 1, pp. 41–64, Jul. 2003.

[17] C. W. Holsapple and K. D. Joshi, "A collaborative approach to ontology design," *Com. of the ACM*, vol. 45, no. 2, pp. 42–47, Feb. 2002.

[18] N. F. Noy, "Semantic integration: a Survey of Ontology-Based Approaches," *ACM SIGMOD Record*, vol. 33, no. 4, p. 65–70, Dec. 2004.

[19] M. E. Fagan, "Design and code inspections to reduce errors in program development," *IBM Systems Journal*, vol. 15, no. 3, pp. 182–211, 1976.

[20] M. Kalinowski and G. H. Travassos, "A computational framework for supporting software inspections," in: *International Conference on Automated Software Engineering (ASE)*, 2004, pp. 46–55.

[21] A. Abran, J. Moore, P. Bourque, R. L. Dupuis, and L. Tripp, *Software Engineering Body of Knowledge-SWEBOK*, trial version. IEEE-Computer Society Press, 2001.

[22] O. Laitenberger and J.-M. DeBaud, "An encompassing life cycle centric survey of software inspection," *Journal of Systems and Software*, vol. 50, no. 1, pp. 5–31, Jan. 2000.

[23] A. Grünwald and T. Moser, "umlTUowl - A Both Generic and Vendor-specific Approach for UML to OWL Transformation," in: *Int. Conf. on Soft. Eng. and Know. Eng. (SEKE)*, 2012, pp. 730–736.

[24] A. Aurum, H. Petersson, and C. Wohlin, "State-of-the-art: software inspections after 25 years," *Software Testing, Verification and Reliability*, vol. 12, no. 3, pp. 133–154, Sep. 2002.

[25] A. Jedlitschka, N. Juristo, and D. Rombach, "Reporting experiments to satisfy professionals' information needs," *Emp. Soft. Eng.*, Aug. 2013.

# A Body of Knowledge for Executing Performance Analysis of Software Processes

Natália Chaves Lessa Schots, Ana Regina Rocha  
COPPE/UFRJ  
Universidade Federal do Rio de Janeiro  
Rio de Janeiro, Brazil  
{natalia, darocha}@cos.ufrj.br

Gleison Santos  
PPGI/UNIRIO  
Universidade Federal do Estado do Rio de Janeiro  
Rio de Janeiro, Brazil  
gleison.santos@uniriotec.br

**Abstract**—The periodic execution of process performance analysis represents an important step for an organization to achieve quality in their processes and products. However, it is possible to notice that few software development organizations can establish and maintain this practice effectively. Many of the difficulties in executing performance analysis are related to the lack of knowledge in the field and difficulties of learning that are due to limitations and dispersion of currently available literature. In this sense, this paper presents the definition of a body of knowledge in process performance analysis, which will be made available through a knowledge-based system, aiming to support the execution of performance analysis in software development organizations. Particularly, we highlight the description of which knowledge the system must provide, as well as the ways how this knowledge is being captured and presented. For helping the understanding of the body of knowledge, an example of use is illustrated.

**Keywords**- *Process Performance Analysis; High Maturity Practices; Body of Knowledge; Knowledge-based Environment; Knowledge Management*

## I. INTRODUCTION

In software development organizations that are looking for improvement in their processes and products, the adoption of techniques for process performance analysis is essential, since these techniques allow them to learn about their process, as well as control and predict them through the process metrics and statistical techniques [1].

There are many works in the literature that discuss the application and benefits of using process performance analysis in software, e.g. [2][3][4]. However, it is possible to see that few software development organizations adopt this practice in their organizational culture. An evidence of this scenario is the low number of evaluated software organizations in CMMI levels 4 or 5; according to [5], only 8% of the organizations evaluated are/were on these levels. Furthermore, few works in the literature show the effective implementation of process performance analysis in the software field [6].

The main difficulties pointed by the organizations for not implementing process performance analysis are: (i) lack of planning and collection of adequate software metrics; (ii) lack of knowledge on the techniques and methods to execute process performance analysis; and (iii) lack of knowledge about the data required to adequately perform the analysis [3][6][7].

As it can be seen in the literature reviews and in practice through consulting experience, process performance analysis is a human-dependent activity, as it requires several decision-making actions that cannot be completely solved by a computer. Due to this, knowledge becomes a critical success factor [8]. These decision-making actions require deep technical knowledge of performance analysis and a good understanding of the process being analyzed.

In fact, for an effective process performance analysis, it is mandatory for those responsible for this activity to have technical knowledge of the concepts and techniques to carry out performance analysis; one should also have knowledge of both the process and the organizational context in which the analysis will be performed, as otherwise the benefits of the analysis may not be achieved. As it can be seen, knowledge is one of the most important assets in an organization that seeks for excellence in the development of its products and services [1]. Thus, it must be properly managed, so that such organization can take advantage from its several benefits.

In this sense, knowledge management practices can play an important role to support this kind of activity by collecting, storing and applying the right knowledge to the right people at the right time. In order to support performance analysis by using knowledge management practices, a knowledge-based environment named SPEAKER (Software Process pErformance Analysis Knowledge-based EnviRonment) is being developed. This environment aims to provide a body of knowledge on the concepts, tasks, and techniques of process performance analysis, designed to support software development organizations to perform this analysis properly.

Thus, this paper aims to describe the body of knowledge that will be made available through the proposed environment, as well as how this environment must deal with the management of such knowledge. To this end, the paper aims to answer the following questions: (Q1) which knowledge should be made available for supporting the execution of process performance analysis? and (Q2) how this knowledge should be documented in SPEAKER?

The remainder of this paper is structured as follows: Section II presents basic concepts about performance analysis and knowledge management. In Section III, the body of knowledge for software process performance analysis is described, and a hypothetical example of its use is presented in

Section IV. Some considerations on how the proposed knowledge-based environment will manage this knowledge are presented in Section V. Section VI presents some related works. Finally, Section VII provides some final remarks.

## II. BACKGROUND AND MOTIVATION

Process performance analysis involves a set of methods and techniques first targeted at understanding the process and then analyzing its stability [1]. The execution of process performance analysis is one of the requirements defined by maturity models (such as CMMI-DEV [9] and MR-MPS-SW (the Reference Model of the Brazilian Software Process Improvement Program for Software) [10]) and quality standards (such as ISO 15504 [11]).

In order to use measures to predict future results or as a basis for process improvement, it should be ensured that the process behavior is stable [12]. A process is stable when its variability is due only to common causes, i.e., caused by inherent factors that are part of the process. If the process is unstable, variability is caused by special causes (also called assignable causes), i.e., caused by events that do not usually occur during the process execution, and thus should be prevented [1][12]. When variability is due only to common causes, the process behaves between known limits and thus it becomes possible to predict it [12].

According to [1], a process performance analysis aiming at continuous process improvement can be accomplished by executing the following tasks: (i) clarify business goals; (ii) identify and prioritize process issues; (iii) select and define measures of processes or products; (iv) collect, verify and retain data on process execution; (v) analyze process stability; and (vi) analyze process capability.

The main tools used for analyzing process stability are control charts [12]. From them, it is possible to verify whether the process variability is due only to common causes or not. There are many types of control charts, and each one can be more appropriate for a specific context. For an adequate control chart, one needs to check the question to be answered, the type of data measured (variables or discrete), the size of subgroups (if applicable), the data distribution model, among others [1].

Although each type of control chart has some rules for detecting if a process is stable, they are difficult to analyze when the responsible for the analysis is not aware of both the process and the organizational context. Moreover, the wrong type of control chart can lead to mistakes in the analysis (e.g., indicating that a process is stable when it is not). In such cases, only an expert may be able to notice this issue and analyze the data with another type of control chart.

In order to support a non-expert person in the execution of performance analysis activities, a knowledge-based system (KBS) can be used. It aims at imitating human reasoning in problem-solving for assisting decision making [13], with a combination of artificial intelligence techniques and a database of specific knowledge. The concept of KBSs is widespread amongst academics and researchers [14]. Particularly, many knowledge management initiatives have been proposed in software engineering, such as [15] and [16]. These studies illustrate the synergy between these two areas.

Basically, knowledge-based systems seek to implement knowledge management activities in an organization. Usually, the main activities are knowledge identification, knowledge acquisition, knowledge development, knowledge dissemination, knowledge utilization, and knowledge maintenance [17]. However, such activities may vary depending on strategic goals and the organizational context.

Knowledge management deals with two kinds of knowledge: explicit and tacit. Explicit knowledge, also known as “codified knowledge”, can be easily described in a textual or symbolic form. Tacit knowledge, in turn, is on people’s minds and is hard to express [17]. In this work, both kinds will be captured and made available in the body of knowledge.

## III. A BODY OF KNOWLEDGE FOR SOFTWARE PROCESS PERFORMANCE ANALYSIS

Based on the execution of a systematic mapping study (briefly described in the Section VI), in addition to literature reviews, it could be noticed that there is a lack of works that support process performance analysis by knowledge management practices. Moreover, there are only a few papers in the literature that describe process performance analysis in software, as pointed out in [6]. This scenario makes the learning and adoption of process performance analysis by software organizations more difficult, even for those organizations that already have achieved intermediate maturity levels (e.g., CMMI level 3 or MR-MPS-SW level C).

The knowledge base is a major component of a KBS. Therefore, for developing SPEAKER, the knowledge to be made available was identified and organized in a body of knowledge. Such body of knowledge is presented in the following subsections, highlighting the knowledge required for executing properly the performance analysis (answering Q1 showed in Section I), as well as how this knowledge is being documented in SPEAKER (answering Q2).

### A. Knowledge required to process performance analysis

Based on the technical literature about performance analysis in software (e.g. [1][9][10]) and also in non-software area (mainly in manufacturing area) (e.g. [12]), a set of activities and tasks was identified. They served as input for the identification of the required knowledge to execute software process performance analysis. The main activities and tasks are listed in Table 1. This knowledge involves the breakdown of activities and tasks, and the description of techniques or methods that can be used for supporting the execution of performance analysis. It also involves the requirements expected by maturity models and quality standards. The description of each technique or method has the following structure: name, description, application context, data characteristics to which the method applies (variable or attribute, sample size, data distribution, etc.), and a practical example in the software area.

For composing the body of knowledge, both types of knowledge (explicit and tacit) are intended to be captured. In the context of this work, explicit knowledge is that spread in books, papers, journals and academic theses, while tacit knowledge is collected from interviews with experts.

**Table 1. Activities and Tasks for Performance Analysis**

Activities	Tasks
Preparing for Performance Analysis	Identify quantitative goals of process quality and performance
	Identify critical processes
	Assess the adequacy of the process for performance analysis
Verifying Process Stability	Group data of similar projects
	Select appropriate type of control chart
	Construct the control chart
	Apply stability and trends tests
	Identify and execute corrective actions to stabilize the process (if necessary)
	Confirm process stability (if necessary)
	Establish performance baseline
Verifying Process Capacity	Determine process capacity
	Compare capacity results with quantitative goals of process quality and performance
	Identify and execute corrective actions to make the process capable (if necessary)
Establishing Performance Model	Identify dependent variable
	Identify possible independent variables
	Select appropriate analysis method according to variable types
	Develop the regression equation, probabilistic model, or simulation
	Calibrate and test the performance model
Monitoring Process Stability	Collect new measures
	Update control chart
	Check need to recalculate performance baseline
	Apply stability and trends tests
	Confirm process stability
Monitoring Process Capacity	Monitoring Process Stability (activity)
	Verifying Process Capacity (activity)

The explicit knowledge involves technical knowledge about quantitative and statistics methods, and is present in the literature on process performance analysis in both software and not related to software areas. In the body of knowledge, when such knowledge comes from an area not related to software, an analysis is made to verify whether it has been already applied to software or not, and if some adjustment is necessary. In order to verify if the knowledge to be stored in SPEAKER is correct and adequate, peer reviews with experts in the corresponding key areas are planned. Thus, it is expected to minimize the risk that the knowledge provided by the environment does not achieve its purpose.

Tacit knowledge, in turn, is rarely described in books or papers and it is difficult to capture [17]. However, tacit knowledge can be partially converted to explicit knowledge and then be managed by a knowledge system [8]. To this end, in this work, tacit knowledge is collected and enriched by performing successive interviews with experts, who describe how one executes each performance analysis activity and in which context. After that, the collected knowledge is included in the body knowledge of the system, becoming an explicit knowledge. These experts are involved in the implementation of high maturity level in software organizations or are certified appraisers to lead appraisals for organizations aiming to attain high maturity levels. Therefore, this knowledge will allow SPEAKER works as an assistant, allowing the practitioner to think about aspects that usually would be overlooked.

## B. Knowledge documentation

Besides identifying which activities and tasks must be supported and which knowledge must be comprised in the body of knowledge, it is necessary to identify the format this knowledge must be organized and displayed to practitioners.

The way this knowledge is documented and presented influences the ease of its understanding and use. Therefore, the knowledge must be documented in a format that is suitable for practitioners. In this sense, SPEAKER must allow different ways of documenting and presenting the knowledge related to performance analysis activities, as follows:

- *Process*: The knowledge about activities and tasks related to process performance analysis should be documented in the form of a process, so that they can be described with their goals, input and output criteria, their responsible members etc. Besides, the sequence and dependencies between activities must also be evidenced.
- *Recommendations*: The suggested techniques for supporting the execution of each performance analysis activity should be presented through descriptions and examples of use. Tips and points of attention related to the execution of those activities must be presented as well.
- *Rules*: A result of a previous activity can affect or limit the options to execute the next activity. This knowledge is presented through rules.
- *Best practices*: Describe the consensus between experts in executing certain aspects of process performance analysis.
- *Lessons learned*: After executing performance analysis activities in the organizational context, it should be possible for the practitioner to describe lessons learned about this execution. This way, the initial knowledge base of the environment can be constantly fed back.

## IV. EXAMPLE OF USE

Aiming to present how the body of knowledge can help a software development organization in executing performance analysis, a fictitious scenario of use is given. It was derived from professional experience of the authors and findings from the literature. Due to space limits, only the activity “Preparing for Performance Analysis” (shown in Table 1) is described.

The Alpha organization develops corrective and evolutive improvements of its products. It has achieved CMMI level 3, and the top management decided to start the implementation of level 4. As the organization does not have a consultant for supporting the necessary improvements, it was decided that Alpha’s software engineering process group (SEPG) should study the necessary concepts for applying the performance analysis and implement the process. For supporting the SEPG, the body of knowledge (to be provided by SPEAKER) is used.

The activity “Preparing for Performance Analysis” aims to identify which critical subprocesses (and their attributes) must be subject to performance analysis. The body of knowledge must then comprise all the tasks that must be executed to this end. Such tasks are listed in Table 2. The necessary knowledge to perform each task (whose samples are also described in

Table 2) provides some definitions, a description of applicable techniques, some examples and the lessons learned related to the task at hand. Further information on a concept or a specific technique will also be provided for helping its understanding.

**Table 2. Tasks and related knowledge**

Tasks	Corresponding Knowledge
<b>Identify quantitative goals of process quality and performance</b>	
Define/clarify the organization's vision and mission	Definition and examples of vision and mission
Identify strategic goals and their indicators	Balanced Scorecard (description and example) Goal-Question-Metric (GQM) (description and example)
Analyze external environment (market, clients, competitors etc.)	SWOT analysis (description and example) Porter Forces (description and example)
Analyze internal environment (organization's strengths and weaknesses)	SWOT analysis (description and example)
Identify medium term tactical software goals	Definition and example of medium term tactical goals
Identify short term tactical software goals	Definition and example of short term tactical goals
Identify quantitative goals of process quality and performance	Definition and example of quantitative goals of quality and performance
<b>Identify critical processes</b>	
Establish selection criteria for subprocesses	Examples of selection criteria Recommendations from maturity models
Evaluate subprocesses regarding the degree of compliance with the criteria	Delphi technique (description and example) Weighted Multi-voting (description and example) Pareto Analysis (description and example)
Prioritize critical subprocesses	-
<b>Assess the adequacy of the process for performance analysis</b>	
Identify attributes related to each critical subprocess	-
Evaluate the suitability of each attribute to performance analysis	Requirements needed for performance analysis
Identify and execute corrective actions for adjusting measures to performance analysis (if applicable)	Recommendations of corrective actions Results of corrective actions previously adopted by the organization
Define list of processes for performance analysis, with their related attributes and priorities	-

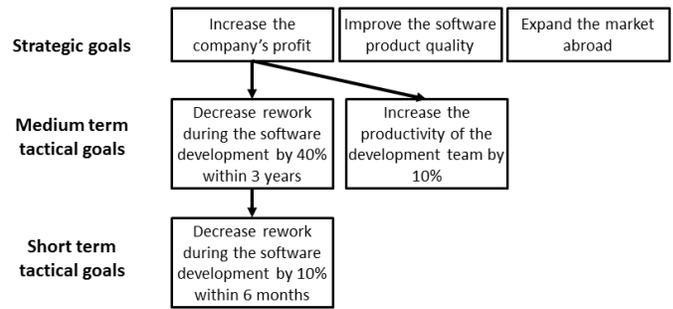
From the information provided through the body of knowledge, the SEPG establishes along with top management the following organization's vision and mission.

**Mission:** "Develop high quality software in the medical field, providing clients with more agility in their processes, and providing resources for a more effective decision making. This way, we hope that our clients can have a high productivity in their consultations and value the preservation of their patients' health with confidence".

**Vision:** "Be recognized as a company of excellence in the development of software in all sectors of the medical domain, reaching a continuous and sustainable growth".

Proceeding with the tasks set out by the body of knowledge, the SEPG identifies strategic and tactical goals (both short and long term) from successive decompositions. An example of these goals is shown in Figure 1 (for illustration purposes, only one requirement is derived at each step). For decomposing the short term software goal in quantitative goals of quality and performance, the SEPG needs to identify which subprocesses

are related to this goal, as recommended in the body of knowledge. In the example shown, the subprocesses Software Coding and Software Testing are identified.



**Figure 1. Alpha's Goals**

In order to assist the organization in properly defining quantitative goals, the information necessary for their completion are presented with an example indicating how they should be filled out. The quantitative goals of quality and performance identified by Alpha are presented in Table 3.

**Table 3. Quality and performance quantitative goals for the Alpha organization**

Subprocess	Action	Value	Measurable Element
Software Coding	decrease	10%	rework effort
Software Testing	decrease	5%	rework effort

For selecting critical subprocesses, a set of criteria identified in the literature is presented, which may be used according to the organizational context, as well as recommendations from maturity models, e.g., CMMI's suggestion stating that a minimal set of subprocesses must encompass at least: a subprocess for each stage of the life cycle (product engineering activities), a subprocess for project management and a subprocess for supporting processes.

From the information provided, the SEPG establishes a set of criteria to be adopted, and choose to use the Weighted Multi-voting technique [18]. After applying the technique, the subprocesses Software Coding, Software Testing and Requirements Specification are identified as critical. In order to assess whether the critical subprocesses and their attributes are suitable for performance analysis, a checklist is presented for evaluating their attributes. Table 4 presents the application of the checklist to the subprocess Software Coding by the SEPG.

**Table 4. Results from the evaluation of the subprocess Software Coding**

Attribute	Result	Problem
Coding effort	Suitable	N/A
Rework effort	Suitable	N/A
Schedule	Not suitable	Data are collected in a non-standard way

From the results, the SEPG cannot execute performance analysis with the attribute "schedule", but can establish corrective actions, so that such attribute can be properly collected (hence properly analyzed) in future projects. The body of knowledge comprises recommendations that can guide the SEPG in this task. Examples include (i) a proper training

focused on the responsible for the data collection and (ii) the definition of clearer collection procedures.

Thus, at the end of the activity “Preparing for Performance Analysis”, Alpha has a list with critical subprocesses identified and prioritized. From this list, the organization can begin performance analysis with the verifications of stability of the first subprocess (as shown in Table 1), with regards to an attribute considered a priority.

## V. TOWARDS A KNOWLEDGE-BASED ENVIRONMENT FOR SOFTWARE PROCESS PERFORMANCE ANALYSIS

Aiming to minimize the difficulties faced by software organizations for implementing process performance analysis (as pointed in the previous sections), we are developing SPEAKER, a knowledge-based environment for supporting the execution of software process performance analysis. By using this environment, we expect to help software organizations in analyzing their process properly, taking into account the organizational context and correctly applying the concepts and techniques of process performance analysis. Through SPEAKER, the presented body of knowledge must be accessed and maintained effectively by practitioners.

The main module of SPEAKER is the Knowledge-Based System (KBS), which aims to provide the knowledge about performance analysis and perform the knowledge management activities. Such activities involve the capturing, storage, recovery, and maintenance of the knowledge and rules described in the body of knowledge. In the context of the KBS, the main knowledge management activities will be performed as follows:

- *Knowledge identification and acquisition:* This activity involves both explicit and tacit knowledge, and also covers the knowledge identification, when the required knowledge to execute the process performance analysis is identified. For the construction of the initial knowledge base of the environment, this activity has been performed through successive technical literature reviews and interviews with experts. After the environment is set up and used by an organization, the practitioners will be responsible for this activity by registering their understanding of performance analysis execution, or as a result of the *knowledge maintenance* activity;
- *Knowledge organization and storage:* The knowledge is organized and stored so that it can be available and understandable by practitioners. To this end, different knowledge formats will be adopted depending on the performance analysis activity being executed at the time. These formats include lessons learned, best practices, case studies, examples and stories;
- *Knowledge retrieval and utilization:* This activity allows the knowledge to be used by practitioners to support them when a performance analysis activity is executed. The knowledge related to this activity is displayed to better assist its execution. Thus, only the knowledge that is relevant at the time is presented to the users, in order to avoid information overloading. Besides, one can search for a particular knowledge that is not being presented;

- *Knowledge maintenance:* While the knowledge is used to execute performance analysis, users can evaluate the usefulness and accuracy of such knowledge. Thus, knowledge with low scores can be reviewed or excluded from the knowledge base. This evaluation may also indicate the lack of a specific knowledge; in such case, a new round of the *knowledge identification and acquisition* activity can be conducted.

SPEAKER also consists of two other modules (not detailed in this paper): a process components library and a tool for process instantiation and execution. They are responsible for defining and storing the execution history of the process performance analysis and establishing the communication between the KBS and the user.

## VI. RELATED WORKS

In order to explore how knowledge management practices are used in software process performance analysis, a systematic mapping study was conducted. Such study, also known as Scoping Study, is designed to provide a wide overview of a research area [19]. The main goal of this study was to identify papers that (i) report difficulties, problems or challenges regarding knowledge issues in the process performance analysis, and (ii) suggest or adopt knowledge management methods or techniques to deal with these issues. The planning and results of this study are described in [20].

Through this systematic mapping study, only one approach that deals with some knowledge management practices to support process performance analysis was identified. This approach, called “SPC-Framework”, proposes to adapt the original statistical process control concepts for application in the software development scenario [21].

To achieve this purpose, this framework comprises [21]: (i) Test set: the selection of a test set, amongst those presented in the SPC literature, along with its rearrangement in logical classes; (ii) Test interpretation: the interpretation of every test in each class, from the software process point-of-view; (iii) Investigation process: a process to guide software process monitoring and stability investigation; and (iv) Experience Base: a framework of an experience base for collecting experience in the use of the SPC, based on Decision Tables.

Based on the few results of the study, non-structured reviews in the literature were made to better ground the proposed system. These reviews took into account areas non-related to software, and some works were identified, especially in the manufacturing area. One of them is an expert system [22] which aims to support pattern recognition in a control chart. According to the authors, this system allows plotting control charts, identifying instabilities patterns, calculating the capability index, identifying possible root causes, and suggesting improvement actions.

Another work in the manufacturing area is an advisory system aimed at capturing and disseminating knowledge to support the selection of an adequate control chart in a specific situation [23]. This system is based on pre-set rules that drive the user to make the selection.

The few identified ones have some significant drawbacks that can harm the execution of process performance analysis, namely: (i) they generally consider only one type of control chart; (ii) they do not deal with the whole performance analysis process, but only some tasks; and (iii) they do not take into account the context information during the analysis.

## VII. CONCLUSIONS

This paper presented how a body of knowledge on process performance analysis was organized. By integrating it into SPEAKER, software development organizations can be properly supported in carrying out performance analysis.

After the complete organization and structuring of the body of knowledge, the next research steps are the development of SPEAKER, based on the infrastructure provided by CORE-KM (Customizable Organizational Resources Environment with Knowledge Management) [24], a customizable environment that supports the definition, customization and execution of a knowledge management system that is specific for each organization. This environment was developed by our research group at COPPE and was already used in academic contexts and in Brazilian organizations for creating knowledge-based environments that support their business processes.

A survey was conducted with some experts in software process performance analysis, aiming to verify the activities and tasks identified as required for executing performance analysis. 4 out of 10 participants answered the survey. All of them stated that the proposed activities are indeed necessary. They also agreed with the sequence of these activities. However, according to them, the representation of iterations between activities should be emphasized, and the verification of the process capability should be better explained. One respondent indicated that statistical tools should be recommended as well. Some of these improvements are already being incorporated into the body of knowledge.

In addition to the evaluation of the knowledge identified and organized in the system, the SPEAKER environment will be evaluated as a whole. To this end, it is planned to use SPEAKER in the context of a real software development organization that is preparing to adopt performance analysis in their processes. The evaluation will include aspects such as (i) the sufficiency of the knowledge provided by SPEAKER when supporting non-experts in executing process performance analysis, and (ii) the adequacy of the knowledge representation format with respect to the stakeholders' needs.

## ACKNOWLEDGMENT

The authors would like to thank CNPq for the financial support to this research.

## REFERENCES

- [1] W. A. Florac, and A. D. Carleton, *Measuring the Software Process: Statistical Process Control for Software Process Improvement*, Addison Wesley, 1999.
- [2] M. Komuro, "Experiences of Applying SPC Techniques to Software Development". In: 28th International Conference on Software Engineering (ICSE), Shanghai, China, pp. 577-584, 2006.
- [3] D. Card, K. Domzalski, and G. Davies, "Making Statistics Part of Decision Making in an Engineering Organization", *IEEE Software*, 25(3), pp. 37-47, 2008.
- [4] C. Hale, and M. Rowe, "Do not Get Out of Control: Achieving Real-time Quality and Performance", *CrossTalk*, v. 25 (1), pp. 4-8, 2012.
- [5] CMMI Product Team. "Maturity Profile Reports – CMMI® for SCAMPI v1.2/v1.3 – Class A Appraisal Results", Software Engineering Institute, Carnegie Mellon University Pittsburgh, 2013. Available in <http://cmmiinstitute.com/resource/process-maturity-profiles/>
- [6] A.Tarhan, and O. Demirörs, "Investigating Suitability of Software Process and Metrics for Statistical Process Control", *Software Process Improvement*, LNCS, vol. 4257, pp. 88-99, 2006.
- [7] J. L. Boria, "What's Wrong With My Level 4?", *Personal Communication*, 2007.
- [8] M. S. Abdullah, C. Kimble, I. D. Benest, R. F. Paige, "Knowledge-based systems: a re-evaluation", *Journal of Knowledge Management*, vol. 10(3), pp. 127-142, 2006.
- [9] CMMI Product Team, CMMI® for Development (CMMI-DEV), V1.3, Software Engineering Institute, 2010. Available in: <http://www.sei.cmu.edu/>. Accessed in February/2014.
- [10] SOFTEX, Brazilian Software Excellence Promotion Association, MR-MPS-SW – General Guide, 2013, Available in: <http://www.softex.br/mpsbr>. Accessed in March/2014.
- [11] ISO/IEC, ISO/IEC 15504: Software Engineering – Process Assessment – Part 2: Performing an Assessment, International Organization for the Standardization and International Electrotechnical Commission, 2003.
- [12] D. J. Wheeler and D. S. Chambers, *Understanding Statistical Process Control*, 2<sup>nd</sup> ed., SPC Press, Inc, 1992.
- [13] J. A. Clark, and F. Soliman, "A Graphical Method for Assessing Knowledge-based Systems Investments". *Logistics Information Management*, 12(1/2), pp. 63–77, 1999.
- [14] H. Huang, "Designing a Knowledge-based System for Strategic Planning: A Balanced Scorecard Perspective". *Expert Systems with Applications*, v. 36, pp. 209–218, 2009.
- [15] I. Rus and M. Lindvall, "Knowledge Management in Software Engineering". *IEEE Software*, v. 19 (3), pp. 26–38, 2002.
- [16] F. O. Bjørnson and T. Dingøy, "Knowledge Management in Software Engineering: A Systematic Review of Studied Concepts, Findings and Research Methods Used". *Information and Software Technology*, v. 50 (11), pp. 1055-1068, 2008.
- [17] A. Abecker, A. Bernardi, K. Hinkelmann, et al. "Toward a Technology for Organizational Memories". *IEEE Intelligent Systems*, 13(3), pp.30-34, May, 1998.
- [18] Kilbride, J. "Making Better", Kilbride Consulting, Inc, 2003.
- [19] B. Kitchenham and S. Charters, "Guidelines for performing Systematic Literature Reviews in Software Engineering", Technical Report EBSE 2007-001, Keele University and Durham University Joint Report, 2007.
- [20] N. C. L. Schots, "A Knowledge-based Environment for Software Process Performance Analysis". Qualifying Exam, Federal University of Rio de Janeiro, 2013 (in Portuguese).
- [21] T. Baldassarre, N. Boffoli, D. Caivano and G. Visaggio, "Managing Software Process Improvement (SPI) through Statistical Process Control (SPC)", In: 5th International Conference on Product Focused Software Process Improvement (PROFES), pp. 30-46, 2004.
- [22] M. Bag, S. K. Gauri and S. Chakraborty, "An Expert System for Control Chart Pattern Recognition". *International Journal of Advanced Manufacturing Technology*, v. 62 (1-4), pp. 291-301, 2011.
- [23] S. M. Alexander, and V. Jagannathan, "Advisory System for Control Chart Selection", *Computers & Industrial Engineering*, v. 10 (3), pp. 171-177, 1986.
- [24] C. Galotta, D. Zanetti and A. R. Rocha, "Organizational Learning Based on a Customizable Environment for Knowledge Management Using Intranet". In: E-LEARN 2004 – World Conference on e-Learning in Corporate, Government, Healthcare & Higher Education, v. 2, pp. 2626-2633, Washington, EUA, 2004.

# Towards a flexible approach to manage varying and altering information representations

Tobias Haubold, Georg Beier, Wolfram Hardt

Zwickau University of Applied Sciences, Informatics, Zwickau, Germany

E-mail: tobias.haubold@fh-zwickau.de

## Abstract

*Information and content is described using content or document models. Content production always has a particular purpose, target audience and language. Often various forms of representations are created in different data formats and all overlap in content and structure. Fragments of such content have relations across different origins. Such relations refer to a finer granularity than the elementary unit of most management tools. Thus management tools are unable to provide automated support for such relations and consistency must be manually ensured during maintenance.*

*Using the example of learning and teaching content this paper outlines key challenges of the problem domain, fundamental characteristics of content production and management and necessary concepts of content and document models to address these problems. Related work on content and document models as well as management approaches is discussed and an approach for a repository architecture is presented to target these problems on management level.*

## 1. Motivation

Digital learning and teaching materials are usually the result of target oriented content production processes. They are created using appropriate editing tools (Office-Tools like Word, Powerpoint, Apache OpenOffice, LibreOffice, image processing applications, special tools to produce SCORM-based e-learning content, and more) and managed using the specific data format of the tools.

Every kind of knowledge transfer pursues an intention and follows a didactic approach. Thus teaching in presence (presentation slides), self-study (lecture notes) and interactive online learning (web content) demand all different kinds of content representations. The purpose of materials is significantly characterized by the learner as target audi-

ence. Thus materials differ by means of the didactic design, prior knowledge, expert and non-skilled target audience or fundamental or advanced knowledge.

To create such forms of content within a reasonable time copy and paste is usually used to reuse content fragments by duplication. In addition the use of materials in international cooperation demand translation and localization.

On update or revision of content the consistency of partially redundant content fragments spread across different document files need to be ensured manually. The process of locating related content, detecting differences and collecting content for refreshing translation and localization must be done by hand and leads inevitably to time consuming maintenance of content.

This paper focuses on requirements of content and document models as well as content management approaches.

## 2. Key Challenges in Problem Domain

The process of producing learning and training content starts by defining a main learning path through the knowledge domain. According to this path the material is structured and created using an didactic approach (see [1]). Thus different documents are produced (e.g. presentation slides and scripts) sharing the same structure but contain different representations of the content. During content maintenance the common structure must be ensured manually to keep the materials consistent.

During the production of materials with a similar target audience copy and paste is usually used to reuse content by duplication. Thus content fragments are stored redundantly in different document files. During update or revision all duplicated content fragments must be maintained manually to ensure consistency. The same applies to derived content. Content fragments are duplicated and subsequently changed to better fit other purposes or author styles. During change of the comprised knowledge all derived content must be considered for manual maintenance.

Multilingual materials with a high rate of change need frequent refreshing translations and localizations. When the

---

The work was funded by the European Union and the Free State of Saxony.

latest updates are not yet available, one must deal with incompletely translated and localized materials.

Approaches to address these challenges involve document and content models as well as content management approaches.

### 3. Fundamental Characteristics of Content Production and Management

#### 3.1. Reuse of Content

By using copy and paste content can be reused by duplication. This results in redundant storage and thus content updates have to be done manually on each copy to ensure consistency.

Content references allow to reuse content without duplication and redundant storage. Instead of the content itself just a reference to it is stored<sup>1</sup>. Whereas copy and paste is a feature of the editing tool, content references are a feature of the document model and must be supported by it.

#### 3.2. Granularity in Content Production

In this context granularity is understood as the extend of content. Document models define concepts to structure content (e.g. headings for hierarchical structures in text documents or slides in presentation software) and capture content (e.g. text in text boxes or paragraphs). Each concept represents a level of granularity. Thus there are levels with finer and coarser granularity.

Especially in document oriented content production processes resulting document files are tailored to particular purposes and have a coarse granularity. Within learning object based content production processes Hamel et al. states that the granularity should be relatively small [2]. In addition Duval et al. states that learning objects with finer granularity are more easily reusable [3].

#### 3.3. Granularity in Content Management

All management tasks including versioning, categorization, meta data management as well as translation are based on an elementary content management unit. Document Management Systems (DMS) and versioning tools use simple files as management unit. Due to the coarse granularity of document files such tools are not suited to address the key challenges of section 2. Content Management Systems (CMS) use the concepts of their content model as management unit. Thus they provide finer levels of granularity but prescribe particular content models.

<sup>1</sup>links in contrast serve as navigation

## 4. Solution Approaches in Content Production

To address the key challenges content or document models need the following mechanisms.

### 4.1. Content References

The document model must provide support for content references. To use them they must be allowed on the desired position and the type of content must be supported.

There are two kinds of content references: references to files and references to file fragments. In the latter case just the referenced fragment is included instead of the whole file. This requires mechanisms to address content fragments defined by the referenced document model.

### 4.2. Content Aggregations

Mechanisms to aggregate content are based on content references and allow to compose documents out of other documents and document fragments. In addition to content inclusion the content is adapted to the current context if required, i.e. it is included on the desired position in the hierarchical structure of the document and the document wide layout and formatting settings are applied. Layout and formatting settings particularly defined for the included content are not adjusted.

### 4.3. Content Filtering

Mechanisms for content filtering allow for conditional content inclusion. Some kind of configuration settings specify whether or not particular content fragments are included or not. Usually such mechanisms evaluate meta data attached to content fragments.

### 4.4. Separation of Content and Presentation

There are two common approaches to separate content and presentation. First, the content contains meta data referencing presentation details. A presentation engines resolves these references and renders the content appropriately. Examples are the HyperText Markup Language (HTML) which references Cascading Style Sheets (CSS) files or Open Document (ODF) which defines format templates referenced by content.

Second, there are transformation based approaches. The content is either described by an XML based document model or an implemented content model. Transformations are described using the Extensible Stylesheet Language Transformations (XSLT) or an appropriate template language. An XSLT or template processor takes the content and transformations as input and produces output documents. Examples are DocBook, DITA or CMSs.

## 5. Proposed Content Management Approach

To provide automated tooling support for the key challenges mention in section 2, content management must detect and manage content references and content aggregations and must support meta data for content filtering on a fine level of granularity. The proposed approach focuses on a repository architecture to manage fine granular content fragments and to capture content references and aggregations in a document and content model independent manner.

### 5.1. Content Structure Layer

The structural aspects of content are captured by this layer, illustrated in figure 1. Opaque content represents any

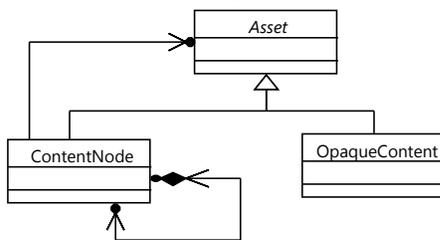


Figure 1: Content Structure Layer UML Sketch

unstructured content and can be compared to the management unit of other management approaches. Content nodes are used to represent structured content in form of a tree by having any number of children nodes. Content can be attached directly to content nodes or by referencing other assets. All assets are identifiable by an URI (Uniform Resource Identifier), may have any number of arbitrarily meta data attached and store content binary.

### 5.2. Semantic Layer

All assets are resources in the semantic sense. This layer is used to classify content and to capture all kinds of content relationships. It resembles a large graph with nodes being content assets or concepts defined by semantic technologies. The edges constitute relationships between content assets, relationships to semantic meta data as well as relationships between semantic concepts.

This layer enables the use of existing meta data standards like Dublin Core [4] or LOM (Learning Object Metadata)[5] as well as their extension or supplement with custom concepts, categories, terminologies or taxonomies defined using SKOS (Simple Knowledge Organization System)[6], RDFS (Resource Description Framework Schema)[7] or OWL (Web Ontology Language)[8]. Having all content assets and semantic meta data accessible in one huge graph allows for simple querying using SPARQL[9] or gremlin[10] as well as easy exploration for similar content.

### 5.3. Data Storage Layer

The main purpose of this layer is to persist data. All data is stored as binary stream. A separate data storage layer is suited to support deduplication and distribution of data to enable the use as distributed repository.

### 5.4. Repository Architecture

An adapter mechanism is used to map content into the proposed management model. File based adapters build appropriate content structures based on content or document model concepts as well as content meta data. A plug-in mechanism supports extensible file based mappings. Adapters may be integrated into content production environments like Office-Suites as well.

The adapter concept allows for type as well as instance based mappings. Thus individual content fragments of the same content type may be mapped differently. Furthermore adapters may overcome the limitations of document models by providing extensions to support content references and aggregation mechanisms.

## 6. Related Work

L<sup>A</sup>T<sub>E</sub>X supports content references to files and separates content and presentation. It does not support content aggregations (because explicit outline level for headlines), content filtering and references to file fragments.

DocBook 5[11] supports content references to files, content filtering and separates content and presentation. Together with XInclude[12] and XPath[13] content references to document fragments are supported as well as limited content aggregations<sup>2</sup>. In future DocBook 5.1[14] will support content aggregations.

DITA[15] supports content references to files and document fragments, content filtering, separation of content and presentation as well as content aggregations directly.

Document models of office suites support content references basically to images, multimedia and OLE objects (Object Linking and Embedding). Content aggregations are not supported. OLE is used to include document fragments of other OLE supported applications in a display oriented manner. If no appropriate application is available, included document fragments are usually replaced by images. Open Document[16] text documents provide the concept of sections for modular editing. It allows to include fragments of other documents by reference and replaces the document wide style settings. Because outline levels are explicitly stated it cannot be considered as content aggregation mechanism.

<sup>2</sup>the element `section` can be nested and specifies the outline level relatively

Only DITA provides mechanisms for all solution approaches mentioned in section 4. All other document models are not suited by default to target all key challenges of section 2. In addition content management approaches lacking support too.

According [17] there are different kinds of CMSs but there is no clear distinction between them. In accordance with [18] related kinds of CMS are Web CMS, Learning CMS, and Document Management Systems (DMS). Web CMS as well as Learning CMS are limited to web technology based publishing[18]. They might be suited to manage content for interactive online learning but not for materials used for teaching in presence like presentation slides or lecture notes. The proposed approach is suited for any kind of content model and don't have such limitations.

DMSs are designed to manage document files. They do not consider their internal structure. Thus they are not capable of detecting and managing content references and content aggregations. Alfesco[19] as example provides support to manage translations on document level. Thus it supports the third key challenge with file-based granularity. The proposed approach is able to manage content with relations on any level of granularity. The management software edu-sharing[20] is a distributed repository for teaching content. It is based on Alfresco and therefore bound to the same content management approach [21].

## 7. Summary

Document oriented production of learning and teaching materials is still prevalent in institutions of higher education like universities. With "documents must die" criticized [22] such approaches almost 10 years ago and states that efforts for share and reuse based on simple file oriented document models are bound to fail. He argued that we need more sophisticated content and document models. Though many content and document models evolve to open standards, they still have limitations and need to be supported by a flexible yet scalable content management approach.

Further research includes the mapping of content and document models to the management model including the analysis of consequences, its eligibility and effort as well as feasibility to extend document models with aggregation mechanisms.

## References

- [1] Michael Kerres. *Mediendidaktik: Konzeption und Entwicklung mediengestützter Lernangebote*. Oldenbourg Wissenschaftsverlag, 2012.
- [2] Cheryl J. Hamel and David Ryan-Jones. Designing instruction with learning objects. *International Journal of Educational Technology (IJET)*, 3(1), November 2002.
- [3] Erik Duval and Wayne Hodgins. A LOM research agenda. In *Proceedings of the twelfth international conference on World Wide Web*, pages 1–9. ACM Press, 2003.
- [4] DCMI Metadata Terms. DCMI Recommendation, Dublin Core Metadata Initiative (DCMI), June 2012.
- [5] IEEE Learning Technology Standards Committee. IEEE Standard for Learning Object Metadata. Approved Publication of IEEE, Institute of Electrical and Electronics Engineers (IEEE), 2002.
- [6] Sean Bechhofer and Alistair Miles. SKOS Simple Knowledge Organization System Reference. W3C Recommendation, World Wide Web Consortium (W3C), August 2009. <http://www.w3.org/TR/2009/REC-skos-reference-20090818/>.
- [7] Ramanathan Guha and Dan Brickley. RDF Schema 1.1. W3C Recommendation, World Wide Web Consortium (W3C), January 2014. <http://www.w3.org/TR/2014/PER-rdf-schema-20140109/>.
- [8] W3C OWL Working Group. OWL 2 Web Ontology Language Document Overview (Second Edition). W3C Recommendation, World Wide Web Consortium (W3C), 2012. <http://www.w3.org/TR/2012/REC-owl2-overview-20121211/>.
- [9] Andy Seaborne and Steven Harris. SPARQL 1.1 Query Language. W3C Recommendation, World Wide Web Consortium (W3C), March 2013. <http://www.w3.org/TR/2013/REC-sparql11-query-20130321/>.
- [10] Gremlin Graph Traversal Language. <https://github.com/tinkerpop/gremlin/wiki>.
- [11] The DocBook Schema Version 5.0. OASIS Standard, Organization for the Advancement of Structured Information Standards (OASIS), November 2009. <http://docbook.org/specs/docbook-5.0-spec-os.html>.
- [12] Jonathan Marsh, David Orchard, and Daniel Veillard. XML Inclusions (XInclude) Version 1.0 (Second Edition). W3C Recommendation, World Wide Web Consortium (W3C), November 2006. <http://www.w3.org/TR/2006/REC-xinclude-20061115/>.
- [13] James Clark and Steven DeRose. XML Path Language (XPath) Version 1.0. W3C Recommendation, World Wide Web Consortium (W3C), November 1999. <http://www.w3.org/TR/1999/REC-xpath-19991116>.
- [14] DocBook 5.1 Candidate Release 2, December 2013. <http://docbook.org/xml/5.1CR2/>.
- [15] Darwin Information Typing Architecture (DITA) Version 1.2. OASIS Standard, Organization for the Advancement of Structured Information Standards (OASIS), December 2010. <http://docs.oasis-open.org/dita/v1.2/spec/DITA1.2-spec.html>.
- [16] Open Document Format for Office Applications (OpenDocument) Version 1.2. OASIS Standard, Organization for the Advancement of Structured Information Standards (OASIS), September 2011. <http://docs.oasis-open.org/office/v1.2/OpenDocument-v1.2.html>.
- [17] Wolfgang Maass. *Semantic Technologies in Content Management Systems: Trends, Applications and Evaluations*, chapter 1, page 4. Springer, 2012.
- [18] Ann Rockley and Charles Cooper. *Managing Enterprise Content: A Unified Content Strategy*. Voices That Matter. Pearson Education, 2 edition, 2012.
- [19] Alfresco homepage, 2014. <http://www.alfresco.com/>.
- [20] edu-sharing homepage. <http://edu-sharing.net/>, 2014.
- [21] Michael Klebl, J. Bernd Krämer, Annett Zobel, Matthias Hupfer, and Christian Lukaschik. Distributed Repositories for Educational Content. *elead*, 7(1), 2010.
- [22] Erik Duval. We're on the road to ... In Lorenzo Cantoni and Catherine McLoughlin, editors, *Proceedings of the ED-MEDIA 2004 World Conference on Educational Multimedia, Hypermedia and Telecommunications*, pages 3–5, Lugano, Switzerland, 2004. AACE.

# Flood Citizen Observatory: a crowdsourcing-based approach for flood risk management in Brazil

Lívia Castro Degrossi, João Porto de Albuquerque  
Dept. of Computer Systems, ICMC  
University of São Paulo  
São Carlos, Brazil  
{degrossi, jporto}@icmc.usp.br

Maria Clara Fava, Eduardo Mario Mendiondo  
São Carlos School of Engineering  
University of São Paulo  
São Carlos, Brazil  
mclarafava@usp.br, emm@sc.usp.br

**Abstract**—The number and intensity of floods have increased worldwide due to climate change, causing more damage, deaths and economic impacts than any other natural disaster. Coping with this type of disaster requires up-to-date, complete and accurate information about the current state of environmental variables. To make a contribution in this context, this paper proposes a crowdsourcing-based approach for obtaining useful volunteer information for the context of flood risk management. Furthermore, an experimental evaluation was performed in order to verify the effectiveness of this approach.

**Keywords**- *Flood Risk Management; Volunteered Geographic Information; VGI; Crowdsourcing; Citizen Observatory*

## I. INTRODUCTION

Climate change is increasing the number and intensity of natural disasters around the world [1]. Between 2000 and 2012, natural disasters caused approximately US\$ 1.7 trillion in economic losses and affect 2.9 billion people and cause 1.2 million deaths [2]. The occurrence of natural disasters is linked not only to environmental characteristics, but also to the vulnerability of the social system [3]. In this perspective, disaster management (DM) aims to reduce or avoid potential losses and provide assistance to the victims. In particular, spatial information consists of an essential resource in the whole process of DM [4], e.g. to reduce the impacts caused by the disaster or identify its imminence. To accomplish this, information must be reliable, accurate and updated [4], [5].

In particular, floods represent 30% of the natural disasters that occur worldwide, causing more damage, deaths and economic impacts than any other event [6]. In Brazil, floods are intensified during the rainy season, between the months of December and March. Flood Risk Management (FRM) is the process of managing a situation of existing flood risk by controlling the impacts caused by it and being prepared for it [7]. Thus, information about the current state of the environmental variables is needed to allow the simulation of the effects and severity of a disaster [6], [7]. In addition, continuous monitoring of flood risk requires specific local information, such as precipitation and water height, to assess the potential flood risk. However, such information is difficult to obtain in developing countries such as Brazil.

Spatial information obtained from volunteers, so-called Volunteered Geographic Information (VGI) [8], is being used

as a data source for disaster management. This is due to the fact that VGI offers a potentially large number of volunteers who act as “sensors”, recording important local parameters for DM [9]. In particular, crowdsourcing platforms are used as a tool to assist organizations involved in DM to assist victims of natural disasters, collecting information from volunteers. In the past few years, different crowdsourcing platforms were deployed to map earthquakes (e.g. Haiti 2010), floods (e.g. Pakistan 2010/2011), among other hazards. However, to the best of our knowledge, there is not a crowdsourcing-based approach adapted and empirically evaluated for the reality of the informational needs of floods in Brazil. Thus, this paper proposes and evaluates a crowdsourcing-based approach to obtain useful volunteer information for context of flood risk management in Brazil.

The remainder of this paper is structured as follows: Section II presents the theoretical background for this work. In section III, the crowdsourcing-based approach proposed herein is explained. Section IV presents the empirical evaluation in detail. Finally, Section V concludes the paper and suggests directions for future work.

## II. BACKGROUND

In this section, we present the concepts and underlying principles related to this work.

### A. Flood Risk Management

Flood Risk Management is the process of managing a situation of an existing flood risk. The main goal is to control a flood, being prepared for it and minimizing its impacts [7]. The FRM comprises actions before, during and after a flood occurs. These actions involve early warning and forecasting scenario, contingency plans and restoration [10].

In the FRM, the preparation phase has as main objective reduce the residual risk through early warning systems and measures that can be taken to minimize the flood impacts. For this, the constant monitoring of the risks and the assessment of the danger, arising from recent information, is required. Every dollar invested in flood prevention reduces in US\$ 25 dollars the damage incurred in a natural disaster [10].

Currently, geographic information and related technologies play a fundamental role in all phases of FRM. Natural disasters are typically monitored using different devices such as sensors,

satellites, seismometers, among others. In particular, different information is used in the development of model-based prediction for flood warning systems. In this context, local and up-to-date data is essential for supporting decision-making. A crucial challenge is the availability of data at all points considered strategic. However, volunteer information, as well as sensor data, can be entered in the model for forecasting, since volunteers can act as human sensors [11]. Thus, it is possible to statistically calculate the confidence intervals between the actual and estimated value. This provides probabilistic information for the following iterations of the prediction for the filling of the points where there is no real-time measurement. Still, it is possible to use these data with monitoring data to develop predictive methods correlating the information available. Furthermore, volunteers may provide information about important parameters of local conditions on the verge of occurring a natural disaster. This information can help to provide a more effective and immediate response.

### B. Volunteered Geographic Information

With the advancement of Internet and mobile devices, users not only start to use the geographic information available online, but also provide it. Formerly, geographic information was created only by official agencies. However, with the increase of interactions made possible by the Web 2.0, the widespread use of devices equipped with GPS (Global Positioning System) and the availability of broadband access to the Internet, geographic information is being produced by people who have little formal qualification. This type of information is called Volunteered Geographic Information (VGI) [8]. Among the advantages associated with VGI, researchers emphasize its use to enhance, update or complement existing geospatial datasets [8].

Recent natural disasters have shown that volunteered information, provided through the Internet, can improve situational awareness by providing an overview of the present situation [9]. This fact occurs because VGI offers a great opportunity to raise awareness due to the potentially large number of volunteers, more than six billion people, that can potentially act as “sensors”, recording important parameters for disaster management in a local environment [8], [9].

VGI has been widely used in the context of disaster management. Some examples are the floods in Pakistan in 2010, Queensland in 2010/2011, Thailand in 2011, the Haiti earthquake in 2010, forest fire in France in 2009, among others [13]–[17]. [9] used information provided by the affected citizens by the flood of 2002 in the city of Eilenburg, Germany, to estimate impacts. Furthermore, the use of volunteered information proved essential after the earthquake in Haiti in 2010, because the existing maps of the city of Port au Prince (worst affected area) were outdated and did not contain sufficient information, making it difficult to coordinate the rescuers’ activities [18].

Despite the potential of VGI, data quality is a major concern. Information from many individuals can lead to doubts about their credibility [19]. According to [20], the credibility of VGI can be understood as a subjective concept that describes whether a piece of information can be trusted, considering any possible intentional or unintentional omission or exaggeration

error. Moreover, it is not known beforehand how and wherefrom the information will be provided. Another challenge faced refers to the location, because volunteers are in constant movement. Furthermore, VGI is often regarded as poorly structured, documented and validated [20].

In this scenario, different software platforms have been employed in order to collect volunteered information, allowing its visualization and analysis. In particular, these are used as tools to help the victims of natural disasters.

### C. Citizen Observatory

Currently, several crowdsourcing platforms support disaster management, enabling the gathering of information from citizens about the affected areas, as well as their analysis and visualization. The term crowdsourcing refers to a way of organizing the work, which involves an information system to coordinate and monitor tasks performed by people [21]. Moreover, this term can be understood as a production model where the intelligence and knowledge of volunteers are used to solve problems, create content and develop new technologies. Using volunteers to perform a specific task, such as environmental monitoring, collectively make a Citizen Observatory (CO), where data can be collected, collated and published [21]. Thus, the term Citizen Observatory can be understood as a software platform in order to obtain volunteered information about a specific topic through different devices (e.g. web browser, mobile application and SMS), and allow their visualization. In this manner, a CO can be used to share information about flood risks, such as water height in the riverbeds or flooded areas.

Different COs were used in the context of flood risk management. In 2011, during the floods in Queensland, the Australian Broadcasting Corporation launched a CO to map the flooded areas [14]. This allowed individuals to send information about the flood through email, text messages, Twitter, or the platform itself, being this information available to anyone with Internet access. Still, other COs were employed during the floods in Pakistan<sup>1</sup> in 2010 and Boulder<sup>2</sup> (USA) in 2013. While these observatories were used to provide information about the impacts caused by floods and their victims, they do not provide information to prevent or minimize the impacts of flood events. In addition, these observatories do not have any kind of integration with Spatial Data Infrastructures (SDI) as they are employed only to obtain volunteered information.

## III. FLOOD CITIZEN OBSERVATORY

In order to fill the gaps identified in the previous section, this paper presents a crowdsourcing-based approach to obtain useful volunteered information for the context of flood risk management in Brazil, which includes the definition of mechanisms to help volunteers to better interpret the environmental variables (i.e. the water level). In this section,

---

<sup>1</sup> <http://pakreport.org/ushahidi/>

<sup>2</sup> <https://boulderflood2013b.crowdmap/>

the interpretation mechanisms developed are presented along with the proposed crowdsourcing platform.

### A. Interpretation Mechanisms

Due to the vast extent of Brazilian rivers and inability to provide resources to make measurements of the environmental variables at all required strategic points, four mechanisms were proposed which aim at supporting volunteers to better interpret those variables. These mechanisms were jointly designed by an interdisciplinary team composed of computer scientists and hydrologists. Three different scenarios were considered so the volunteer can interpret environmental variables in various situations, which are described as follows.

The first scenario corresponds to a controlled point, i.e. a point where there is an interpretation resource that enables the volunteer to perform the measurement more accurately. In this scenario, there is a water level ruler (Fig. 1, item 1) laid down in the riverbed which supports the measurement of the water level at that point. Rulers were installed in fourteen points located in the five most relevant water streams of the urban area of São Carlos/SP. Each of them has at least two points, one in the upstream and one in the downstream, before the confluence with other streams.

The second scenario corresponds to a semi-controlled point, i.e. a point where there is an interpretation resource for determining the value of the water level in the riverbed in a less precise way. In this scenario there are two types of interpretation resources, a puppet similar to the human figure and multi-color bands (green, yellow, orange and red). The color bands (Fig. 1, item 3) correspond to the hazard index (HI), i.e., the danger which the population is exposed, the risk of human instability. This risk represents the forces exercised on an individual in water currents, i.e., the individual's vulnerability level exposed to floods [22].

In addition, a puppet was proposed to assist the volunteers to determine the water level more easily (Fig. 1, item 2). Thus, the volunteer can use it as a visual resource to determine the water level, because this is painted on the riverbed in some points of the streams of São Carlos/SP city or as imaginary resource, helping to determine the water level at points where there is no available resources. In the puppet, water level measurement is carried out according to pre-defined tags, i.e., ankle, knee, waist, neck and above the head (the whole body). The three proposed mechanisms for the scenarios described above are shown in Fig. 1, item 2.

Finally, the third scenario corresponds to an uncontrolled point, i.e., there is no recourse in the riverbed which could help the volunteers to interpret the water level. For this mechanism, tags (low, normal, high and overflowing) were adopted that are of simple determination and approach to popular knowledge. These tags are commonly cited in Brazilian media to report the water level of a river. Thus, it is expected that the volunteers can interpret this environmental variable more easily.

### B. Flood Citizen Observatory

The Flood Citizen Observatory (FCO) (Fig. 2) is a crowdsourcing-based approach that enables the collection of

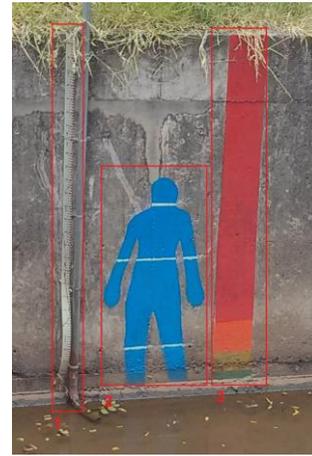


Figure 1. Resources for determining the water height in the river bed

volunteer information. This approach is inserted in a research project about flood risk management called AGORA<sup>3</sup>. Its main objective is to obtain useful volunteer information related to flood risk management, more specifically about flooded areas and water level in the riverbed, in order to provide those information for decision-making. FCO is based on the Ushahidi crowdsourcing platform, which is used worldwide by activists, emergency agencies and citizens to map extreme events [23]. In addition, in this platform, volunteers are considered “human sensors” since these can observe important parameters of flood risk management in a local environment.

In order to facilitate the information provision about flood risk in FCO, the interpretation mechanisms are represented by different categories, whereas the tags for each mechanism are represented by subcategories. Thus, the volunteer can identify, more easily, the category that best represents the observed scenario.

To send a report, volunteers can use both a mobile application and a Web site. Sending a report requires that volunteers provide the following mandatory information:

- **Title:** represents the subject addressed in the report;
- **Description:** represents the observation performed by the volunteer, for example, the water level or flooded area;
- **Category:** represents the mechanism used to interpret the environmental variable, which provides information about the water level in the riverbed;
- **Place's name:** represents the place where from the volunteer is sending the report.

Due to the uncertainty about the credibility of this information, reports are checked before they are made available online. The purpose of this verification is to reduce the number of false or inaccurate information disseminated to the public and emergency agencies.

<sup>3</sup> <http://www.agora.icmc.usp.br>

Figure 2. Crowdsourcing platform Flood Citizen Observatory

Thus, before the reports are made available online, the authors of this work approve the reports inserted into the platform. As such, the authors of this study initially play the platform administrator role, checking and approving reports inserted on it. However, it is expected that the emergency agencies involved in flood risk management will play this role.

#### IV. EXPERIMENTAL EVALUATION

The approach proposed in this work is aimed at obtaining useful volunteered information for the context of flood risk management through a crowdsourcing platform. An experiment was conducted with the objective of evaluating our approach. In the pursuit of this goal, two questions were formulated by the authors of this work, in order to direct the execution of the experiment. The first question refers to the effectiveness of the platform in obtaining volunteer information for the context of flood risk management. Thus, this question was defined as:

**Q1)** Is the crowdsourcing-based approach, Flood Citizen Observatory, effective in obtaining useful volunteer information for the context of flood risk management?

The second question was formulated in order to check if the difference between the average of the information provided by volunteers and the average of data acquired from sensors is significant. This is important for if there is a significant difference, volunteered information may not reflect the real state of the environmental variable observed, resulting in erroneous predictions about the flood risk. Thus, the following question was elaborated:

**Q2)** Is the difference between volunteered information and sensor data significant?

Once they were formulated, for each question was defined a null hypothesis, which the authors wanted to refute with the most possible significance, and an alternative hypothesis,

which the authors intended to prove. Regarding the effectiveness of the crowdsourcing platform (Q1), we intended to refute the statement that the percentage of useful volunteered information obtained by platform is less than or equal to 50% (Null Hypothesis). Furthermore, we attempted to demonstrate that the percentage of useful volunteer information obtained by the platform is more than 50% (Alternative Hypothesis). Volunteered information is considered useful if it can be used for hydrological models or for decision making by emergency agencies. As regards to the difference between the average of the volunteered information and the average of sensor data (Q2), we attempted to refute the statement that the average of the volunteered information is different from the average of sensor data (Null Hypothesis). Furthermore, we tried to prove that the average of the volunteered information is equal to the average of sensor data (Alternative Hypothesis).

After defining the experiment variables, metrics were developed to measure those variables. A criterion was defined in order to determine the usefulness of the obtained information to the context of flood risk management. For this, three different values were assigned to the following grades: 0 (the category does not represent the scenario observed, i.e. the selected category does not match to the report description), 5 (the category partially represents the scenario observed, i.e. the volunteer selected more than one category), 10 (the category completely represents the scenario observed, i.e., the volunteer selected only one category that represents exactly the scenario observed). Thus, the effectiveness of the platform is calculated by the percentage of volunteered information that has grades greater than or equal to five in this criterion. Moreover, the average values of the volunteered information is calculated by summing the values contained in the report description, referring to the water level on the water level ruler, and divided by the number of reports with this information. On the other hand, the average of sensor data is calculated by the sum of the measurements carried out by it during the period of time that the volunteers performed the observations, and divided by the number of measurements taken during this period.

The participants were strategically selected, in order to include participants with different levels of experience and expertise in the area of flood risk management. Before the execution of the experiment, participants attended a training about the crowdsourcing platform and the mechanisms used for the interpretation of the water level in the riverbed. Furthermore, participants were also shown how to insert a new report in the crowdsourcing platform. After the training, all participants were asked to insert reports about the water level observed in the water level ruler of one of the controlled points, and a second report about the water level in the same point as observed with the assistance of one of the interpretation mechanisms to be chosen by the volunteer.

For the analysis and interpretation of the results, three statistical tests were used. First, the Shapiro-Wilk [24] was used to verify the normality of the samples. Then the Levene Test [25] was applied to verify homoscedasticity of the sample, i.e., if different samples have equal variance. Finally, the T Test [26] was applied in order to statistically reject or accept the null hypothesis. According to [27], it is possible to reject the null hypothesis based on statistical test results.

A. Results

The experiment was conducted with ten volunteers (participants), with different levels of experience and knowledge in the flood risk management area (TABLE I), in a point of the watershed of São Carlos/SP city. During the execution of the experiment, the volunteers inserted fifteen reports on the platform.

First, the reports were analyzed regarding to the category selected by the volunteer in order to verify the effectiveness of the approach. For each report, a grade was assigned according to the number of selected categories and if those represented completely, partially or even not represented the scenario observed (see previous section). Among the reports inserted, nine had only one category selected, which completely represent the observed scenario. For these, it was assigned the value 10. Two reports had more than one category selected, which partially represent the observed scenario. For these reports, it was assigned the value 5. Finally, four reports had one or more categories selected, however these did not represent the observed scenario, being assigned to them the value 0. Thus, eleven reports obtained note greater than or equal to five, i.e. approximately 73% of the inserted reports can be used in the context of flood risk management. Thus, the null hypothesis was rejected, i.e., the platform is effective in obtaining useful volunteer information for the context of flood risk management. Since the volunteers attended to a training before the execution, we can assert that the training was sufficient to enable volunteers to produce useful observations for the context of flood risk management in our approach.

The reports related to the water level observed in the ruler were analyzed to verify if the difference between the average of the volunteered information and the average of sensor data was significant. For this, only ten reports were considered, since only those have the value of the water level in the ruler. First, a correspondence between the values observed by the volunteers and the values measured by the sensor was performed. For this, the authors of this work made two observations on different days regarding to the water level in water level ruler at the same point where the experiment was conducted. Each observation was compared to the value measured by the sensor to determine the difference between those values (TABLE II). Thus, it was established that there is a difference of 7 cm between the value observed by volunteers and the value measured by the sensor. This difference was added to each of the observations made by the volunteers before performing the statistical tests.

The Shapiro-Wilk Test was performed to determine whether the samples had normal distribution. The results (TABLE III) showed that the volunteer information does not have a normal distribution, since the value of Sig. (p-value) is less than the significance level<sup>4</sup> adopted (5%). In contrast, sensor data has normal distribution, because the value of Sig. is greater than the significance level.

TABLE I. PARTICIPANTS' EXPERIENCE IN THE CONTEXT OF FLOOD RISK MANAGEMENT

Experience	Number of Participants
0 month	5
1 month	1
7 months	1
12 months	2
240 months	1

For samples with normal distribution, the most appropriate hypothesis test is the T Test, in its parametric unpaired version. According [26], this test can also be applied to samples that did not have a normal distribution as long as these do not contain to many elements. Thus, this test was selected to determine whether the difference between the average of the volunteer information and the average of sensor data was significant.

To perform the t test, first it was performed the Levene test to analyze the homoscedasticity of the samples, i.e., to analyze if the samples have equal variance. The value of Sig. (p-value) (TABLE IV) shows that the samples do not have equal variance, since this value is less than the significance level. Then, the t test was performed to check whether the difference between averages was significant. The result obtained (TABLE IV) shows that the difference between the averages is not statistically significant because the value of Sig (2 - tailed) is greater than the significance level. Thus, it was possible to reject the null hypothesis. However, it is noteworthy noticing that the water level ruler was not firmly attached to the river bed (Fig. 1), so that the threat cannot be ruled out that this fact may have introduced a bias in the observations of volunteers.

Thus, we conclude that the Flood Citizen Observatory facilitates the provision of useful and accurate information for flood risk management.

V. CONCLUSION

In this work, we propose a crowdsourcing-based approach for obtaining useful volunteer information for the context of flood risk management. In this approach, volunteers are

TABLE II. DIFFERENCE BETWEEN THE VALUE OBSERVED BY THE VOLUNTEER AND THE VALUE MEASURED BY TH SENSOR

Date	Value observed by the volunteer	Value measured by the sensor	Difference
10/02/2014	7 cm	14,12 cm	7,12 cm
12/02/2014	6 cm	13,82 cm	7,82 cm

TABLE III. SHAPIRO-WILK TEST'S RESULTS

Variable Tested	Instrument	Shapiro-Wilk Test	
		Number of elements	Sig.
Water Height	Volunteer	10	0,004
	Sensor	5	0,494

<sup>4</sup> The significance level corresponds to the probability of rejecting the null hypothesis, being this true.

TABLE IV. RESULTS OF THE LEVENE TEST AND T TEST

Variable Tested	Levene Test	T Test	
	Sig.	Sig. (2-tailed)	Mean Difference
Water Height	0,002	0,093	-2,799

considered human sensors, providing information about the environment, such as water level and flooded areas.

Based on the experimental validation, the authors found that this platform is effective in obtaining useful and accurate volunteer information, since volunteers can easily provide information about the water level in the riverbed through the platform categories. This is an important step since in certain regions of Brazil there do not exist water gauges to perform such measurement in real time.

The present work thus fills the gap of empirically evaluated crowdsourcing approaches in the context of flood risk management in Brazil. However, it is necessary to move forward not only in obtaining volunteered information about flood risk, but also in the automated processing of this information to estimate the likelihood of a flood. Additionally, the development of a risk map for the affected population based on the acquired information is an essential goal for future work.

#### ACKNOWLEDGMENT

We would like to express our thanks for the financial support provided by the FAPESP-IVA project "Assessment of Impacts and Vulnerability to Climate Change in Brazil and Strategies for Adaptation Options, FAPESP 2008/58161-1". J. P. Albuquerque is granted by CAPES (12065-13-7) and E. M. Mendiondo is granted by CNPq PQ 307637/2012-3.

#### REFERENCES

- [1] U. N.- UN, "Humanitarian and Disaster Relief Assistance," 2013.
- [2] U. N. O. F. D. R. R.- UNISDR, "Economic losses from disasters set new records in 2012," 2013.
- [3] I. Alcántara-Ayala, "Geomorphology, natural hazards, vulnerability and prevention of natural disasters in developing countries," *Geomorphology*, vol. 47, pp. 107–124, 2002.
- [4] Y. Tu, Q. Li, and R. Liu, "A Geospatial Information Portal for Emergency Management of Natural Disasters," in *IEEE International Geoscience and Remote Sensing Symposium*, 2009, vol. 2, pp. II-404 – II-407.
- [5] K. Poser and D. Dransch, "Volunteered Geographic Information for Disaster Management with Application to Rapid Flood Damage Estimation," *Geomatica*, vol. 64, no. 1, pp. 89–98, 2010.
- [6] M. Van der Kooij, "Flood Monitoring and Disaster Management Response," *GEOconnexion Int. Mag.*, pp. 26–28, Apr. 2013.
- [7] E. J. Plate, "Flood risk and flood management," *J. Hydrol.*, vol. 267, pp. 2–11, 2002.
- [8] M. F. Goodchild, "Citizens as sensors: the world of volunteered geography," *GeoJournal*, vol. 69, no. 4, pp. 211–221, Aug. 2007.
- [9] K. Poser and D. Dransch, "Volunteered geographic information for disaster management with application to rapid flood damage estimation," *Geomatica*, vol. 64, no. 1, pp. 89–98, 2010.
- [10] E. M. Mendiondo, "Flood Risk Management of Urban Waters in Humid Tropics: Early Warning, Protection and Rehabilitation," in *Workshop on Integrated Urban Water Managmt*, 2005, no. April.
- [11] M. C. Fava, E. M. Mendiondo, V. C. B. Souza, J. P. de Albuquerque, and J. Ueyama, "Proposta Metodológica para Previsão de Enchentes com uso de Sistemas Colaborativos," in *XX Simpósio Brasileiro de Recursos Hídricos*, 2013.
- [12] S. Schade, L. Díaz, F. Ostermann, L. Spinsanti, G. Luraschi, S. Cox, M. Nuñez, and B. De Longueville, "Citizen-based sensing of crisis events: sensor web enablement for volunteered geographic information," *Appl. Geomatics*, vol. 5, no. 1, pp. 3–18, 2013.
- [13] R. Munro, "Crowdsourcing for Pakistan flood relief," Aug-2010. [Online]. Available: <http://www.crowdfunder.com/blog/2010/08/crowdsourcing-for-pakistan-flood-relief>. [Accessed: 21-Nov-2013].
- [14] K. McDougall, "Using Volunteered Information to Map the Queensland Floods," in *Proceedings of the Surveying & Spatial Sciences Biennial Conference*, 2011.
- [15] L. Kaewkitipong, C. Chen, and P. Ractham, "Lessons Learned from the Use of Social Media in Combating a Crisis A Case Study of 2011 Thailand Flooding Disaster," in *Thirty Third International Conference on Information Systems*, 2012.
- [16] M. Zook, M. Graham, T. Shelton, and S. Gorman, "Volunteered Geographic Information and Crowdsourcing Disaster Relief: A Case Study of the Haitian Earthquake," *World Med. Heath Policy*, vol. 2, no. 2, p. 7, Jul. 2010.
- [17] B. De Longueville, R. S. Smith, and G. Luraschi, "OMG, from here, I can see the flames! - A use case of mining location based social networks to acquire spatio-temporal data on forest fires," in *Proceedings of the 2009 International Workshop on Location Based Social Networks*, 2009, pp. 73–80.
- [18] S. Roche, E. Propeck-Zimmermann, and B. Mericskay, "GeoWeb and crisis management: issues and perspectives of volunteered geographic information," *GeoJournal*, vol. 78, no. 1, pp. 21–40, Jun. 2011.
- [19] M. A. Erskine and D. G. Gregg, "Utilizing Volunteered Geographic Information to Develop a Real-Time Disaster Mapping Tool: A Prototype and Research Framework Utilizing Volunteered Geographic Information to Develop a," in *International Conference on Information Resources Management*, 2012.
- [20] B. De Longueville, G. Luraschi, P. Smits, S. Peedell, and T. De Groeve, "Citizens as Sensors for Natural Hazards: A VGI Intergration Workflow," *Geomatica*, vol. 64, pp. 41–59, 2010.
- [21] D. Miorandi, I. Carreras, E. Gregori, I. Graham, and J. Stewart, "Neutrality in Mobile Internet: Towards a Crowdsensing-based Citizen Observatory," in *IEEE International Conference on Communications*, 2013.
- [22] J. Rotava, E. M. Mendiondo, and V. C. B. Souza, "Simulação de Instabilidade Humana em Inundações: Primeiras Considerações," in *XX Simpósio Brasileiro de Recursos Hídricos*, 2013.
- [23] Ushahidi, "The Ushahidi Platform," Jan-2014. [Online]. Available: <http://ushahidi.com/products/ushahidi-platform>. [Accessed: 10-Jan-2014].
- [24] J. P. Royston, "Some Techniques for Assessing Multivariate Normality Based on the Shapiro-Wilk W," *J. R. Stat. Soc.*, vol. 32, pp. 121–133, 1983.
- [25] I. Olkin, *Contributions to Probability and Statistics: Essays in Honor of Harold Hotelling*. Stanford University Press, 1960.
- [26] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering: an introduction*. Norwell, MA, USA: Kluwer Academic Publishers, 2000.
- [27] G. H. Travassos, "Experimentação em Engenharia de Software: Fundamentos e Conceitos," in *VIII Experimental Software Engineering Latin American Network*, 2011.

# Lightweight Risk Management in Agile Projects

Edzreena Edza Odzaly<sup>1,2</sup>, Des Greer<sup>1</sup>, Darryl Stewart<sup>1</sup>  
{eodzaly01|des.greer|dw.stewart}@qub.ac.uk

<sup>1</sup>Queens University Belfast  
University Road  
Belfast, Northern Ireland  
UNITED KINGDOM

<sup>2</sup>Faculty of Information Technology and Quantitative  
Science (FTMSK)  
Universiti Teknologi MARA, Shah Alam  
MALAYSIA

*Abstract*— Risk management in software engineering has become a recognized project management practice but it seems that not all companies are systematically applying it. At the same time, agile methods have become popular, partly because proponents claim that agile methods implicitly reduce risks due to for example, more frequent and earlier feedback, shorter periods of development time and easier prediction of cost. Therefore, there is a need to investigate how risk management can be usable in iterative and evolutionary software development processes. This paper investigates the gathering of empirical data on risk management from the project environment and presents a novel approach to manage risk in agile projects. Our approach is based on a prototype tool, Agile Risk Tool (ART). This tool reduces human effort in risk management by using software agents to identify, assess and monitor risk, based on input and data collected from the project environment and by applying some designated rules. As validation, groups of student project data were used to provide evidence of the efficacy of this approach. We demonstrate the approach and the feasibility of using a lightweight risk management tool to alert, assess and monitor risk with reduced human effort.

*Keywords*- software risk, risk management, agile projects.

## I. INTRODUCTION

The Oxford dictionary defines ‘risk’ as an exposure to danger, harm or loss. Risk can also be seen as an event with a negative impact that may or may not occur in future. On the other hand, risk can also be positive and invites opportunities [29]. In estimating and measuring risk, Boehm [1] defines Risk Exposure as a fundamental concept that can be used to quantify risk: Risk Exposure (RE) = Prob(UO) x Loss(UO) where Prob (UO) refers to the probability of an unsatisfactory outcome and Loss (UO) refers to the impact of the unsatisfactory outcome. A lower risk exposure can be obtained by reducing the probability or by reducing the associated loss. Risk management is a process that involves identifying risk, assessing and prioritizing risk, as well as monitoring and controlling risk. Risk is a necessary evil in the software processes, even those that are claimed to inherently reduce risk, such as in agile methods [5].

## II. RESEARCH PROBLEMS

### A. Traditional Risk Management

More than a decade ago it was recognized that there should be a change and new risk management discipline developed [2]. This was around the same time as agile methods started to become popular. Little work has been done to date on the role of risk management in agile methods. This may be due to a common theme in research on risk management e.g. Higuera

and Haimes (1996) [3] stated that risk management is difficult to implement and is complex. The implication made is that existing heavyweight risk management is contrary and to the philosophy of agile. Thus, a lightweight or improved risk management method would be more likely to be adopted.

In a survey done in 2009 [4], we gathered responses from companies in Northern Ireland as part of an investigation of the barriers to risk management. The results concluded:

- There is no standard or commonly adopted risk management process and/or tool being used in every software development situations.
- Risk Identification was the most effort intensive process and 30% agreed that Risk Monitoring is most difficult and needs more effort.
- The most recognized barrier was that where visible (and tangible) development costs get more attention than intangibles like loss of net profit and downstream liability.

Overall, traditional risk management processes is multifaceted, complex and traditionally a heavyweight process.

### B. Risk Issues in Agile Software Projects

Due to the fact that agile methods depends a lot on the credibility of the people involved in the projects [5][6] as well as their motivation in applying the agile practices [7][8], most issues encountered relate to the people and the practices involved. This echoes one of the values in agile manifesto i.e. “*individuals and interactions over processes and tools*” (Agile Manifesto). This implies that not having the right people doing the right process will be a source of risk.

Cho [9] developed some research work on issues and challenges of agile software development with Scrum, among which the following points are discussed: (i) forming a Scrum team with relevant skills; (ii) one individual with multiple responsibilities and overloaded tasks; (iii) lack of accountability where team members do not take responsibility for delayed tasks, coupled with a lack of supervision and (iv) Daily scrum meeting and monthly sprint planning are considered to be a waste of time or taking too much time.

Cockburn and Highsmith [5] highlighted that one of the most important success factors in a project is individual competency emphasizing the qualities of the people involved in the project. This is also supported by Boehm and Turner

[10] where people issues are the most critical and it is very important to address them before adopting and integrating agile practices into a project. Deemer and Benefield [11], discuss challenges of the ability of a team to provide estimation of effort in their development work especially when it is done for the first time. There are also many studies addressing issues with agile skills and personnel turnover as well as job dissatisfaction [12][13][14]. Individual motivation is important in Scrum as this leads to the team adhering to agile process practices, for example attending Daily Scrum Meetings [16]. Team member behavior, where teams fail to comply with practices, can provide early sign of risks e.g. low morale expressed during the daily meeting or avoiding discussing problems when behind schedule [17].

### III. SOLUTION APPROACH

Based on the research problems discussed in the previous section, there is a strong motivation to improve the management of risk in agile projects without unduly threatening the agility of projects. In reality, contemporary risk management should be looked as an integral part of the agile process and decision making. This includes taking into account human factors such as developer skills and ability as well as their behavior in performing tasks. In the following subsections, we will explain the architecture of the Agile Risk Tool (ART) and the process flow of our approach.

#### A. The architecture of Agile Risk Tool(ART) Prototype

The ART model can be described in terms of two main architectural components:

(i) The *Agile Risk Tool* refers to the main engine of the tool which consists of the graphical user interface (GUI) for the Input and Output, the Rule engine and the ART agents. It interacts with the ART template, which is a template that is used to define the environment data. Once the ART template file that contains environment data is uploaded, this data can be modified using a GUI. Further explanation on ART template is discussed later.

(ii) *Environment Data* refers to the data from the project environment. Changes in this data stimulate dynamic reaction from the ART agents. To achieve this, an ART template must be created for the project environment and data from the real project environment must be translated into this template. The categories of data used for this work were ‘Project’, ‘Team’, ‘Task’ and ‘Progress’. Any risks triggered will be stored in a risk data repository so that this data can be used in future to support risk decisions.

#### 1) Agile Risk Tool elements

The architecture of the main engine of the Agile Risk Tool (ART) prototype has three main elements; the Input/Output, the ART agents, and the Rule engine.

a) *Input/Output*. Previously, issues in agile projects were discussed (Section B) and it was found that most of the problems related to the people involved and their motivation and skills in software development. Indeed Agile relies heavily on the competency of the people involved. Therefore we converted these issues to risk factors i.e. situations or events that may cause a loss to occur and therefore that we need to monitor in a project.

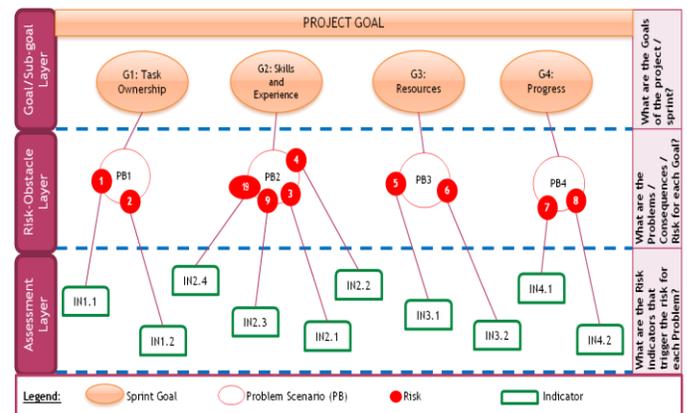
**Table 1: Mapping Problem Identified to the Sprint Goal of the Project**

Sprint Goal	Problems Identified	Identifier
In Sprint X, Task Y should be assigned to appropriate number of developers once Sprint X is started	<ul style="list-style-type: none"> <li>• Pair programming</li> <li>• No accountability or ownership</li> <li>• Collective code ownership</li> </ul>	G1: Task ownership
In Sprint X, Task Y should be assigned to a proper skilled team member	<ul style="list-style-type: none"> <li>• Not enough people skilled in agile / forming Scrum team with relevant skills</li> <li>• Insufficient agile training</li> </ul>	G2: Skills and Experience
In Sprint X, developer should focus on one role and one project at a time	<ul style="list-style-type: none"> <li>• Multiple responsibilities</li> </ul>	G3: Resources
In Sprint X, developer should attend Daily Scrum Meeting and provide task Y progress	<ul style="list-style-type: none"> <li>• Personnel turnover</li> <li>• Daily meetings in Scrum ceremonies – inefficient meeting, waste of time because sometimes involves more time than usual</li> </ul>	G4: Progress

Further, there is a need to specify the input for the project, which consists of the type of risks and the risk indicators as well as the environment data which can be used to identify the risks for the project. Thus the issues discussed earlier are transformed into a set of sprint goals (Table 1). These will later be used to define the risks and their indicators thus allowing risks to be monitored continuously.

In order to identify the sprint goal for this project, we grouped the list of issues found earlier and assigned an appropriate goal for each item. An identifier was assigned for each goal and this identifier is used throughout this work (Table 1). Sprint goal rather than project goal was used to allow each sprint to have different goals and different risks associated to each goal. For this project, we adopted and reused the first three layers of the GSRM model [18] for translating the project goals into problem scenarios (Figure 1).

At the top layer (Goal/Sub-goal layer) of the model, we developed a set of sprint goals. For this work, we proposed four sprint goals based on the problems identified (Table 1) and mapped this to the problem scenario and risks (Risk-Obstacle layer) that could possibly threaten the sprint goal. Further, we mapped accordingly between the risk and possible indicators (Assessment layer) that could later provide an alert which will trigger a risk.



**Figure 1: The Agile Risk Tool (ART) GSRM Model**

In a real world situation, each project might have different goals and risks associated to the goals. However, we limited the set of goals proposed for this work to the problems identified in the literature. An example of how the ART GSRM model is applied is shown below.

**Goal 1 (G1):** *In Sprint X, Task Y should be done in pairs*  
**Problem Scenario (PB1):** *During the sprint, the developer does not practice pair programming*  
**Risk 1:** *Pair programming is not applied, single expert risk occurs*  
**Indicator 1.1:** *When the sprint cycle is started, a task being selected in the sprint and the selected task has no pair with another developer*

b) *ART agents.* As discussed in the previous section, a lightweight risk management approach was introduced to reduce barriers to risk management application. One way to move towards automation is to give agents responsibility to identify, assess and monitor risk. These agents ideally should be able to autonomously react to environmental changes, where the environment in this case is the software development environment, including the set of tools being used.

There are four ART agents: Manager Agent, Identify Agent, Assess Agent and Monitor Agent. The goal and purpose of each of these is discussed below.

- *Manager Agent* acts as an intermediary between the other three agents. It manages and executes rules, notifies the Identify agent if any risk is triggered, gets data from the Environment and passes data requested from the agents.
- *Identify agent* is notified if any risk is triggered. It requests from the Manager agent what risk has been identified and notifies the Assess agent.
- *Assess agent* is invoked by the Identify agent and its goal is to estimate the Risk Exposure (RE) of the identified risk where  $RE = Probability (P) \times Impact (I)$ . The identified risk will then be ranked as High, Medium or Low and the Monitor agent is notified to take subsequent action.
- *Monitor agent* is invoked by the Assess agent with some data: RE and rank of the identified risk. The Monitor agent will establish the location of the identified risk along with the owner of the risk. These data are then displayed in the *Risk Register*.

The Risk Register acts as a screen to display all identified risk data. Data displayed in the Risk Register can be recorded and saved in the Risk data repository. The documented risk data can be used in future to plan and mitigate risks for the future projects.

**Table 2: Rule template**

<b>Goal</b>	The sprint goal that needs to be defined before the project starts
<b>Problem scenario</b>	A possible risk event that associated with the sprint goal
<b>Consequences</b>	The penalty if the risk is occurred
<b>Indicators</b>	The events or measures that forewarn of the risk event and their values
<b>Repository/ Data</b>	The list of repository of environment data involve in this risk event
<b>Rule(s)</b>	The list of rule(s) that trigger the risk event
<b>Risk name</b>	The unique name for the risk

c) *Rule engine.* In this subsection, we use two methods: risk drivers and generated rules to identify input for the Rule engine.

Earlier, we summarized problems and issues and mapped these to a set of sprint goals. In the ART GSRM model presented earlier, we derived a set of indicators for the identified problems and risks. As such, we used the indicators to generate rules to then develop inputs for the rule engine. The indicators are determined beforehand using the Rule template. Table 2 shows the template of a rule and risk drivers (indicators) for a problem scenario. Each problem scenario proposed one or more possible risk event that is associated with a sprint goal for the project. Sprint goal is important in the sense that using sprint goals to identify how this environment data could be used as indicators of threats to the goals and trigger the risks. Therefore, we generated the rules that use the indicators / risk drivers to identify events that cause loss (delay/extra cost/loss of value) i.e. risks. The problem scenario was derived from the research problem discussed in previous Section B and summarized in Table 1.

This also represents a standard template that allows the manager to define the Sprint Goal, Problem Scenario, Consequences, Indicators, Repository, Rules and Risk Name before the project starts. Each problem scenario proposes a possible risk event that is associated with a sprint goal for the project. Defining a sprint goal is important in that it allows us to identify how data from the development environment can serve as an indicator to a risk event in the project. Hence, the rules were generated using the indicators to identify risk based on the objects defined in the environment data. For this work we started by employing details from the ART GSRM model which consists of sprint goal, problem scenario, risks and indicators and further defined the consequence of each problem if the risk where to occur. Subsequently, the rule is constructed to produce an alert for the risk.

2) *The Environment data*

As far as the agile development process is concerned, XP and Scrum are the most widely used agile methodologies [19][20].

**Table 3: List of selected data used in this work**

Data/ Objects	List of Attributes
Product / Project*	Project ID / Name Project Start date / End date User Story ID / Name User Story Estimation User Story Priority User Story Owner ID / Name Task ID / Name Task Estimation Task Priority Task Owner ID / Name Task Status Task Paired By (Pair Programming)
People**	Team Member ID / Name Total no. of Role in a Project Total no. of Project Involved Programming Skill Level Agile Experience Level
Progress**	Attendance in Daily Standup Meeting Daily Progress Report on Assigned Task

\*Refers to data available in both EM and RS

\*\*Additional data collection that is required in this work

Further an agile development survey [20], stated that agile companies use a wide variety of agile tools and 60% of them are currently using an agile project management tool. The tool choice might be different ranging from a simple spreadsheet to commercial tools such as VersionOne. Tools help the project manager to have better visualization in delivering the project and in more interactive manner. Therefore, we studied two agile project management tools, Rally software (RS) and Extreme manager (EM), considering what is accessible in terms of configuring their product features and data design. In both cases information was gathered on the environment data model to include their process model, the objects used, and the attributes and values for those objects. The outcome from this was a more generalized definition of the available data.

Since the list of data available was massive, we have screened it and show only the data that are used, as shown in Table 3. Both tools when compared, have almost the same objects and attributes, although the naming convention used might be slightly different. These data were then translated into the ART template in form of objects and attributes, where the value of each attribute can be collected from the real project. The ART template represents as data within the project environment in which changes in the project environment are captured dynamically by the ART agents.

### B. ART Process

In order to support the ART process the ART prototype tool was developed and used to demonstrate the reactions of the ART agents towards the changes in the environment data following the execution of the set of rules built from consideration of goals. The ART process flow is depicted in Figure 2.

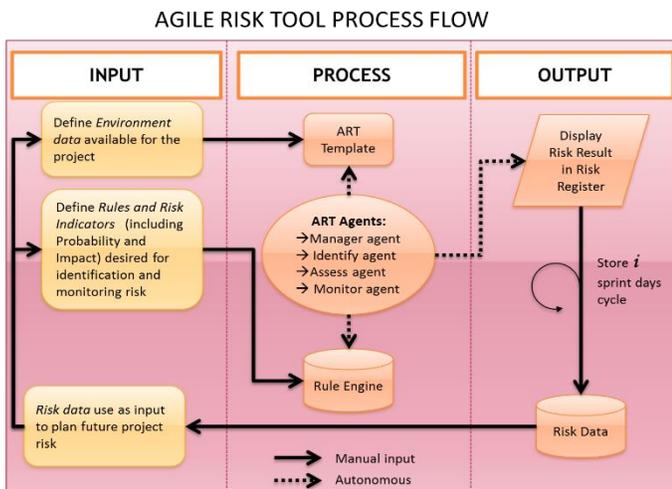


Figure 2: ART Process describing the flow or steps starting from defining the Input and storing the Output in the Risk data file

1) *Input*. This process is started at the Input stage. At this stage, there are two main inputs needed: defining the environment data available for the project and defining the rules and risk indicators to monitor risks.

#### a) Defining the environment data available for the project

Firstly, one should define what environment data is available in the project. Since we are constrained by the modern Software Engineering environment and the need for agility, we are limited to data can be easily collected. An

example of the form of environment data in a project includes user stories name and id, task name and id, task priority, task owner and so on. The collected data are then translated into the ART template. The ART template consists of objects and attributes used in this work as shown previously in Table 3. It contains data and values collected from the project environment.

#### b) Defining the rules and risk indicators

The definition of rules and risk indicators for the project allows one to identify what risks to monitor for this project and from which data or indicator that the risks could possibly be triggered. At this stage, one can either add new rule or add an existing rule where the rule was created from a previous project.

#### 2) Process

At the Process level, the ART agents will monitor the risk by acknowledging any rules or risk indicators triggered as informed by the ART template. The ART agents will initiate communication between them. Messages are passed according to request and each agent will notify another agent in prompting any further action to be taken.

Rules and the environment data are dynamically editable. In the event where changes need to be made, one can modify the environment data (which has been translated into the ART template earlier) as well as the risk rules and indicators using the provided main screen area. On the other hand, when developing possible risks associated with rules and risk indicators, one might find the environment data used to be insufficient to detect certain risks. In some cases, a small change in collection of the environment data would allow defining or detecting more risks. For example, adding the information on developer's skill will allow monitoring the developer's programming capability especially in completing high priority task. An example of a rule syntax that can be used is, "IF the developer skill level is 'Low' AND the developer involved with a 'High' priority task, THEN there is probability of a risk of the task cannot be completed on time because of the developer's poor programming skill".

ART agents will react dynamically to input data, process the input by assessing any risk triggered and produce a risk result in the Risk Register.

3) *Output*. At the Output stage, the risk data are stored in the Risk data repository. The risk data also can be captured daily up to the  $i$  no. of days in a sprint and the data will be saved in spreadsheet format. Later, one can use this risk data, analyse the risk according to project and use the analysed data as an input for identifying future project risk as shown earlier in Figure 2.

This application tries to support Continuous Risk Management (CRM) [21]. Applying the ART process accompanied with the designated tool will help the project manager to manage risk continuously. This is where, when changes take place in the environment data, these are captured by the ART agents who constantly run updates on the risk data and display the results in the risk register. As far as the CRM is concerned, manual implementation of this technique can be minimized and the monitoring is moved towards being autonomous.

#### IV. CASE STUDIES

Two case studies were used to validate the approach and tool support. The first case study was developed in 2011 involving 38 undergraduate students, assigned into 6 teams with 6 or 7 developers each. The second case study was developed in 2012 and involved a total of 56 undergraduate students with eight groups and each group consisting of 5 to 8 developers. The case studies were used to demonstrate the ART Process Flow (Figure 2) and the tool. All groups were required to develop the same product requirements. Likewise, all groups were given the same product requirements, in both case studies. In the theoretical part of the course, students received lectures on general agile development practices with an emphasis on Scrum. During the course, students were required to build a large software artefact using Microsoft.NET technologies using an industrial strength environment adopting both agile project and software engineering practices. This includes applying important Agile project management practices such as Pair Programming, Test Driven Development, Release and Iteration Planning and Refactoring in their software project.

The most vital part of the process is to determine its environment data and risk rules for the project. The sources used in this case study are as shown in Table 3, the actual physical sources being:

- *Hartmann-Orona Spreadsheet [15]* – a widely used spreadsheet tool, here providing data to capture Sprint Backlog Information on tasks, their estimates and progress status. It also provides team member information such as working time and activity details;
- *Sprint Backlog* – a document containing a list of user stories, story points and dependencies;
- *Scrum Minutes of Meeting (Daily)* – teams were required to keep data on ScrumMaster and work progress. In Agile Life Cycle tools this data would likely be available from the tool;
- *SVN [25] Repositories* - Directory and Files Versioning, Commit Files/Code, Details of commits and changes;
- *Source Code (C#) & Resharper [26]* quality metrics.

A small sample of the risk rules defined on the data using the ART prototype is shown in Figure 3.

Risk Name	Rules [Object.Attribute] == [Value]	Prob Score	Imp Score
Pair programming	PROJECT.PROJECTSTATUS = Completed, TASK.PAIREDBY = ""	3	5
Task ownership	PROJECT.PROJECTSTATUS = Completed, TASK.TOTALOWNED > 2	1	1
High priority task assigned to inappropriate team member cannot be completed on time	TASK.PRIORITY = High, TEAM.SKILLLEVEL = -1	5	4
Overload tasks can cause difficulty in time management	PROJECT.PROJECTSTATUS = Completed, TEAM.TOTALNOROLE > 1	5	1
Developer absent in meeting possible of employee turnover	PROJECT.PROJECTSTATUS = Completed, PROGRESS.DAILYMEETING = N	1	3
No progress report	PROJECT.PROJECTSTATUS =	1	3

	Completed, PROGRESS.IMPEDIMENTDATA = N		
Unable to understand agile process and meet the target	PROJECT.PROJECTSTATUS = Completed, TEAM.AGILE = false	3	3
Unable to comply with the agile process	TEAM.AGILE = true TEAM.AGILE_EXPERIENCE=Very Poor	1	1

Figure 3: Sample of Risk Rules used in Case study

Running the tool will then generate a risk repository at any point. Figure 4 shows a sample screen shot of the risk register showing risks that have been identified at the end of a sprint.

Given the novelty of this study, it was expected that there would be some issues raised while performing it. The first issue that we found at this stage was the difficulty in matching the data from the studied tools with the data available in the student project artefacts. The archived artefacts available however, did not provide as much data as the studied tools. Nevertheless, we found that the archived artefacts contained enough useful information, particularly related to the sprint backlog and the user stories, breakdown of the tasks, details of the developer responsible for a task and so on. In addition, the goal of the study was to demonstrate the approach and tool support, not applicability to every data item collected in mainstream tools.

Based on the lessons learnt from the first case study, some improvements were made to data collection but also rules were modified. One rule on pair programming rule was modified due to the observation that not using pair programming in very small tasks did not constitute a significant risk. The ability to modify the rules as needed demonstrates that the solution approach and tool support can dynamically respond to changes, as is required in agile projects.

Risk ID	Risk Name	Location	Owner	Risk Severity	Resolution
R0001	High risk of code is not collectively owned	TS060	MG	Critical risk	Assigned pair member
R0001	High risk of code is not collectively owned	TS057	MO	Critical risk	Assigned pair member
R0001	High risk of code is not collectively owned	TS050	MO	Critical risk	Assigned pair member
R0001	High risk of code is not collectively owned	TS029	MO	Critical risk	Assigned pair member
R0001	High risk of code is not collectively owned	TS011	WL	Critical risk	Assigned pair member
R0001	High risk of code is not collectively owned	TS058	MG	Critical risk	Assigned pair member
R0001	High risk of code is not collectively owned	TS062	MG	Critical risk	Assigned pair member
R0001	High risk of code is not collectively owned	TS059	MO	Critical risk	Assigned pair member
R0001	High risk of code is not collectively owned	TS061	MO	Critical risk	Assigned pair member
R0005	Overload tasks can cause difficulty in time management	TS007	WL	Moderate risk	Reduce overload task
R0005	Overload tasks can cause difficulty in time management	TS034	SO	Moderate risk	Reduce overload task
R0005	Overload tasks can cause difficulty in time management	TS013	AR	Moderate risk	Reduce overload task
R0005	Overload tasks can cause difficulty in time management	TS003	CM	Moderate risk	Reduce overload task
R0005	Overload tasks can cause difficulty in time management	TS032	SO	Moderate risk	Reduce overload task
R0006	Overload tasks can cause difficulty in time management	TS027	MG	Moderate risk	Reduce overload task

Figure 4: Risk Register

Risk data derived using the tool displayed in a Risk Register and can also be recorded and saved in the Risk Data Repository. A further output from the tool is a means of assessing the total risk in the project at any point or in a post sprint review as shown in Figure 5. This includes information on the breakdown of risk identified each day. Total Risk Score is a metric based on the generic severity score of a risk item and size of the task it is related to. It provides results on counting of risk daily and cumulative risk counting in a sprint.

Normally one would expect risk should decrease (burn down) over a sprint, but using Figure 5 a sprint review for Team Alp1 would demonstrate clearly that that particular sprint for that team was problematic. Further, the reports provide useful insights in correlating identified risks to agile practices.

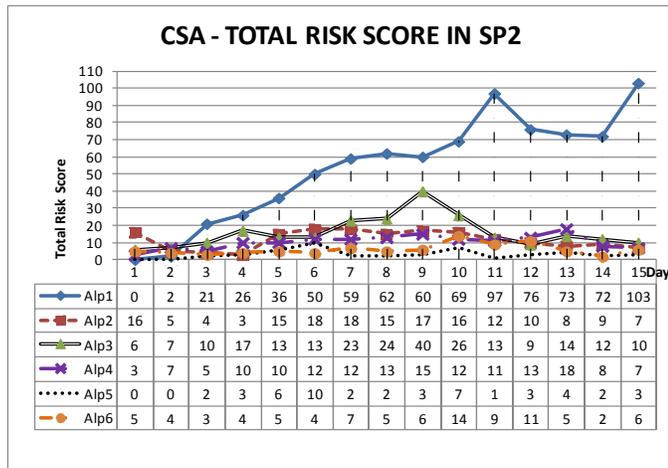


Figure 5: Total Risk Score graph of Case Study Alpha (Sprint 2)

## V. STUDY VALIDITY

Since the study presented introduces a new approach in managing risk in agile project, the main issues are focused on the internal threats. The first internal threat is in terms of the accuracy of the measured data, especially because the data used was based on historical artefacts. Further, confirmation of this data was not possible as the project had already been completed at the time of analysis. Secondly, the approach used entailed manual collection and translation of data from archived artefacts into the ART tool. This human effort was required before the ART agents could begin reacting towards environment data as they were designed to work in. This effort could be minimized by selecting a proper individual in the team to conduct this process, for example the Scrum Master in a Scrum project. One step taken to ensure the quality of the study was that cross checking was done from time to time with the Product owner to confirm perception based on his observation. Considering external validity threats, the risk management approach and tool supports were designed to be as general as possible so that this is applicable in general to agile project environments. This includes taking into account two popular agile project management tools studied for this work so that the approach is as applicable as possible to other contexts but also lightweight and unobtrusive to the team daily activities. Nonetheless, no claim can be made of good fit with tools not studied. Additionally, the study used student project data rather than industrial data. Hence, there will be arguments whether this is applicable to a real world environment.

## VI. CONCLUSION

In this paper we presented a novel approach to manage risk in agile projects. We provide a contemporary and lightweight risk management approach which consists of the ART prototype and process flow. We validated the approach by using student project artefacts. An interesting by product of the research is a series of observations on how non-compliance with agile principles can increase risk to a project and also negatively affect the quality of the software product. This will

be a subject of further analysis. In future, we plan to improve this approach by adding knowledge therefore helping in automatic learning and, decision support regarding risks, as well extending to other risk management steps.

## REFERENCES

- [1] Boehm, B.W. 1989, Tutorial: Software risk management, IEEE Computer Society Press.
- [2] Kontio, J. 2002, "Risk Management: What went wrong and what is the new agenda?," in Kontio, J. and Conradi, R. (Eds.) Software Quality - ECSQ 2002, Quality Connection - 7th European Conference on Software Quality".
- [3] Higuera, R.P. & Haimes, Y.Y. 1996, Software Risk Management.
- [4] Odzaly, E.E., Greer, D. & Sage, P. "Software risk management barriers: An empirical study", Empirical Software Engineering and Measurement, 2009. ESEM 2009. 3rd Interlocation.
- [5] Cockburn, A. & Highsmith, J. 2001, "Agile software development, the people factor", Computer, vol. 34, no. 11, pp. 131-133.
- [6] Nerur, S., Mahapatra, R. & Mangalaraj, G. 2005, "Challenges of migrating to agile methodologies", Communications of the ACM, vol. 48, no. 5, pp. 72-78.
- [7] Layman, L., Williams, L. & Cunningham, L. 2006, "Motivations and measurements in an agile case study", Journal of Systems Architecture, vol. 52, no. 11, pp. 654-667.
- [8] Conboy, K., Coyle, S., Wang, X. & Pikkarainen, M. 2010, "People over process: key people challenges in agile development", IEEE Software.
- [9] Cho, J. 2008, "Issues and Challenges of agile software development with SCRUM", Issues in Information System, vol. 9, no. 2.
- [10] Boehm, B. & Turner, R. 2005, "Management challenges to implementing agile processes in traditional development organizations", Software, IEEE, vol. 22, no. 5, pp. 30-39.
- [11] Deemer, P., Benefield, G., Larman, C. & Vodde, B. 2010, "The scrum primer", <http://assets.scrumtraininginstitute.com/downloads/1/scrumprimer121.pdf>, vol. 1285931497 accessed 14 March 2014.
- [12] Boehm, B. & Turner, R. 2003, "Using risk to balance agile and plan-driven methods", Computer, vol. 36, no. 6, pp. 57-66.
- [13] Melnik, G. & Maurer, F. 2006, "Comparative analysis of job satisfaction in agile and non-agile software development teams" in Extreme Programming and Agile Processes in Software Engineering Springer, pp. 32-42.
- [14] Melo, C., Cruzes, D.S., Kon, F. & Conradi, R. 2011, "Agile team perceptions of productivity factors", Agile Conference (AGILE), 2011 IEEE, pp. 57.
- [15] Hartmann-Orona Scrum Spreadsheet, [www.bryanavery.co.uk/file.axd?file=2009%2F5%2FSprint+Backlog.xls](http://www.bryanavery.co.uk/file.axd?file=2009%2F5%2FSprint+Backlog.xls), accessed 14 March 2014.
- [16] Hossain, E., Babar, M.A., Paik, H. & Verner, J. 2009, "Risk identification and mitigation processes for using Scrum in global software development: A conceptual framework", Software Engineering Conference, 2009. APSEC'09, Asia-Pacific, IEEE, pp. 457.
- [17] Lindvall, M., Basili, V., Boehm, B., Costa, P., Dangle, K., Shull, F., Tesoriero, R., Williams, L. & Zelkowitz, M. 2002, "Empirical findings in agile methods" in Extreme Programming and Agile Methods—XP/Agile Universe 2002 Springer, pp. 197-207.
- [18] Islam, S. 2009, "Software development risk management model: a goal driven approach", Proceedings of the doctoral symposium for ESEC/FSE on Doctoral symposium, ACM, pp. 5.
- [19] Ambler, S. 2006, Results from Scott Ambler's 2006 Agile Adoption Rate Survey, [www.ambysoft.com](http://www.ambysoft.com), accessed 14 March 2014.
- [20] VersionOne 2013, "7th Annual State of Agile Development Survey", [www.versionone.com](http://www.versionone.com), accessed 14 March 2014.
- [21] Dorofee, A.J., Walker, J.A., Alberts, C.J., Higuera, R.P. & Murphy, R.L. 1996, Continuous Risk Management Guidebook.
- [22] Runeson, P. & Höst, M. 2009, "Guidelines for conducting and reporting case study research in software engineering", Empirical Software Engineering, vol. 14, no. 2, pp. 131-164.
- [23] Wohlin, C., Host, M., Henningsson, K., 2003, "Empirical Research Methods in Software Engineering," Empirical Methods and Studies in Software Engineering, R. Conradi and A. Wang, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, pp 7-23.
- [24] Yin, R. K., 2009, "Case study research: Design and methods", vol. 5. SAGE Publications.
- [25] Subversion, <http://subversion.apache.org/>, accessed 14 March 2014.
- [26] JetBrains, Resharper, <http://www.jetbrains.com/resharper/>, accessed 14 March 2014.
- [27] Rally, <http://www.rallydev.com>, accessed 14 March 2014
- [28] Hindsa, Extreme Manager, <http://www.hindsa.com>, accessed 14 March 2014
- [29] Snyder, C. S., 2013, A User's Manual to the PMBOK Guide. John Wiley & Sons.

# Snowball Effects on Risk Mitigation Scheduling

## Process and Tool

Hareton K. N. Leung

Department of Computing  
The Hong Kong Polytechnic University  
Hong Kong, China  
hareton.leung@polyu.edu.hk

Kim Man Lui

Department of Computing  
The Hong Kong Polytechnic University  
Hong Kong, China  
cskmlui@comp.polyu.edu.hk

Peng Zhou

School of Computer  
Dongguan University of Technology  
Dongguan, China  
zhoupeng@dgut.edu.cn

**Abstract**—Risk management is widely accepted as a routine activity in software project management. Many project risks are often complex and intertwined. Once some risks are materialized, they spawn active issues which can cause some other risks to occur. Moreover, these risks will have stronger impact than they could happen alone. Such snowball effect has been well known in risk management; however, few studies have addressed their impacts to risk mitigation plan. This paper will apply the autoregressive model to estimate the snowball effect on six risk mitigation strategies.

**Keywords**—Scheduling strategy, Risk mitigation, Time element, Risk management, Software project management

### I. INTRODUCTION

The positive correlation between effective risk management and project success was emphasized in [1, 2, 3]. The adoption of risk management practices can help to increase the success rate of project and then enhance the competitiveness of organizations. Risk factors are often intertwined [4] making any assumption of their independence underestimate the complexity of risk problems in the end. When we consider that the impact of a risk is brought forward from some other previously materialized risks, which is named as “snowball effects.” In order to manage this kind of accumulated effect, one may add a number of risk review sessions from time to time. For example, agile software development, which is a highly iterative process, intends to reduce risk interconnectedness or snowball effect in different development phases [5].

Risk mitigation is essential for risk management because it aims to reduce or eliminate risks. To make the best use of project resources and to prioritize which risk factors should be dealt with, a scheduling strategy for risk mitigation is needed to determine the risks to be mitigated and when to mitigate them. The three basic strategies for scheduling risk mitigation are risk value first strategy, emergency first strategy and lowest-effort first strategy [6]. For easy reference, these strategies are denoted as “V”, “E” and “L” in this paper, respectively. As in V strategy, risks are scheduled for mitigation according to their risk values (i.e. Risk Exposure) [7] so that risks with higher risk values will be treated earlier. As in E strategy, risks are scheduled for mitigation according to possible time they may occur. As in L strategy, the mitigation is based on how much effort we will need to control each risk factor so that higher productivity throughput can be achieved earlier. Although a

risk mitigation plan in reality is a highly customized combination of V, L and E based on not only resources but also project politics and project experience, understanding the three basic strategies is still helpful.

How the three mitigation strategies perform in different situations has been studied through a computer simulation [6]. However, one unrealistic assumption has been made: risk factors are independent and a risk can only be considered mitigated as far as the mitigation process for that risk is done. Such limitation is huge and it hinders the development of a risk software management tool in practice, in which users work out a list of risk factors with probability and impacts, and a customized mitigation plan, the tool will perform the simulation and address the overall impact as feedback.

This paper aims to investigate different scheduling strategies for risk mitigation by a computer simulation. The impact of risks is accumulated as an autoregressive model AR(1) [8] and a partial mitigation is allowed.

The paper is organized as follows. We briefly review the risk process and simulation model of risk management process (SMRMP) in section 2. In section 3, we formally present different scheduling strategies, the autoregressive model of risk management process and partial risk mitigation. The next section compares the performance of identified strategies with or without snowball effects. In the final section, we conclude our study.

### II. BACKGROUND

#### A. Risk Management Process and Simulation Tool

Risk management aims to identify risks and take actions to reduce or eliminate their probability and/or impact so that the project is kept from being damaged by risks. There are many models and standards to guide the risk management practice, such as risk management paradigm developed by Software Engineering Institute, PMI framework, IEEE Std 1540, AS NZS 4360 and ISO 31000. Although these models and standards address the risk management processes in different manners, they can be mapped to each other to a large extent. Generally, these paradigms, models and standards follow the cyclic process shown in Fig. 1.

Risk Management Planning defines how to conduct risk management practices throughout the project. Risk

identification aims to identify risks that would affect the project objectives and document their characteristics. It is expected that a list of risk factors can be identified at this stage. The risk analysis aims to understand the identified risks and provide data to assist in managing them. Generally, risk analysis includes: (1) estimate the probability, impact, and the expected timing of the risk; (2) establish a risk mitigation plan. Risk monitoring and control aims to tracking the change of all identified risks and identifying new risks, monitoring residual risks, and evaluating risk response effectiveness and performance of risk management.

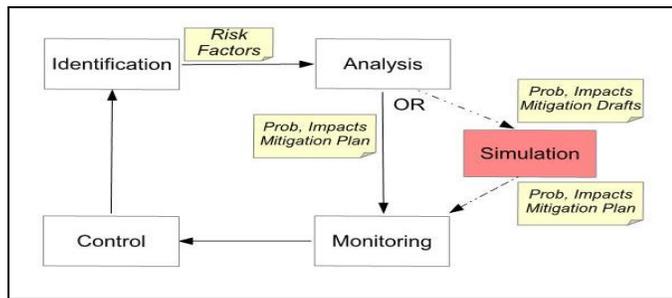


Figure 1. Cyclic Process of Risk Management and Simulation Tool

When a risk simulation tool is in place, it enhances the cyclic process of risk management shown in the right part of Fig 1. A risk manager can input several risk mitigation plans together with a list of detailed risk factors and the overall impact for each draft on the project can be estimated. He/she may then revise and finalize the mitigation plan. As mentioned, a risk mitigation plan often takes project politics and organization culture into account. Thus it may not be practical to have a fully optimized plan generated by a system. This paper will propose an autoregressive model for risk simulation.

### B. Simulation Model of Risk Management Process (SMRMP)

Few studies have explicitly modeled the time elements of risk. [6] proposed variants of risk, presented a model of risk lifecycle, and gave the relationship between the risk variants by explicit consideration of the occurrence time of risk.

A stochastic simulation model of risk management process called SMRMP with due consideration of time elements of risks is defined in [6]. The simulation model can be used for many risk management issues, such as understanding of risk management process, predicting risk management outcome, and making informed risk management decision.

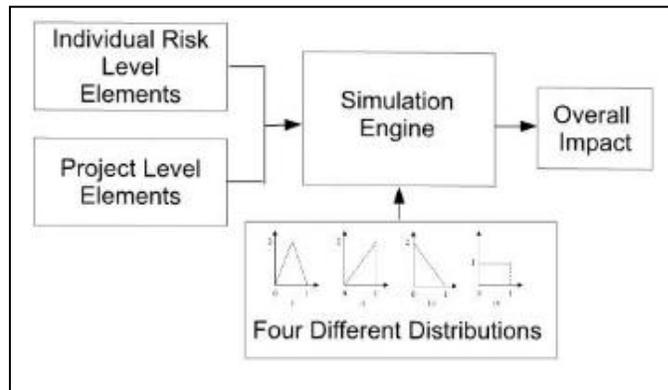


Figure 2. Architecture of SMRMP

Fig. 2 shows the “Simulation Model of Risk Management Process” (SMRMP). The first level is the risk level which focuses on a single risk. The second level is the project level which considers all risks of the whole project. There are four different distributions to generate the random number. Algorithms are also developed to compute output of the simulation from the input parameters.

There is one major limitation. Each risk is independent to each other. The simulation process is a strictly static and the expected variance is constant.

## III. METHODOLOGY

### A. Modeling of Risk Mitigation

A project may have a number of phases and each phase can have its cycle of risk management. It is possible that one cycle of risk management crosses the whole project. For easy explanation, we will consider one project phase. Although each project phase has different length of time, it can be easily normalized so that each phase has 100 time units. A risk ( $r$ ) occurrence period will be a closed time interval  $[t_{eo}, t_{lo}]$  within the phase. The risk can only be materialized within its occurrence period. Surely some risk will cover the whole length; however, the more precise information can be defined, the better estimate can be obtained. The probability ( $p$ ) to occurrence of a risk is 0 to 1. Each risk impact scale ( $i$ ) is 0 to 1. The mitigation effort ( $eff$ ) to input is person-day and it can be converted into person-time-unit as a project phase always has 100 time units.

Once a detailed list of risks has been identified, a risk manager based on the manpower resources can work out a risk mitigation plan. Fig 3 illustrates the relationships between a risk mitigation plan and a risk. In Fig 3, a risk  $r_i$  with  $[t_{eo}=40, t_{lo}=90]$  and  $eff=20$  person-unit-time has been identified at the stage of analysis of a cyclic process of risk management given in Fig 1. Assume that the risk would be materialized and identified at time  $k$ . There are four cases of a risk mitigation plan for  $r_i$ . Case 1 is preferred. The duration of case 2 has passed  $t_{eo}$  which stands for a chance that the risk  $r_i$  might be materialized before its full mitigation. Case 3 illustrates that the mitigation for  $r_i$  is not yet complete. In this situation, we consider that the impact has been proportionally lessened. Case 4 is unfavorable since the  $r_i$  occurs at time  $k$ .

The work is supported by Hong Kong Polytechnic University grant G-YK27.

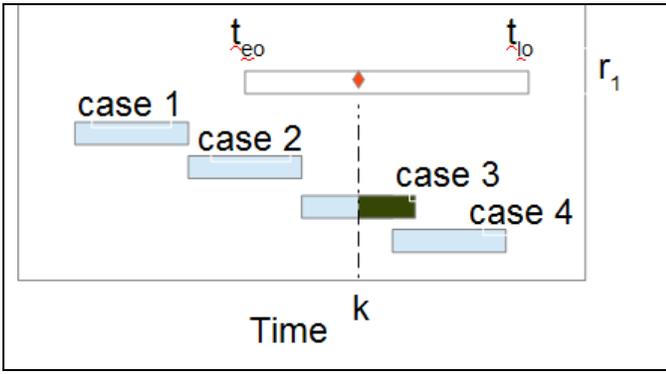


Figure 3. Cases for a mitigation plan

From Fig 3, we know that how the mitigation plan is defined could affect the outcome. Although mitigation plans involve a number of factors including organization culture and project politics to dominate which risks should be mitigated, it will be still practical and beneficial that a risk manager defines several draft plans and a simulation engine based on an autoregressive model estimates the overall impacts for each draft.

In this paper, we will adopt three different risk mitigation strategies to develop six mitigation plans. The objective is to understand whether some of six mitigation plans may substantially affect the overall impact as outliers. These plans should be taken out of consideration. It should be noted that one should not look for an optimized mitigation plan. It is impractical because many risk factors and values in a project are subjectively defined. Their values are not objectively obtained by measurement.

The three mitigation strategies are discussed below.

1) *Risk value first strategy (V strategy)*: It is straightforward. Risks are prioritized by their impact scales. The V strategy does not consider the time elements or efforts taken for mitigation.

2) *Emergency first strategy (E strategy)*: We order all risks according to their  $t_{eo}$ , and then risks with an earlier  $t_{eo}$  will be treated earlier. For example, a risk with  $t_{eo}=30$  will be mitigated before one with  $t_{eo}=40$ . This strategy attempts to mitigate the risk before it would occur.

3) *Lowest effort first strategy (L strategy)*: We schedule all risks according to the efforts needed for mitigating the risk, then risks requiring a lower effort will be treated earlier. For example, a risk with  $eff=40$  person-unit-time will be handled before one with  $eff=80$ . This strategy can mitigate more risks within the same time period because mitigating a risk with lower effort will use less time

The combination of V, E and L generates three more strategies as VE, VL and EL. For example, EL is defined as from  $i \cdot t_{eo}$ . In this paper, we adopt a naïve approach as our goal is to discover any significant difference between strategies after the snowball effect by AR(1) rather than optimization for mitigation strategy.

## B. Snowball Effects Simulation

Snowball effect is a figurative term for a process that accumulates from earlier effects or impacts and builds upon them. In this regard, the autoregressive model AR(q) [8] is adopted. The model is a stochastic process in which the variable at time k depends linearly on its own previous values at k-1. It is a special case of the more general ARIMA(0,0,q) model of time series. It has been used to describe certain time-varying processes in nature and economics. The AR(q) model is generally defined as:

$$X_t = c + \sum_{i=1}^p \varphi_i X_{t-i} + \varepsilon_t \quad (1)$$

where  $\varphi_1, \dots, \varphi_p$  are the parameters of the model, c is a constant, and  $\varepsilon$  is white noise.

In this paper, the snowball effect to accumulate the overall impact is an AR(1) model defined as:  $S_k = \alpha S_{k-1} + i_k - m_k$ , where  $S_k$  is the accumulated project impact at time k,  $\alpha$  is a constant,  $i_k$  is the planned impact scale of an identified risk at time k and  $m_k$  is an mitigation effect for  $i_k$ . Fig. 4 illustrates a snowball simulation with k consecutive integers from 1 to 100,  $\alpha=1$ ,  $i_k = |N(0,1)|$  and  $m=0$ .

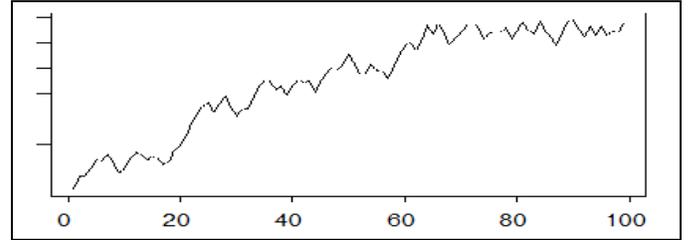


Figure 4. Snowball Simulation Using AR(1)

## IV. PERFORMANCE OF RISK MITIGATION STRATEGIES

### A. Results of Simulation

We generated 1000 projects with different risk parameters. According to the risk sets and six scheduling strategies, we establish a set of six mitigation plans for each project. The implementation was written in R which is a tool for statistical computing and provides libraries for the AR process and quantitative risk management. The implementation platform was R Cloud Workbench. The service is provided by European Bioinformatics Institute [9].

The result is given in Fig 5. We include a normalized curve, labeled “No Mitigation”, which accumulated all the planned impacts along time without mitigation or snowball effect as a baseline.

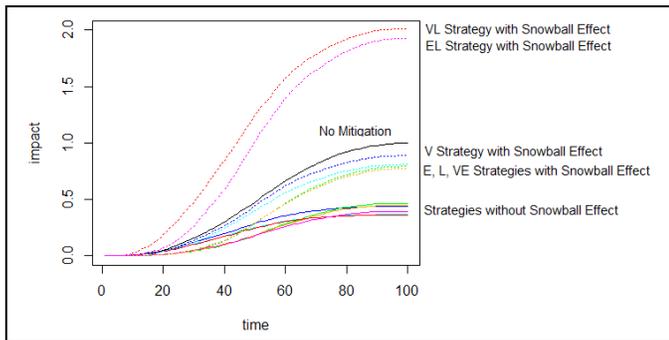


Figure 5. Normalized Impacts by Autoregressive Model AR(1)

### B. Discussion

Without snowball effect, the impact range of those strategies is 0.24, calculated by (Highest – Lowest)/Lowest. This is consistent with the results of average overall impact (Zhou and Leung, 2012), in which (Highest – Lowest)/Lowest is 0.26. Although we can do the ranking in order to select better strategies, it may not be very practical as (1) many (if not all) risks are subjectively defined by a risk team and they are not objectively measured variables (2) the overall impacts vary in a range smaller than three standard deviations provides little meaning that there is a statistical significant difference between them.

Once the snowball effect is considered, two groups “E, L, VE & V” and “VL & EL” are formed. Their difference is substantial and the VL and EL can be considered as outliers. It is recommended that a risk manager should not select VL or EL.

### V. CONCLUSION

Modeling and simulation for the purpose of software development risk management has been considered as quite limited [10]; however, this paper proposes a practical way to enhance software development risk management. Given a cyclic process of risk management in Fig 1, a risk team defines a set of detailed risks and several mitigation draft plans. The team can adopt our proposed simulation engine as a tool to

estimate which mitigation plans should be taken out. The risk manager can select one of the remaining plans and then finalize the plan before execution. The approach maximizes the usage of risk information. Although risk items and mitigation plan are often subjectively defined, it is valuable and effective to reduce unpleasant surprises [11, 12]. In this paper, we are even able to quantitatively compare the overall impacts of different mitigation plans on the project so that worse plans can be eliminated. In addition, the paper illustrates that the simulation engine should include the snowball effect; otherwise, many mitigation plans will have no statistical significant difference. As the nature of risk is uncertain, such simulation engine provides little practical values to risk management process in reality. With snowball effects, mitigation strategies can vary significantly. We can be better informed by the tool and take out those mitigation plans when their overall impacts are huge.

### REFERENCES

- [1] F. J. Heemstra & R. J. Kusters, “Dealing with risk: a practical approach,” *Journal of Information Technology*, vol. 11, 1996, pp. 333-346.
- [2] T. Lister, “Risk management is project management for adults,” *IEEE Software*, vol. 14, no. 13, 1997, pp. 20-22.
- [3] S. A. Sherer, “Managing risk beyond the control of IS managers: the role of business management,” *Proceedings of 37th Hawaii International Conference on System Sciences*. Hawaii, 2004.
- [4] T. W. Kwan and H. K. N. Leung, “A risk management methodology for project risk dependencies,” *IEEE Transactions on Software Engineering*, vol. 37, no. 5, 2011, pp. 635-648.
- [5] P. G. Smith and R. Pichler, “Agile risks/agile rewards,” *Software Development Magazine*, April, 2005, pp. 50-53.
- [6] P. Zhou and H. K. N. Leung, “A stochastic simulation model for risk management process,” *19th Asia-Pacific Software Engineering Conference (APSEC 2012)*, Hong Kong, 2012.
- [7] B. W. Boehm, *Software Risk management*, IEEE Computer Society Press, 1989.
- [8] A. R. Pagan, and A. Ullah, “The econometric analysis of models with risk terms,” *Journal of Applied Econometrics*, vol 3, no. 2, 1988, pp. 87-105.
- [9] R cloud, <http://www.ebi.ac.uk/Tools/rccloud/>, 2014
- [10] D. X. Houston, G. T. Mackulak, and J. S. Collofello, “Stochastic simulation of risk factor potential effects for software development risk management,” *Journal of Systems and Software*, vol 59, no 2, 2001, pp. 247-257.
- [11] M. Keil, P. E. Cule, K. Lyytinen and R. C. Schmidt, “A framework for identifying software project risks,” *Communications of ACM*, vol 41, no. 11, 1998, pp. 76-83.
- [12] S. Murthi, “Preventive risk management software for software projects,” *IT Professional*, vol. 4, no. 5, 2002, pp. 9-15.

# Reasoning at Runtime using time-distorted Contexts: A Models@run.time based Approach

Thomas Hartmann\*, Francois Fouquet\*, Gregory Nain\*, Brice Morin<sup>‡</sup>, Jacques Klein\* and Yves Le Traon\*

\*Interdisciplinary Centre for Security, Reliability and Trust (SnT), University of Luxembourg, Luxembourg, first.last@uni.lu

<sup>‡</sup>SINTEF ICT Norway, Norway, first.last@sintef.no

**Abstract**—Intelligent systems continuously analyze their context to autonomously take corrective actions. Building a proper knowledge representation of the context is the key to take adequate actions. This requires numerous and complex data models, for example formalized as ontologies or meta-models. As these systems evolve in a dynamic context, reasoning processes typically need to analyze and compare the current context with its history. A common approach consists in a temporal discretization, which regularly samples the context (snapshots) at specific timestamps to keep track of the history. Reasoning processes would then need to mine a huge amount of data, extract a relevant view, and finally analyze it. This would require lots of computational power and be time-consuming, conflicting with the near real-time response time requirements of intelligent systems. This paper introduces a novel temporal modeling approach together with a time-relative navigation between context concepts to overcome this limitation. Similarly to time distortion theory, our approach enables building time-distorted views of a context, composed by elements coming from different times, which speeds up the reasoning. We demonstrate the efficiency of our approach with a smart grid load prediction reasoning engine.

**Keywords**—Temporal data, Time-aware context modeling, Knowledge representation, Reactive systems, Intelligent systems

## I. INTRODUCTION

An intelligent system needs to analyze both its surrounding environment and its internal state, which together we refer to as the *context* of a system, in order to continuously adapt itself to varying conditions. Therefore, building an appropriate context representation, which reflects the current context of a system is of key importance. This task is not trivial [1] and different approaches and languages are currently used to build such context representations, *e.g.* ontologies [2] or DSLs [3]. Most approaches describe a context using a set of concepts (also called classes or elements), attributes (or properties), and the relations between them. Recently, in the domain of model-driven engineering, the paradigm of models@run.time [4], [5] has rapidly proved its suitability to safely represent, reason about and dynamically adapt a running software system. Nevertheless, context representations (or models), as abstractions of real system states and environments, are only able to reflect a snapshot of a real system at a specific timestamp. However, context data of intelligent systems rapidly change and evolve over time (at different paces for each element) and reasoning processes not only need to analyze the current snapshot of their contexts but also historical data.

Let us take a smart grid as an example. Due to changes in the production/consumption chain over time, or to the sporadic availability of natural resources (heavy rain or wind), the properties of the smart grid must be continuously monitored and adapted to regulate the electric load in order to positively impact costs and/or eco-friendliness. For instance, predicting the electric load for a particular region requires a good understanding of the past electricity production and consumption in this region, as well as other data coming from the current context (such as current and forecast weather). Since the electrical grid cannot maintain an overload for more than a few seconds or minutes [6], it is important that protection mechanisms work in this time range. This is what we call *near real-time*.

It is a common approach for such systems to regularly sample and store the context of a system at a very high rate in order to provide reasoning algorithms with historical data. Fig. 1 shows a context —represented as a graph (inspired by object graphs)— sampled at three different timestamps,  $t_i$ ,  $t_{i+1}$ , and  $t_{i+2}$ . Each graph in the figure represents the context at a given point in time, where all context variables, independently from their actual values, belong to the same time. Therefore, each graph lies in a horizontal plane (in time).

This systematic, regular context sampling, however, yields to a vast amount of data and redundancy, which is very difficult to analyze and process efficiently. Moreover, it is usually not sufficient to consider and reason just with data from one timestamp, *e.g.*  $t_i$  or  $t_{i+1}$ . Instead, for many reasoning processes, *e.g.* to investigate a potential causality between two phenomena, it is necessary to simultaneously consider and correlate data from different timestamps (*i.e.*  $t_i$  and  $t_{i+1}$ ). Reasoning processes therefore need to mine a huge amount of data, extract a relevant view (containing context elements from different snapshots), and analyze this view. This overall process would require some heavy resources and/or be time-consuming, conflicting with the near real-time response time requirements such systems usually need to meet.

Going back to the smart grid reasoning engine example: In order to predict the electric load for a region, a linear regression of the average electric load values of the meters in this region, over a certain period of time, has to be computed. Therefore, reasoning processes would need to mine all context snapshots in this time period, extract the relevant meters and electric load values, and then compute a value.

To address these issues, we propose to make context models aware of time *i.e.* to allow context elements (data) from different timestamps in the same model. We refer to such contexts as *time-distorted* contexts. Fig. 2 shows such a context

The research leading to this publication is supported by the National Research Fund Luxembourg (grant 6816126) and Creos Luxembourg S.A. under the SnT-Creos partnership program.

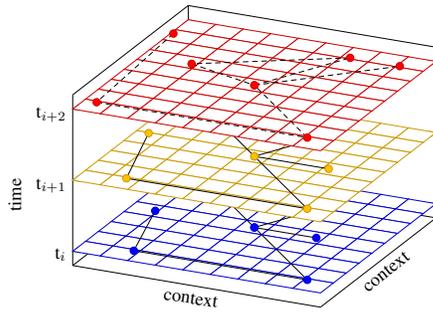


Fig. 1. Linear sampled context

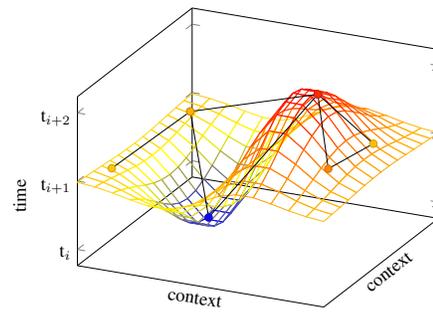


Fig. 2. Time-distorted context

representation, again represented as a graph. Here, the context variables —again independently from their actual values— belong to different timestamps. Such a context can no longer be represented as a graph lying entirely in one horizontal plane (in time). Instead, graphs representing time-distorted contexts lie in a curved plane. They can be considered as specialized *views*, dedicated for a specific reasoning task, composing navigable contexts to reach elements from different times. In contrast to the usage of the term *view* in database communities we do not aggregate data but offer a way to efficiently traverse specific time windows of a context.

Physics, and especially the study of laser [7], relies on a time distortion [8] property, specifying that the current time is different depending on the point of observation. Applied to our case, this means that context elements can have different values depending on the origin of the navigation context, *i.e.* depending on the timestamp of the inquiring actor. **We claim that time-distorted context representations can efficiently empower continuous reasoning processes and can outperform traditional full sampling approaches by far. The contribution of this paper is to consider temporal information as a first-class property crosscutting any context element, allowing to organize context representations as time-distorted views dedicated for reasoning processes, rather than a mere stack of snapshots. We argue that this approach enables many reasoning processes to react in near real-time (the range of milliseconds to seconds).**

The remainder of this paper is as follows. Section II introduces the background of this work. Section III describes the concepts of our approach and section IV the details on how we implement and integrate these into the open source modeling framework KMF. The provided API is presented in section V. We evaluate our general approach in section VI on a concrete smart grid reasoning engine for electric load prediction. After a discussion about the approach and related work in section VII the conclusion of the paper is presented in section VIII.

## II. BACKGROUND

Over time different languages, formalisms, and concepts to build and represent the context of intelligent systems have been developed and used [1], [9], [10] for different purposes. Entity-relationship models [11], as a general modeling concept for describing entities and the relationships between them, are widely used for building context representations. Ontologies, RDF [12], and OWL [13] are particularly used in the domain of

the Semantic Web. These allow to describe facts in a *subject-predicate-object* manner and provide means to reason about these facts. Over the past few years, an emerging paradigm called *models@run.time* [4], [5] proposes to use models both at design and runtime in order to support intelligent systems. At design time, following the model-driven engineering (MDE) paradigm [14], models support the design and implementation of the system. The same (or similar) models are then embedded at runtime in order to support the reasoning processes of intelligent systems, as models offer a *simpler, safer and cheaper* [15] means to reason. Most of these approaches (RDF, OWL, models) have in common that they describe a context using a set of concepts (also called: classes, types, elements), attributes (or properties), and the relations between them. We refer to the representation of a context (set of described elements) as a *context model* or simply as *model* and to a single element (concept) as *model element* or simply as *element*. The concepts of our approach are, in principle, independent of a concrete context representation strategy. However, the implementation of our approach and the provided API are build on a *models@run.time* based context representation approach and are integrated into an open source modeling framework, called Kevoree Modeling Framework [16] (KMF<sup>1</sup>). KMF is the modeling pillar supporting the Kevoree *models@run.time* platform [17]. We decided to leverage a *models@run.time* based approach for several reasons: First of all, models provide a semantically rich way to define a context. Second, models can be used to formally define reasoning activities. Last but not least, the *models@run.time* paradigm has been proved to be suitable to represent this context during runtime [4], [5].

KMF is an alternative to EMF [18] and specifically designed to support the *models@run.time* paradigm in terms of memory usage and runtime performance. Two properties of KMF are particularly important for the implementation of our approach: First, in KMF, each model element can be accessed within the model by a path (from the root element of a model to a specific element following containment [19] references), which defines the semantic to efficiently navigate in the model. Our contribution extends the path of model elements with temporal data in order to provide seamless navigation, not only in the model but as well in time. Second, in KMF each model element can be serialized independently, using paths to represent elements of its relationships. We use this property in our implementation to incrementally store model element modifications.

<sup>1</sup><http://kevoree.org/kmf>

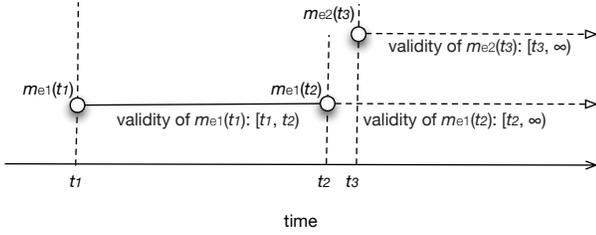


Fig. 3. Continuous validity of model elements

### III. ENABLING TIME-DISTORTED CONTEXTS

Our hypothesis is that temporal knowledge is part of a domain itself (e.g. electric load or wave propagation prediction, medical recommender systems, financial applications) and that defining and navigating temporal data directly within domain contexts is far more efficient and convenient than regularly sampling a context and independently querying each model element with the appropriate time. Therefore, we provide a concept to define and navigate into the time dimension of contexts. Most importantly, we enable a context representation to seamlessly combine elements from different points in time, forming a *time-distorted* context, which is especially useful for time related reasoning. We claim that our approach, which weaves time directly into the context model, as opposed to a full sampling and classic data mining approach, is compatible with near real-time requirements. In the following we present the concepts to enable time-distorted context models.

#### A. Temporal Validity for Model Elements

Instead of relying on context snapshots, we define a context as a continuous structure. Nevertheless, each context (model) element of this structure can evolve independently. We first define an implicit *validity* for model elements. Therefore, we associate a timestamp to each model element. They reflect a domain-dependent notion of the time at which a model element was captured and can be accessed on the element itself. In other words, a timestamp defines a *version*  $v_{m_e}(t)$  of a model element  $m_e$  at a time  $t$ . If a model element now evolves, an additional version of the same element is created and associated to a new timestamp (the domain time at which the new version is captured). Timestamps can be compared and thus form a chronological sequence. Therefore, although timestamps are discrete values, they logically define intervals in which a model element can be considered as *valid* with regards to a timestamp. A model element is valid from the moment it is captured until a new version is captured. New versions are only created if necessary, i.e. if the model element changed. Fig. 3 shows two model elements,  $m_{e1}$  with two versions and  $m_{e2}$  with one version, and their corresponding validity periods. As represented in the figure, version  $m_{e1}(t_1)$  is valid in interval  $[t_1, t_2]$ . Since there is no version of model element  $m_{e1}$ , which is captured later than  $t_2$ , the validity of version  $m_{e1}(t_2)$  is the open interval  $[t_2, +\infty[$ . Accordingly, version  $m_{e2}(t_3)$  is valid in  $[t_3, +\infty[$ . Since model elements now have a temporal validity, a relationship  $r$  from a model element  $m_{e1}$  to  $m_{e2}$  is no longer uniquely defined. Instead, the timestamps of model elements have to be taken into account for the resolution of relationships (described in III-C). The association of each model element with a timestamp and thus

a validity, provides the foundation for a continuous context representation. Although model elements are still sampled at discrete timestamps, the definition of a continuous validity for each model element allows to represent a context as a continuous structure. This structure provides the foundation for our time-distorted context models.

#### B. Navigating through Time

Based on the idea that it is necessary for intelligent systems to consider not only the current context but also historical data to correlate or investigate potential causalities between two phenomena, we provide means to enable an efficient navigation into time. Therefore, we define three basic operations for model elements. These can be called on each model element: The *shift* operation is the main possibility to navigate a model element through time. It takes a timestamp as parameter, looks for the model version of itself, which is valid at the required timestamp, loads the corresponding element from storage (can be RAM) and returns the loaded version.

The *previous* operation is a shortcut to retrieve the direct predecessor (in terms of time) of the current model element.

The *next* method is similar to the *previous* operation but retrieves the direct successor of the current model element.

These operations allow us to shift model elements independently from each other through time. This makes it possible to create context models, combining model elements from different points in time. Now only the last step is missing to have time-distorted context models for efficient runtime reasoning processes: A concept to take the time of model elements into account when navigating between model elements.

#### C. Time-relative Navigation

Navigating temporal data is a complex task, since a relationship  $r$  from an element  $m_{e1}$  to an element  $m_{e2}$  is no longer uniquely defined. Instead—depending on the timestamps  $t_1$  and  $t_2$  of  $m_{e1}$  and  $m_{e2}$ , and depending on the set of versions of  $m_{e1}$  and  $m_{e2}$ —a relationship  $r$  from  $m_{e1}$  to  $m_{e2}$  can link different versions of  $m_{e2}$ . This means which version of  $m_{e2}$  is related to  $m_{e1}$  by  $r$  depends on the timestamp  $t$  of  $m_{e1}$ . Processing this time-relative navigation manually, e.g. to correlate or investigate potential causalities between two phenomena, is complicated and error-prone. We therefore provide a concept to automatically resolve relationships, taking the time aspect into account, while navigating the context model. This time related resolution is completely transparent and hidden behind methods to navigate in the context model. Hereby, a context time can be defined (the curve in fig. 2) and each model element is then resolved accordingly to this definition while traversing the model. For example, the context time can be defined as the current time of a model element minus one day. When navigating from model element  $m_{e1}$  at timestamp  $t_i$  to element  $m_{e2}$ , the version of  $m_{e2}$ , which is valid at timestamp  $t_i - 1 \text{ day}$  is resolved. In case that at timestamp  $t_i$ , object  $m_{e2}$  does not exist, the *prior existing version* of  $m_{e2}$  is returned. Considering model elements in the context of a specific time interval creates a navigable time dimension for model elements. This time relative data resolution is one of the novel concepts of this contribution. Unlike in previous approaches (e.g. relationships in MOF [19] or predicates in RDF [12]), the navigation function is not constant but yields

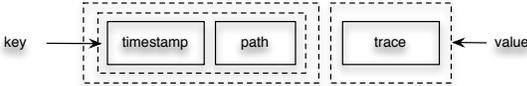


Fig. 4. Key/value structure for time-relative storage

different results depending on the navigation context (*i.e.* the current observation date). This distortion in terms of navigable relations finally enables what we call a time-distorted context.

#### IV. INTEGRATION INTO KMF

In this section we describe how we integrate our time-distorted modeling approach into the open source modeling framework KMF. We rely on two properties to integrate the time dimension as a crosscutting concern into model elements: *i)* each model element must be uniquely identifiable, and *ii)* it must be possible to get a serialized representation of all attributes and relationships of each model element, with no relativity to a time. To ensure the first property, KMF defines a *path* for each model element, starting from the root element of a model to the element following the containment relationships, as a unique identifier. Since the containment graph is actually a tree (each element, except the root, has to be contained exactly once), the path is a unique full qualified identifier. For the second property, KMF serializes models and model elements into *traces*. A trace defines a sequence of atomic actions to construct a model element, using the path concept (including relationship information). Each model element can be transformed into a trace and vice versa [20], [21]. As stated in III-A we inject a timestamp into all model elements. We do that by extending the KMF generator to automatically generate a timestamp attribute for all model elements. As a consequence, a model element, or more precisely a version of a model element, is now no longer simply defined by its path alone, but by a combination of its path and timestamp. Using the path together with a timestamp as key and the trace as value allows us to store and retrieve model elements within their time dimension in a simple *key/value* manner. The resolution (and storage) of a model element is therefore always relative to the context time. This is shown in fig. 4.

This context data organization allows us to use technologies eligible for big data to efficiently store and load context data. The data can be stored using different back ends, *e.g.* key/value stores, relational databases, or simply in memory (as a cache). In our implementation we use Google LevelDB<sup>2</sup> since it has proven to be suitable for handling big context data and, most importantly, it is very fast for our purpose (see VI). The storage implementation itself, however, is not part of our contribution. We intend to provide an efficient layer for runtime reasoning processes, on top of already existing storage technologies.

Since data captured in context models usually evolve at a very high pace (*e.g.* milliseconds or seconds), and our approach foresees to not only store the current version of model elements but also historical ones, context models can quickly become very large. In such cases, context models may no longer fit completely into memory, or at least it is no longer practical

to do so. Therefore, based on our storage concept and the uniqueness of KMF paths in a model, we implement a *lazy loading*<sup>3</sup> mechanism to enable efficient loading of big context models. We use *proxies*<sup>3</sup>, containing only path and timestamp, to reduce the overall memory usage. Attributes and referenced elements are only loaded when they are read or written. To enable this we extend KMF so that, while the context model is traversed, relationships are dynamically resolved. First, it must be determined which version of a related model element must be retrieved. This depends on the timestamp of the model element version (and the context time) from which the navigation starts (discussed in III-C). Second, the actual model element version must be loaded from storage. Our implementation allows to manage context models of arbitrary sizes efficiently and it hides the complexity of resolving—and navigating—temporal data behind an API.

#### V. API

Modeling approaches use meta-model definitions (*i.e.* concept descriptions) to generate domain specific APIs. The following section illustrates our API on a simplified smart grid meta-model definition. The model consists of a smart meter with an attribute for electric load and a relationship to reachable surrounding meters. In addition to a classical modeling API, our time-distorted context extension provides functions for creating, deleting, storing, loading, and shifting versions of model elements. Applications can use this API to create new model elements, specify their timestamps, store them, change their attributes and relationships, and store new versions (with different timestamps). In addition, the API can be used to specify the *context time* on which elements should be resolved while traversing the model. One can imagine the definition of the context time as the curve shown in fig. 2. Listing 1 shows Java code that uses a *Context ctx* (abstraction to manipulate model elements) to perform these actions.

```
// creating and manipulating model elements
m1 = ctx.createSmartMeter("m1","2014/3/1");
m1.setElectricLoad(125.6);
m1.addReachables(ctx.load("m2"));
m1_2 = m1.shift("2014/3/2");
m1_2.setElectricLoad(193.7);
// definition of the context time
ctx.timeDef("m1","2014/3/1");
ctx.timeDef("m2","2014/3/2");
r_m1 = ctx.load("m1");
assert(r_m1.getElectricLoad()==125.6);
r_m2 = r_m1.getReachables().get(0);
assert(r_m2.getTime()=="2014/3/2")
```

Listing 1. Usage of the time-distorted modeling API

The API provides a seamless way to create, manipulate, and navigate in time-distorted context representations.

#### VI. EVALUATION

To quantify the advantages of our approach, we evaluate it on a smart grid reasoning engine to predict the electric load in a region based on current load and historical data. This problem is taken from our cooperation with Creos Luxembourg S.A. and led to the research behind this approach.

<sup>2</sup><https://code.google.com/p/leveldb/>

<sup>3</sup><http://wiki.eclipse.org/CDO>

TABLE I. BENCHMARK USING FULL SAMPLING

Scenario	Reasoning	Insert
Small Deep Prediction (SDP)	1075.6 ms	267 s
Small Wide Prediction (SWP)	1088.4 ms	
Large Deep Prediction (LDP)	180109.0 ms	
Large Wide Prediction (LWP)	181596.1 ms	

TABLE II. BENCHMARK USING TIME-DISTORTED CONTEXTS

Scenario	Reasoning	Insert
Small Deep Prediction (SDP)	1.8 ms	16 s
Small Wide Prediction (SWP)	0.8 ms	
Large Deep Prediction (LDP)	187.0 ms	
Large Wide Prediction (LWP)	157.6 ms	

**Smart grids** are infrastructures characterized by the introduction of reactive entities modernizing the electricity distribution grid. Smart meters, entities installed at customer sites to continuously measure consumption data and propagate it through network communication links, are one of the main building blocks of smart grids. Based on the electrical consumption smart meters can determine the electrical load. The load is regularly sampled, which leads to big context models. The idea for this reasoning engine is to predict if the load in a certain region will likely exceed or surpass a critical value. Our experimental validation focuses on two key indicators: (1) performance of the reasoning process and (2) insertion time.

**Experimental results:** We implemented the reasoning engine case study leveraging our approach on top of the KMF framework, presented in IV. It has been implemented twice, once with a classical systematic sampling strategy, and once using a time-distorted context model. The full sampling approach is implemented using the same data storage (Google LevelDB) as our time-distorted context representation approach. Our context model definition consists of one concept, a *smart meter*, one attribute for the *electrical load*, and a *reachable* relationship, which connects smart meters. The context model under study contains 100 smart meters with 10000 values history each, resulting in one million elements to store and analyze. This amount of data corresponds to roughly 140 days history. The experimental validation consists of an electrical load prediction for a specific point of the grid. We define two kinds of predictions both based on a linear regression: 1) *deep* uses a deep history (in time) of the meter, 2) *wide* uses history and in addition the electric load of surrounding meters. These two strategies are each executed with two different ranges: 1) *small* using ten hours of history (30 time units), 2) *large* uses a history of two months (4800 time units). Using the two strategies and two ranges for each, we create four different test series (SDP, SWP, LDP, LWP) that represent typical use cases for our case study. The experiments were carried out on a MacBook Pro i5 2.4 Ghz, 16 GB RAM. The results using full sampling are presented in table I, the results leveraging time-distorted contexts in table II. As shown in the tables, our time-distorted context strategy leads to a reduction of the reasoning time by factors of: **598** for *SDP*, **1361** for *SWP*, **963** for *LDP*, and **1152** for *LWP*, compared to the classic full sampling strategy. The insert time (for storing the context values) has also been significantly improved by a factor of **17**. The two reasoning strategies (full sampling and time-distorted models) predict the same electric load values for

all tests. Our evaluation shows a reasoning time in the order of magnitude of minutes for regular sampling and milliseconds for time-distorted contexts, to analyze a particular section of the grid. Moreover, according to our smart grid case study, this electric load prediction has to be continuously performed on several hundred grid points. Thus, our solution reduces the computation time from hours to a few seconds, compatible with the near real-time requirements of smart grid overload capacity. The huge gains compared to full sampling can be explained by the fact that time-distorted contexts allow to directly navigate across the time dimension without costly mining the necessary data from different context models.

## VII. DISCUSSION AND RELATED WORK

The lack of a temporal dimension in data modeling has been discussed in detail, especially in the area of databases. In an early work Clifford *et al.* [22] provide a formal semantic for historical databases. Rose and Segev [23] suggest to extend the entity-relationship data model into a temporal, object-oriented one, incorporating temporal structures and constraints in the data model itself rather than at the application level. They also propose a temporal query language for the model. Ariav [24] suggests a temporally-oriented data model as a restricted superset of the relational model. He adds a temporal aspect to the tabular notion of data and provides a framework and a SQL-like query language for storing and retrieving data, taking their temporal context into account. Works of Mahmood *et al.* [25] and Segev and Shoshani [26] take a similar direction and seek to extend the relational model with temporal aspects. In an earlier work Segev and Shoshani [27] examine semantics of temporal data and corresponding operators independently from a data model. In a more recent work, Shih *et al.* [28] describe how including time in a context model helps triggering and handling exceptions in system processes. Selig *et al.* [29] describe an interesting mathematical approach to examine time-dependent association between variables. In the Bigtable [30] implementation, Google incorporates time at its core by allowing each cell in a Bigtable to contain multiple versions of the same data, associated to different timestamps. All this work addresses mainly efficient storage and querying of time related data but largely ignores handling of time in the application domain itself. However, considering time in reasoning processes, like correlating causalities between phenomena, is complex and time-consuming, conflicting with the strict response time requirements intelligent systems usually face. The need to represent and reason about temporal knowledge has also been discussed in RDF [12], OWL [13] and the Semantic Web. Motik [31] suggests a logic-based approach for representing validity time in RDF and OWL. He also proposes to extend SPARQL to temporal RDF graphs and presents a query evaluation algorithm. We suggest to add a time dimension directly into the knowledge representation of a domain itself, *i.e.* into the core of context models. Therefore, we not only efficiently store and query historical data (what is done in other works before), but we propose a way to use time-distorted data sets specifically for intelligent reasoning. Also, we do not extend a specific data model (*e.g.* the relational data model) with temporal structures but use model-driven engineering techniques to integrate a time dimension as crosscutting property of any model element. We do not rely on a complex query language for retrieving temporal data. Instead, our approach aims at providing a natural, query-less

and seamless navigation into the time dimension of model elements, allowing a composition of different time-related values to build a dedicated context model for reasoning purposes (inspired by temporal logic [32]). Like version control systems, e.g. Git<sup>4</sup>, our approach stores incremental changes (over time) rather than snapshots of a complete system.

Our approach is especially useful if model elements evolve at different paces. If all elements of a context evolve at the same pace the main advantage is the navigation concept as well as the lazy loading mechanism. In future work we want to improve and optimize our implementation. Especially the insertion of new model element versions between two existing versions has to be improved. In addition, we want to investigate how to distribute storage across multiple machines and how this affects the performance of our approach.

## VIII. CONCLUSION

Considering time as a crosscutting concern of data modeling has been discussed since more than two decades. However, recent data modeling approaches mostly still rely on a discrete time representation, which can hardly consider model elements (e.g. context variables) coming from different points in time. In this paper, we presented a novel approach which considers time as a first-class property crosscutting any model element, allowing to organize context representations as time-distorted views dedicated for reasoning processes, rather than a mere stack of snapshots. By introducing a temporal validity independently for each model element we allowed model elements to evolve independently and at different paces, making the full sampling of a context model unnecessary. Instead of introducing a dedicated querying language we provided operations to move model elements independently through time, enabling the creation of context models, which combine model elements from different timestamps. Finally, we added a time-relative navigation, which makes an efficient navigation between model elements, coming from different timestamps, possible. This allows us to assemble a time-distorted context model for a specific reasoning purpose and seamlessly and efficiently navigate along this time distortion without manual and costly mining of the necessary data from different context models. Our approach has been implemented and integrated into the open source modeling framework KMF and evaluated on a smart grid reasoning engine for electric load prediction. We showed that our approach supports reasoning processes, outperforms a full context sampling by far, and is compatible with most of near real-time requirements.

## REFERENCES

- [1] M. Perttunen, J. Riekkki, and O. Lassila, "Context representation and reasoning in pervasive computing: a review," *Int. Journal of Multimedia and Ubiquitous Engineering*, pp. 1–28, 2009.
- [2] C.-H. Liu, K.-L. Chang, J.-Y. Chen, and S.-C. Hung, "Ontology-based context representation and reasoning using owl and swrl," in *Communication Networks and Services Research Conf. (CNSR), 2010 8th Annu.*, 2010, pp. 215–220.
- [3] K. Henriksen, J. Indulska, and A. Rakotonirainy, "Modeling context information in pervasive computing systems," in *Proc. 1st Int. Conf. Pervasive Computing*, ser. Pervasive '02, 2002, pp. 167–180.
- [4] G. Blair, N. Bencomo, and R. France, "Models@ run.time," *Computer*, vol. 42, no. 10, pp. 22–27, 2009.
- [5] B. Morin, O. Barais, J. Jezequel, F. Fleurey, and A. Solberg, "Models@ run.time to support dynamic adaptation," *Computer*, vol. 42, 2009.

- [6] J. C. Cepeda, D. Ramirez, and D. Colome, "Probabilistic-based overload estimation for real-time smart grid vulnerability assessment," in *Transmission and Distribution: Latin America Conf. and Expo. (T D-LA), 2012 6th IEEE/PES*, 2012, pp. 1–8.
- [7] H. W. Tom, G. Aumiller, and C. Brito-Cruz, "Time-resolved study of laser-induced disorder of si surfaces," *Physical review letters*, vol. 60, no. 14, p. 1438, 1988.
- [8] P. Hubral, "Time migration-some ray theoretical aspects\*," *Geophysical Prospecting*, 1977.
- [9] M. Baldauf, S. Dustdar, and F. Rosenberg, "A survey on context-aware systems," *Int. J. Ad Hoc Ubiquitous Comput.*, vol. 2, no. 4, 2007.
- [10] T. Strang and C. L. Popien, "A context modeling survey," in *UbiComp 1st Int. Workshop on Advanced Context Modelling, Reasoning and Management*, 2004, pp. 31–41.
- [11] P. P. shan Chen, "The entity-relationship model: Toward a unified view of data," *ACM Trans. Database Syst.*, vol. 1, pp. 9–36, 1976.
- [12] O. Lassila and R. R. Swick, "Resource Description Framework (RDF) Model and Syntax Specification," W3C, W3C Recommendation, 1999.
- [13] W. W. W. C. W3C, *OWL 2 Web Ontology Language. Structural Specification and Functional-Style Syntax*, Std., 2009.
- [14] S. Kent, "Model driven engineering," in *IFM*, 2002.
- [15] J. Rothenberg, L. E. Widman, K. A. Loparo, and N. R. Nielsen, "The nature of modeling," in *Artificial Intelligence, Simulation and Modeling*, 1989, pp. 75–92.
- [16] F. Fouquet, G. Nain, B. Morin, E. Daubert, O. Barais, N. Plouzeau, and J. Jézéquel, "An eclipse modelling framework alternative to meet the models@runtime requirements," in *MoDELS*, 2012.
- [17] F. Fouquet, E. Daubert, N. Plouzeau, O. Barais, J. Bourcier, and J.-M. Jézéquel, "Dissemination of reconfiguration policies on mesh networks, DAIS 2012."
- [18] F. Budinsky, D. Steinberg, and R. Ellersick, *Eclipse Modeling Framework : A Developer's Guide*, 2003.
- [19] OMG, *OMG Meta Object Facility (MOF) Core Specification, Version 2.4.1*, Object Management Group Std., Rev. 2.4.1, 2011.
- [20] X. Blanc, I. Mounier, A. Mougnot, and T. Mens, "Detecting model inconsistency through operation-based model construction," in *Proc. 30th Int. Conf. Software Engineering*, 2008, pp. 511–520.
- [21] J. Klein, J. Kienzle, B. Morin, and J.-M. Jézéquel, "Aspect model unweaving," in *In 12th International Conference on Model Driven Engineering Languages and Systems (MODELS 2009)*, L. 5795, Ed., Denver, Colorado, USA, 2009, pp. p 514–530.
- [22] J. Clifford and D. S. Warren, "Formal semantics for time in databases," in *XP2 Workshop*, 1981.
- [23] E. Rose and A. Segev, "Toodm - a temporal object-oriented data model with temporal constraints," in *ER*, T. J. Teorey, Ed., 1991.
- [24] G. Ariav, "A temporally oriented data model," *ACM Trans. Database Syst.*, vol. 11, no. 4, pp. 499–527, 1986.
- [25] N. Mahmood, A. Burney, and K. Ahsan, "A logical temporal relational data model," *CoRR*, 2010.
- [26] A. Segev and A. Shoshani, "The representation of a temporal data model in the relational environment," in *SSDBM*, 1988.
- [27] —, "Logical modeling of temporal data," in *Proc. ACM SIGMOD Int. Conf. on Management of Data*, ser. SIGMOD '87, 1987.
- [28] C.-H. Shih, N. Wakabayashi, S. Yamamura, and C.-Y. Chen, "A context model with a time-dependent multi-layer exception handling policy," *IJICIC*, vol. 7, no. 5A, pp. 2225–2234, 2011.
- [29] J. P. Selig, K. J. Preacher, and T. D. Little, "Modeling time-dependent association in longitudinal data: A lag as moderator approach," *Multivariate Behavioral Research*, vol. 47, no. 5, pp. 697–716, 2012.
- [30] F. Chang, J. Dean, S. Ghemawat, W.-C. Hsieh, D.-A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R.-E. Gruber, "Bigtable: A distributed storage system for structured data," in *Proc. 7th USENIX Symp. OSDI - Volume 7*, ser. OSDI '06, 2006, pp. 15–15.
- [31] B. Motik, "Representing and querying validity time in rdf and owl: A logic-based approach," in *Proc. 9th Int. Semantic Web Conf. The Semantic Web - Volume Part I*, ser. ISWC'10, 2010, pp. 550–565.
- [32] A. Pnueli, "The temporal logic of programs," in *Foundations of Computer Science, 1977., 18th Annu. Symp.*, 1977, pp. 46–57.

<sup>4</sup><http://git-scm.com/>

# Generating Real-Time Profiles of Runtime Energy Consumption for Java Applications

Muhammad Nassar, Julian Jarrett, Iman Saleh, M. Brian Blake

Department of Computer Science

University of Miami

Coral Gables, Florida, USA

m.mansour1@umiami.edu

{j.jarrett, iman, m.brian.blake}@miami.edu

**Abstract**— Energy consumption of computer-based systems is a growing concern, especially for large scaled distributed systems that operate in data centers and server farms. Currently, there exist many hardware implementations that holistically measure the consumption of energy for hardware and software systems. These approaches are limited as they measure the current power consumption of the overarching system - an approach that does not necessarily assist in making the underlying software more power-aware while under operation. This paper introduces the algorithm and process of calculating the energy consumption of Java applications at runtime. This approach is evaluated using a new energy profiler tool that is designed to integrate with a Java application leveraging the underlying physical architecture of the hardware. By incorporating the awareness of energy consumption within the software application, then that application can dynamically choose the most energy-efficient processing paths within its operations.

**Keywords** - power consumption; java; energy; distributed systems; software profiling

## I. INTRODUCTION

Energy consumption is a significant global challenge. Data centers and servers in the United States consume huge amounts of energy and consequently challenge the sustainability of our natural resources [8,9,10]. The use of enterprise service-oriented systems has only exacerbated the challenges [7,16]. Furthermore, on an individual basis with the prevalence of mobile devices, it is important to optimize software operations in order to extend battery capabilities.

- *What attributes are required and which are most effective in calculating energy consumption in a runtime environment of a Java application?*
- *What are the challenges for disambiguating residual power consumption from the hardware platform with power consumed by the targeted software application?*
- *Are there approaches for reducing the amount of residual power calculated within the software application-specific calculation?*

To address these research questions, we introduce a unique method for profiling applications in terms of their runtime power consumption. To support this method, we conceptualized and developed a novel *Energy Profiler* for Java applications. The profiler integrates with Java applications in such a way that provides an energy consumption estimate for the software in question. The profiler leverages the energy model described in [1], where the authors estimate the power consumption of a computational model with N number of processors, a separate cache memory for each processor and main memory. This approach enables software developers to use energy efficiency as a key performance metric influencing their software design decisions. Moreover, software engineers will be able to abstract design practices and patterns that decrease the energy footprint of applications. We evaluate this model by comparing estimated values produced from the energy profiler with measures produce from a physical power consumption meter. Using varying programming constructs to implement the same function, we show a significant degree of accuracy with our model and implementation when comparing energy consumption trends to processing intensity.

## II. RELATED WORK

Many frameworks have been proposed to profile power and energy consumption across different types of computer systems. These frameworks utilize both hardware and software techniques for profiling energy and target both servers and mobile devices. In the early work of one of the co-authors of this paper, a method was proposed that leveraged design-time testing of web services using generic loads of web traffic. The energy was measured during a training phase and was used to develop a model for a specific web service at various server and software states [5]. The work described in [5] and [6] uses a more predictive model applied towards the efficient power management of web services and redundantly deployed cloud environments.

Some of the previous efforts in this area evaluate the overall system energy usage [11,12], while others have the capability of applying more granular measurement techniques to isolate and profile individual components. The authors in [2] introduce the PowerPak Framework; a combination of hardware and software components capable of low-level

energy profiling of parallel scientific applications on multi-core, multi-processor distributed systems. They use physical instrumentation including sensors, meter circuits and data acquisition devices for measurement. Software components consist of drivers for physical meters, sensors, and a user level API for automated power profiling and synchronization of code. The framework is also capable of producing direct and derived measurements and isolating individual component consumption through the use of fine-grained systematic power measurements. PowerScope is a tool proposed by the authors in [3] to profile the energy usage of mobile applications through statistical sampling. A digital multi-meter is used to sample both the power consumption and system activity and passes this information to a data collection computer running the energy monitor. An energy profile is then generated and the data is analyzed offline to remove profiling overhead.

In [4], researchers at Microsoft present a Windows based fine-grained energy profiler by implementing and integrating a workload manager, an event logger and an energy profiler. Event tracing is started by the workload manager, which executes code under fixed loads of data. Tracing includes kernel level tracing of components such as the CPU, Disk I/O, page faults, heap range creation and context switches. The event logger simply logs all events generated by the workload manager and the energy profiler is then used to correlate system resource usage of an application.

Some other efforts on the hardware level make use of Dynamic Voltage and Frequency Scaling (DVFS). DVFS is a widely used technique for power manipulation [13,15]. Its basic idea is to decrease the CPU frequency to allow a corresponding change in the supplied voltage. This in turn results in a reduction of the overall power consumption. DVFS is being used mainly to control the power consumed by the CPU cores rather than main memory. Despite this fact, the writers of [14] successfully developed a memory frequency-scaling algorithm that could reduce the power consumption of main memory by around 14% on average without any noticeable performance degradation.

Unlike the previous approaches, our tool does not require the use of external instrumentation for measurement. Our approach for application profiling utilizes a plug-in that operates in real-time and produces an estimate of the total energy consumed by the application. The plug-in is provided as a JAR file that can be imported into Java applications. This approach extends the state of the art as it allows application-level estimations. The profiler allows the applications to be more self-aware of their own energy signatures. We use a mathematical model that estimates the energy usage given a set of parameters describing the underlying hardware. We validate our model's accuracy by comparing the estimated values to the ones measured by an energy meter.

### III. ENERGY MODEL

According to our model, the energy consumed by a software system has 3 main components:

- Computational Energy:  $EPI * T_{CPU} * X$

- Memory access energy:  $E_m * T_{Active} * (The\ amount\ of\ memory\ used\ in\ bytes)$
- Leakage Energy:  $E_l * T_{Active} * X$

where,  $EPI$  is the energy consumed per CPU instruction (in the instruction set of the CPU),  $T_{CPU}$  is the total time the CPU is active,  $E_m$  is the energy consumed for a single memory access,  $E_l$  is some hardware constant,  $T_{Active}$  is the total time the system is active (including CPU idle time) and  $X$  is the CPU clock frequency.

This energy model is loosely based on the one described in [1], and relies on the values obtained from the Java Virtual Machine (JVM). The computational energy component is computed as the product of the CPU frequency, the total time the CPU is active – where these two terms provide the total number of instructions executed – and the energy consumed per instruction, which is a chipset constant. The JVM provides the total time the CPU is active, while the user is responsible for providing the CPU frequency, and the energy per instruction value, if available. Some assumptions are made while calculating the computational energy component. The system used for testing has an Intel Core 2 mobile chipset and it is assumed that the CPU performed one instruction per cycle. It is also assumed that the microprocessor is the sole energy consumer and that other parts of the chipset – like the north/southbridge – is not taken into account since the energy model in [1] does not incorporate them. The Graphics Processing Unit (GPU) is not incorporated in the model either, and is not taken into account. However, the tests presented in later sections do not call for any GPU usage.

As for the memory access energy, since there is no direct way of computing the exact number of memory accesses, and since the JVM provides the total amount of memory it uses directly, an approximated calculation is used as described above. The total memory access energy is calculated as the product of the amount of memory used by the JVM, the total time the application is active – where these two terms provide the total amount of memory that needs to be sustained for the lifetime of the application – and the energy consumed per memory access, which is also a chipset constant. The JVM provides the total amount of memory used, and the total time the application is active is calculated as the difference between the end and start times of the application obtained directly from the JVM. The user provides the energy per memory access value, if available.

The leakage energy component is calculated in the exact same manner as described in [1], which is the product of the CPU frequency, the total time the application is active, and  $E_l$ , which is a hardware constant value provided by the user if available.

### IV. DESIGN AND ARCHITECTURE

An energy profiler plug-in that implements the energy model described in Section 3 was developed in Java. It is available as a JAR file which can be easily integrated into other Java applications. As shown in Figure 1, the plug-in gathers the data it needs from two sources, user input about chipset

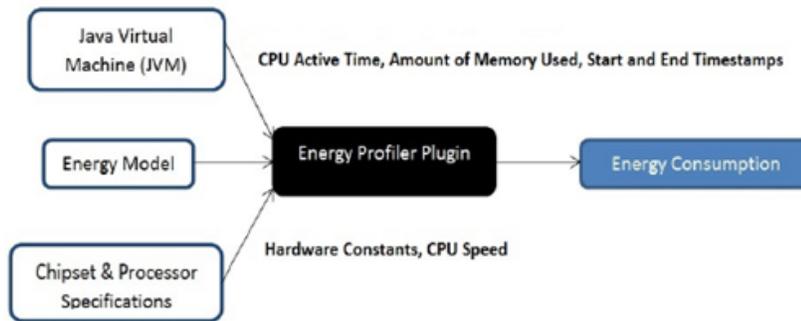


Figure 1. Energy Profiler High-level Approach

specific constants and CPU and memory performance information from the JVM. The plug-in then applies this data to the energy model to produce the energy consumption information.

As shown in the class diagram described in Figure 2, the application is split into three classes. The *CPUMonitor* class is responsible for fetching CPU utilization information from the JVM. The *MemoryMonitor* class gathers memory utilization information from the JVM. The *EnergyCalculator* class collects the data retrieved by these two components, plugs them along with the hardware constants into the energy model, and provides a total consumption estimate in Joules. The hardware constant values  $EPI$ ,  $E_m$  and  $E_l$  are hardware-specific and should be provided by the developers. If not provided, the plug-in uses default values specific to the Intel Core 2 Duo processor architecture. In order to use the plug-in to gather energy consumption data, some steps need to be performed. Before the code block whose energy consumption needs to be analyzed, an object of type *EnergyCalculator* is declared and the current CPU frequency in GHz is passed as a parameter to its constructor. If the hardware constants  $EPI$ ,  $E_m$  and  $E_l$  are known, another constructor is provided that accepts them as parameters alongside the CPU frequency. If these constants are not provided, default values are used as described earlier. After the code block that is being examined, the method *PlugDataIntoModel()* is called from the previously defined *EnergyCalculator* object. This method returns a string value that represents the energy consumed by the enclosed code block in Joules.

Listing 1 shows how an *EnergyCalculator* class object can be used to profile the energy usage of Java code. As explained earlier, the *EnergyCalculator* class object has to be initialized before the code block in question and a call to the *PlugDataIntoModel()* method after the code block will provide the energy consumption estimate. Listing 2 shows the pseudo-code of the *EnergyCalculator* class.

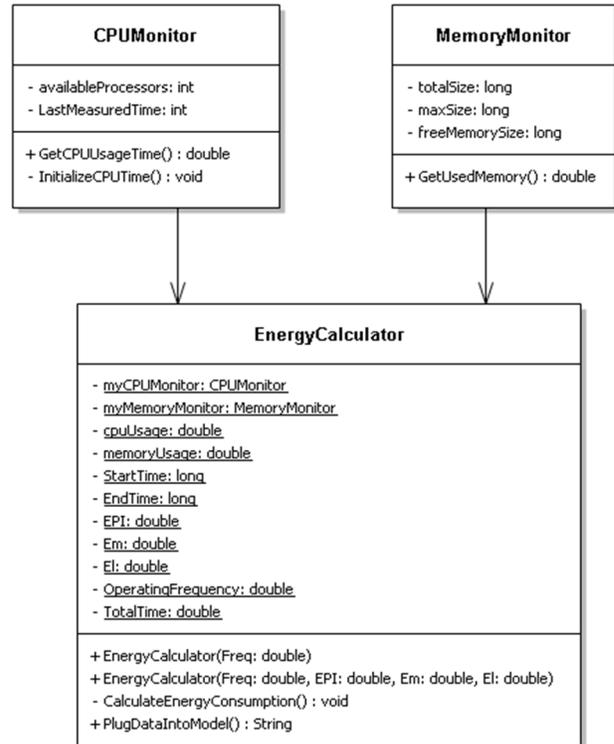


Figure 2. System Class Diagram.

```

EnergyCalculator calc(OperatingFrequency, EPI,
                    Em, El);

<Profiled Java Code>

String Energy_Consumption =
    <calc.PlugDataIntoModel>;

```

Listing 1. Energy Profiler's Usage.

```

public EnergyCalculator (CPUFrequency, EPI, Em, El)
{
    CPUMonitor myCPUMonitor;
    MemoryMonitor myMemoryMonitor;
    double StartTime = <now>;
    double OperatingFrequency = <CPUFrequency>;
    double EPI = <user input or default value
                from Datasheet>;
    double Em = <user input or default value
                from Datasheet>;
    double El = <user input or default value
                from Datasheet>;
}
private void CalculateEnergyConsumption ()
{
    double memoryUsage = <myMemoryMonitor.
                        GetUsedMemory(>;
    double cpuUsage = <myCPUMonitor.
                     GetCPUUsageTime>;

    double EndTime = <now>;
    double TotalTime = <EndTime - StartTime>;
}
public String PlugDataIntoModel()
{
    CalculateEnergyConsumption();
    double E_comp = <EPI * cpuUsage *
                  OperatingFrequency>;
    double E_mem = <Em * TotalTime * memoryUsage *
                  1024>;
    double E_leak = <El * TotalTime *
                  OperatingFrequency>;
    double TotalEnergy = <E_comp + E_mem + E_leak>;
    return TotalEnergy.toString();
}

```

**Listing 2.** Energy Profiler’s Pseudo-code.



**Figure 3.** Picture of the Watts-Up Measurement Devices.

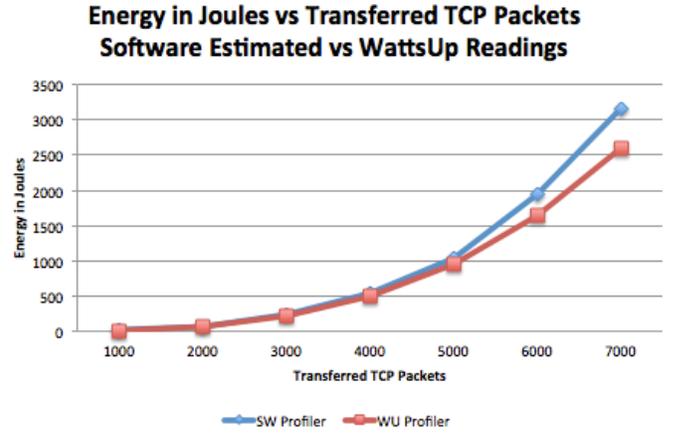
**V. EVALUATION**

The energy profiler can be used to estimate the energy consumed by Java applications as well as compare the consumption of different implementations of the same algorithm. To validate our estimates, we compared energy consumption output from the profiler to energy consumption figures obtained from the WattsUp meter (<https://www.wattsupmeters.com>), a physical device that measures the real time energy consumption of electronic devices in Watts. The WattsUp meter is shown in Figure 3.

Figures 4 and 5 provide sample results illustrating the usage of the profiler. Figure 4 illustrates a comparison between the estimated energy calculated by the energy profiler and the actual readings obtained using the WattsUp meter. The test is conducted on a simple TCP implementation in Java. Figure 5 provides a similar comparison using a recursive implementation of a Fibonacci sequence calculator.

As noted in Figure 4, the estimated and measured energy consumption values increased consistently with the rise in the number of packets sent. The difference between estimated and measured values increased as the load on the system increased. This difference increased from around 10% on lighter system loads to approximately 18% with the highest tested load. We anticipated a slight delta between the estimated and measured energy consumption since the energy profiler only measures the consumption of the application in question, whereas the WattsUp meter measures the entire system’s consumption. Figure 5 shows that the estimated and measured values also increased consistently with the increase in the value of the Fibonacci term. This pattern demonstrates the same behavior as the one seen in Figure 4.

To better understand calculated energy, we decomposed the calculation into the components defined in Section 3. The average relative contributions of computational energy, memory access energy and leakage energy are approximated to 85%, 10% and 5% for the TCP simulation application and 90%, 3% and 7% for the Fibonacci sequence calculator application. The level of contribution of each component is reasonable with respect to the results presented in the related works [17] and [18]. The obtained increase our confidence in the validity of the profiler. Within each experiment, the percentages are consistent as the load increased and the results are also consistent across different applications.



**Figure 4.** Estimated vs. Actual Energy Readings for a Simple TCP Implementation.

Figures 6, 7 and 8, illustrate the results obtained when the profiler is used to measure the energy consumption of both recursive and iterative implementations of a Fibonacci sequence calculator.

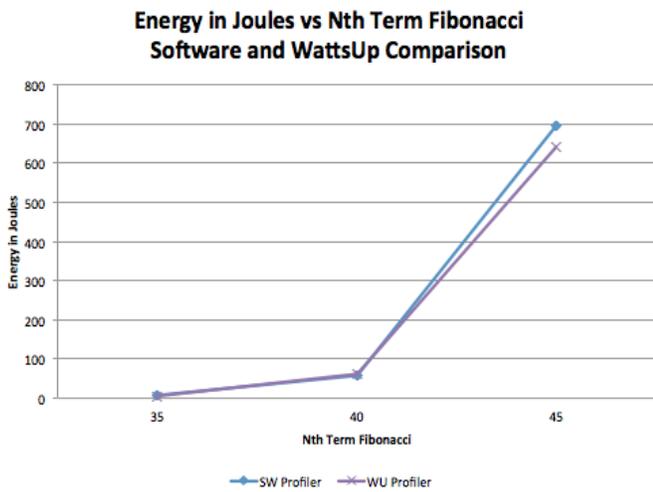


Figure 5. Estimated vs. Actual Energy Readings for a Recursive Fibonacci Implementation.

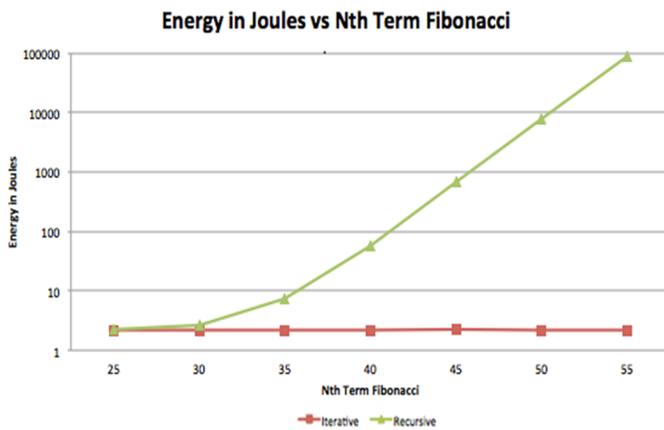


Figure 6. A Comparison between Recursive and Iterative Fibonacci Implementations on Microsoft Windows 7.

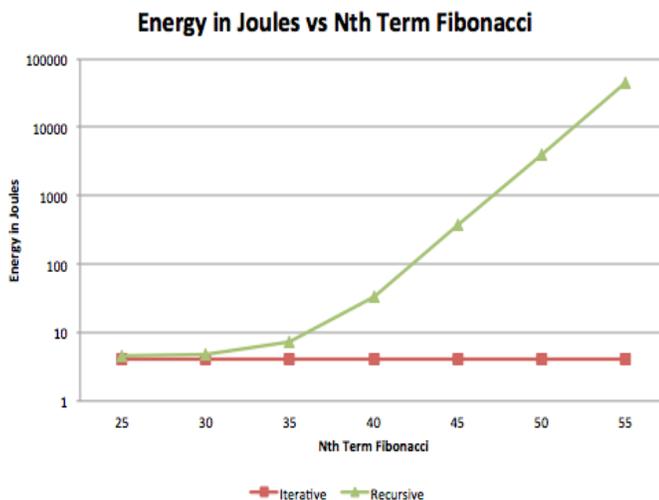


Figure 7. A Comparison between Recursive and Iterative Fibonacci Implementations on Mac OS X 10.8.5.

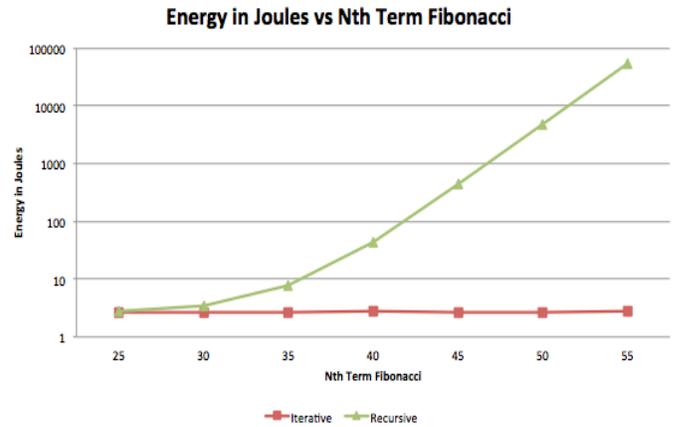


Figure 8. A Comparison between Recursive and Iterative Fibonacci Implementations on Ubuntu Linux 11.10.

The average energy consumption values for each Fibonacci term (from 25 to 55 in increments of 5) are recorded over 12 runs. The tests are repeated across three major operating systems; Microsoft Windows 7, Mac OS X 10.8.5 and Ubuntu Linux 11.10. The results for all three operating systems demonstrated the same pattern. The nature of complexities of the recursive and iterative Fibonacci implementations is evident in the graphs, where the complexity of the iterative implementation grows linearly while that of the recursive implementation grows exponentially. The results obtained demonstrate the expected behaviors of both implementations.

## VI. DISCUSSION

The results of our experiments demonstrate that the energy profiler tool is effective in providing estimates that compare favorably to real measurements. Although many factors are involved when it comes to measuring the energy consumed by different kinds of hardware, the profiler allows for the specification of the hardware constants  $E_{PI}$ ,  $E_m$  and  $E_l$  of the underlying chipset as well as the CPU operating frequency. For the sake of consistency and accuracy of the results, all of the experiments are conducted on the same hardware with no other applications running except the operating system itself.

### A. Estimating Energy Consumption

The first of the two major functionalities of the profiler is to provide absolute estimates of the energy consumed by a certain Java application. Figures 4 and 5 illustrate how those estimates compare to real measurements. It is worth mentioning that in these figures, the energy usage estimates obtained from the profiler are actually higher than those collected by actual measurements. This would not normally be expected since the profiler measures only the energy usage of the Java application under investigation, while the WattsUp meter measures the consumption of the whole system. This could be explained by the fact that no other applications were running while the experiments were being conducted - only the application under investigation and the OS processes - and by the fact that a margin of error in the estimation is expected since many hardware factors and constants are involved in the calculation. Taking these facts and factors into consideration,

the difference between the estimated and actual figures is negligible on lighter levels of system load, however acceptable at the higher load levels.

When it comes to measuring the absolute energy consumption values, the hardware constants described earlier in the energy model play a major role in the correct estimation. The constants that are the hardest to collect are  $EPI$ ,  $E_m$  and  $E_l$ . These constants are chipset dependent and have to be collected from the chipset datasheet. We successfully found the values of  $EPI$  and  $E_m$  for the Intel Core 2 mobile chipset, but we couldn't find an exact value for  $E_l$  and its value is estimated based on information we obtained from the datasheet of a similar chipset to the one we used. The authors in [19] provide the  $EPI$  values for different Intel microprocessors. In its current form, we believe the energy model we use should provide highly accurate results as long as all the variables in the model are as accurate as possible.

### B. Comparing Algorithm Implementations

Figures 6, 7, and 8 demonstrate how the profiler can be effective for another purpose, which is comparing different implementations of the same algorithm from the point of view of energy consumption. This could prove to be very important in real world applications as it provides the capability of comparing the energy consumption of complicated algorithms without having to analyze their complexities thoroughly. It could be considered as a black box technique of analyzing algorithms. The task illustrated (computing the Nth Fibonacci term) provides a good example of such usage. As anticipated, the recursive implementation consumed much more energy than the iterative one, especially as the load on the system increased with the increase of the value of the Fibonacci term. The results are also consistent across different platforms. The marginal error contained in the estimation technique is somewhat irrelevant when comparing different algorithms, since the magnitude of the delta between the estimations is the decisive factor for comparison, rather than the absolute figures.

## VII. CONCLUSION AND FUTURE WORK

This paper presented the experience, method and application of creating an energy profiler tool with the ability to isolate the energy consumption of a specific Java application. Initial experimentation demonstrates that this approach and toolset compare favorably to real measurements. Future work will focus on tweaking the energy model to take into account the internal workings of the JVM, and consequently decreasing the gap between the actual and estimated values. The model can also be expanded to incorporate more hardware component, like the GPU, Video RAM and IO components. We plan to leverage the energy profiler in developing a body of work that describes energy concerns when using specific design patterns for certain types of infrastructures. We believe that this energy profiling approach at the application-level can be used as a training aid for software engineers and developers when it comes to developing sustainable software.

- [1] Agha, G. and Korthikanti, V. 2010. Towards Optimizing Energy Costs of Algorithms for Shared Memory Architectures. *Proceedings of the 22nd ACM symposium on Parallelism in algorithms and architectures*. pp. 157-165.
- [2] Ge, R., Feng, X., Song, S., Chang, H., Li, D., and Cameron, K. W. 2010. "Powerpack: Energy profiling and analysis of high-performance systems and applications." *Parallel and Distributed Systems, IEEE Transactions on* 21, no. 5 (2010). pp. 658-671.
- [3] Flinn, J. and Satyanarayanan, M. 1999. "Powerscope: A tool for profiling the energy usage of mobile applications." In *Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA'99. Second IEEE Workshop on*, pp. 2-10.
- [4] Kansal, A. and Feng, Z. 2008. "Fine-grained energy profiling for power-aware application design." *ACM SIGMETRICS Performance Evaluation Review* 36, no. 2 (2008): pp. 26-31.
- [5] Bartalos, P., Blake, M.B., and Remy, S. 2011. "Green Web Services: Models for Energy-Aware Web Services and Applications" *IEEE International Workshop on Knowledge and Service Technology for Life, Environment, and Sustainability (KASTLES 2011)*, pp. 1-8.
- [6] Bartalos, P. and Blake, M.B. 2012. "Green Web Services: Modeling and Estimating Power Consumption of Web Services", *IEEE International Conference on Web Services (ICWS 2012)*, pp. 178-185.
- [7] Wei, Y. and Blake, M.B. 2010. "Service-Oriented Computing and Cloud Computing: Challenges and Opportunities", *IEEE Internet Computing*, Vol. 14, No. 6, pp. 72-76.
- [8] Talebi, M. and Way, T. 2009. "Methods, metrics and motivation for a green computer science program," *SIGCSE Bull.*, vol. 41, no. 1, pp. 362-366.
- [9] Ghose, A., Hoesch-Klohe, K., Hinsche, L., and Le L. S. 2010. "Green business process management: A research agenda," *Australasian Journal of Information Systems*, vol. 16, no. 2.
- [10] Murugesan, S. 2008. "Harnessing Green IT: Principles and Practices," *IT Professional*, vol. 10, no. 1, pp. 24-33.
- [11] Rivoire, S., Ranganathan, P., and Kozyrakis, C. 2008. "A comparison of high-level full-system power models," in *Proceedings of the 2008 conference on Power aware computing and systems*, 2008, pp. 3-3.
- [12] Li, T. and John, L. K. 2003. "Run-time modeling and estimation of operating system power consumption," *SIGMETRICS Perform. Eval. Rev.*, vol. 31, no. 1, pp. 160-171.
- [13] Le Sueur, E. and Heiser, G. 2010. Dynamic voltage and frequency scaling: the laws of diminishing returns. *Proceedings of the 2010 international conference on Power aware computing and systems, HotPower10*. pp. 1-8.
- [14] David, H., Fallin, C., Gorbatov, E., Hanebutte, Ulf R., and Multu, O. 2011. Memory Power Management via Dynamic Voltage/Frequency Scaling. *ICAC '11 Proceedings of the 8th ACM international conference on Autonomic computing*, pp. 31-40.
- [15] Liang, W., Chen, S., Chang, Y., and Fang, J. 2008. Memory-aware dynamic voltage and frequency prediction for portable devices. In *Embedded and Real-Time Computing Systems and Applications, 2008. RTCSA '08. 14th IEEE International Conference*, pp. 229-236.
- [16] Blake, M.B. and Gomaa, H. 2009. "Agent-Oriented Compositional Approaches to Services-Based Cross-Organizational Workflow", *Special Issue on Web Services and Process Management, Decision Support Systems*, Vol. 40, No. 1, pp 31-50.
- [17] Do, T., Rawshdeh, S. and Shi, W. 2009. "ptop: A process-level power profiling tool." *Proceedings of the 2nd Workshop on Power Aware Computing and Systems (HotPower'09)*.
- [18] Chen, H., Wang, S. and Shi, W. 2011. "Where does the power go in a computer system: Experimental analysis and implications." In *Green Computing Conference and Workshops (IGCC), 2011 International*, pp. 1-6.
- [19] Grochowski, E. and Annavam, M. 2006. "Energy per Instruction Trends in Intel® Microprocessors" *Technology@Intel Mag.*, pp. 1-8

# Towards Sustainability-Oriented Development of Dynamic Reconfigurable Software Systems

Shan Tang, Liping Li, Wenjing Yang, Jianxin Xue  
School of Computer and Information  
Shanghai Second Polytechnic University  
Shanghai, China  
{tangshan, liliping, wjyang, jxxue}@sspu.edu.cn

**Abstract**— Sustainability should be supported by modern software engineering methods to guarantee reliability of the running systems. Dynamic reconfiguration is an important technology implementing the sustainability goal. However, building dynamic reconfigurable software system cost-effectively and in a predictable manner is a major engineering challenge. Aiming at solving this problem, this paper combines AOSD approach and feedback control theory to explore how sustainability goal can be obtained through dynamic reconfiguration. At the same time, a case study is employed to illustrate and evaluate our approach.

**Keywords**- *dynamic reconfiguration; AOP; feedback loop; sustainability; runtime*

## I. INTRODUCTION

Variability is an essential characteristic of software systems. Providing high-quality software in the face of uncertainties (such as dealing with new user needs, changing availability of resources, and faults that are difficult to predict) raises fundamental challenges to software engineers. For making software system can run continuously and can continuously provide service, the system should dynamically adapt its behavior at run-time in response to changing requirements of users and running environment. This is a fundamental requirement that many modern systems must satisfy [1, 20].

Dynamic reconfiguration is a key technology guaranteeing the sustainability of the systems. In recent years, dynamic reconfigurable software systems have been researched and applied in different application domain, and have made a lot of achievements. However, building dynamic reconfigurable software system cost-effectively and there are numerous remaining research problems in developing this kind of systems.

An obvious research problem is that current works address dynamic reconfiguration often loses sight of the existence of system crosscutting concerns (e.g. the quality attribute), causing the system hard to maintain and evolve. Aspect-oriented Programming (AOP) can ease this kind of problem. With the application of AOP [2], the crosscutting concerns can be cleanly encapsulated into aspects and modeled as components, and improve the modularity of software.

Another research problem is: how can we effectively design software so that it can respond to changes in execution environments, without human intervention? Over the past century, feedback control loop has been proposed as an effective model for controlling dynamic behavior of mechanical, electrical, fluid and chemical systems in the corresponding fields of engineering. Inspired by these works, we bring it into the development of dynamic reconfigurable software systems.

In this paper, we take the advantages of AOP technology and feedback control theory to explore how sustainability goal can be obtained through dynamic reconfiguration. In our approach, AOP and feedback loop are regarded as first-class entities. A simplified on-line train ticket reservation system is employed to illustrate our approach throughout this paper.

The remainder of this paper is organized as follows: Section II provides some background knowledge. Section III presents how to implement system's sustainability goal from the perspective of dynamic reconfiguration by applying Aspect-oriented Software Development (AOSD) approach and feedback control theory. Section IV introduces the implementation work and evaluates our approach. And some related works are discussed in section V. Finally, conclusion and future work are presented in section VI.

## II. BACKGROUND KNOWLEDGE

Our research work is founded on MAPE-K control loop, AOSD approach and feedback control theory.

### A. MAPE-K Control Loop

Dynamic reconfiguration systems modify themselves at runtime in order to control the satisfaction of their requirements under changing environmental conditions. Therefore, they are required to monitor themselves and their context's environment, detect significant changes, decide how to react, and act to execute such decisions. All of these behaviors are typically realized using a control loop, which is by means of four components that are responsible for the primary functions of dynamic reconfiguration: Monitor, Analyze, Plan, and Execute, often referred to as the MAPE loop [3].

Figure 1 exemplifies an autonomic controller based upon the MAPE-K loop (monitor, analyze, plan, and execute), whereby data collected by probes is utilized to produce a model of the system and its environment. Through the analysis of such models, we are able to detect system’s state and its environmental conditions, and diagnose performance problems or to detect failures, deciding on how to resolve the problem (e.g., via dynamic load-balancing or healing), and acting to effect the planning decisions made [4].

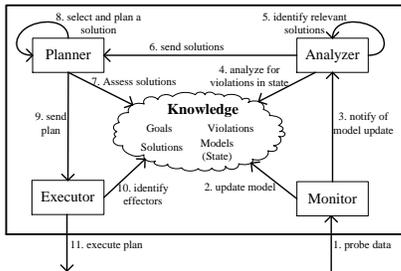


Figure 1. The MAPE-K control loop

### B. AOSD Approach

“Join point” and “aspect” are very important concepts to modularize crosscutting concerns in AOSD. A join point is a well-defined point in the code at which our concerns crosscut the application. Typically, there are many join points for each concern. An aspect specifies a set of join points in the target application where the normal execution is altered and describes the additional behavior that should be executed. The aspect logic is weaved into the target application by aspect weavers.

In existing AOSD approaches, there are different alternatives to tackle the separation of concerns issue, even though the final goal is always the same. These approaches differ mainly with regard to when aspects are applied (before, after or around), the join points where aspects are applied, where the weaving information is placed, whether the weaving process is static or dynamic [5].

Recent years, there are many dynamic AOP techniques being developed. Dynamic AOP (e.g. Spring AOP [6], AspectWerkz [7]) extends static AOP by providing dynamic weaving mechanism, without stopping the whole system. And our implementation is based on Spring AOP. A complete description of AOSD technologies can be found in [8].

### C. Feedback Control Loop

The feedback control loop (closed loop) is the cornerstone structure for building controllers for different kinds of dynamic software systems [9]. Controllers are used to maintain specified properties of the outputs of the target system at (sufficiently near) the given reference values called the set points.

Figure 2 shows a classic feedback loop structure for a target system. This feedback loop performs dynamic control by comparing the target system output to the control objective given as desired state (the set points), yielding the control error (delta), and then adjusting the controlling input based on the output from a sensor, to make the target system to behave closer to the set points.

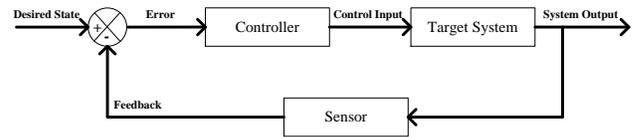


Figure 2. The classic feedback control loop structure

## III. THE APPROACH

In this section, we discuss how to implement system’s sustainability goal from the perspective of dynamic reconfiguration, including how to heal system and how to optimize system’s performance under changing operating environment.

We employ an on-line Train Tickets Reservation System (TTRS) to illustrate our approach. TTRS is a kind of application that must be continuously available for online users, especially during the access peak, like Chinese Spring Festival travel period.

### A. AOP-based Dynamic Reconfiguration

In this section, we describe how to implement dynamic reconfiguration by using AOP technology. Specifically, we will exemplify how system modifies its own structure or behavior to response to the changing internal states and the external environment dynamically.

At first, let us illustrate how to implement dynamic reconfiguration to response to the changing system internal states. Consider such a scenario: there are two message notification components DisImg and DisTxt. DisImg is employed to display vivid graphical information to online-users when the load on the server is normal. DisTxt is used to display pure textual information when the server is highly loaded. Under normal condition, the system instantiates and loads DisImg to provide better service. However when the system comes under heavy load, e.g. during peak times, we assume that the system will instantiate and load DisTxt for improving the request-response time. And if the system load drops to a normal state again, the system needs to adjust itself back to graphical mode dynamically. From this scenario, the system should be dynamic reconfigurable to ensure it can switch back and forth from textual mode to graphical mode automatically. That is to say, the system should load and unload DisImg and DisTxt dynamically for balancing server utilization with service response time.

Figure 3 shows the strategies for dynamic reconfiguration. DisImg and DisTxt are loaded/unloaded dynamically according to different strategy.

```

Strategy SwitchToGraphicalMode() {
  condition{
    sys.Load < concurrAcce_Threshold
  }
  action{
    for(BComp: rBusinessComps){
      BComp.setGraphicalMode(true);
    }
  }
  effect{
    for(BComp: rBusinessComps){
      BComp.isGraphicalMode();
    }
  }
}

Strategy SwitchToTextualModel() {
  condition{
    sys.Load >= concurrAcce_Threshold
  }
  action{
    for(BComp: rBusinessComps){
      BComp.setTextualMode(true);
    }
  }
  effect{
    responseTime < High_T
    for(BComp: rBusinessComps){
      BComp.isTextualModel();
    }
  }
}

```

Figure 3. The strategies for dynamic reconfiguration

Notice that these two message notification components affect some business functions, e.g. register/login an account, pay for orders, and reserve tickets. Therefore, both of DisImg and DisTxt have relationships with other basic components of the system. This means that they crosscut the others components. For making the system easier to maintain and evolve, we employ AOP technology to solve this problem. Concretely, system crosscutting concerns are encapsulated into aspect beans and implemented as aspects components. An aspect bean is an extension of the Java Bean component that specifies crosscutting behavior in a reusable manner.

Figure 4 shows part of the XML-based configuration file for graphical display mode. It's very efficient to implement the dynamic reconfiguration scenario we mentioned above, just by modifying (the values of "bean id" and "ref") and reloading the corresponding configuration file at runtime.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

  <bean id="showGraphicalMessage"
    class="org.springframework.aop.support.NameMatchMethodPointcutAdvisor">
    <property name="advice" ref="DisImg" />
    <property name="mappedNames">
      <list>
        <value>UserRegister</value>
        <value>OrderPayment</value>
        <value>TicketReservation</value>
      </list>
    </property>
  </bean>
```

Figure 4. The XML-based configuration file for graphical display mode

Then, we describe how to implement dynamic reconfiguration to cope with the problems caused by external environment. Suppose that there are various e-bank online payment service components provided by different banks (e.g. BOC, ICBC, ABC, CCB, etc.). Usually, online-users would like to choose a specific e-bank online payment component (e.g. ICBC) to pay for their order.

As all we know that banks server often need to upgrade their service. If one bank server need to be upgraded on someday, and cannot provide payment service to their customers during the upgrade. It is unacceptable for the online users that they couldn't finish their payment after they submit their ticket order due to the service unavailable caused by upgrade, particularly in the case of emergency. In this situation in order to ensure the TTRS application system can continue to provide payment services for all the users, we need to improve sustainability by reconfiguring the target system at runtime to adapt to the environmental failure.

As an online third-party payment platform, AliPay can deal with payment transactions for more than one hundred banks and financial institutions. Assume that AliPay also supports this bank which is in the process of being upgraded. Therefore, the service requests to the upgrading bank will not be refused if we can redirect all the requests to AliPay dynamically. In this scenario, this kind of redirection crosscuts all the e-bank online payment components that AliPay supports. So, we can encapsulate it into a redirection aspect component.

In our work, we model the system in terms of components and connectors for supporting dynamic reconfiguration. A component is an encapsulation of a computational unit. And it specifies its provided services and required services by its provide interfaces and require interfaces. Connectors mediate interactions among components by establishing the rules governing component interaction and specifying any auxiliary mechanisms required [10].

Figure 5 shows the interactions among e-bank online payment components, redirection aspect, and AliPay server. In this interaction diagram, each e-bank online payment component uses services provided by AliPay, and calls the services through connectors, which forward the request to AliPay via the redirection aspect and pass the result to e-bank online payment components. During the process of updating the e-bank server, all the requests from its online-user are buffered and redirected to AliPay to handle sequentially. When the update is finished, all the buffered requests will be redirected back to this e-bank server if these requests have not been handled yet.

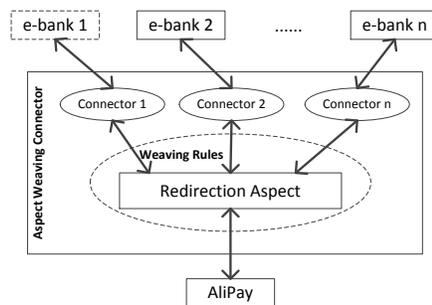


Figure 5. The interactions among e-banks, redirection aspect, and AliPay server

### B. Feedback Loop based Dynamic Reconfiguration

As the previously mentioned scenario, the value of the concurrent accesses threshold (concurrAcce\_Threshold) is a key factor to decide when and how to reconfigure the target system at runtime. In most of the existing approaches, this value is set statically at design time that is very difficult to guide the system to reach an optimal state.

If the value is estimated too high, it will cause the system overloads or even collapses. While if the value is estimated too low, it will lead to a waste of system resources. Therefore, we need to set this value dynamically according to the runtime environment. Here we employ feedback control loop to help us to implement that the system can run reliably and the system resources can be used efficiently.

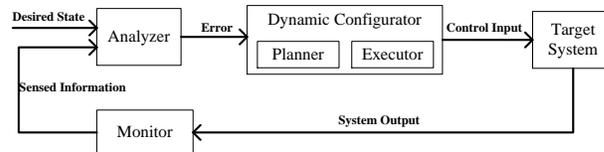


Figure 6. The feedback control loop with architectural components of the MAPE-K loop

Figure 6 shows the feedback control loop with architectural components of the MAPE-K loop. This diagram establishes the

mapping relationship of functional elements between feedback loop and MAPE-K loop.

Monitor is responsible for sensing information that is relevant to dynamic reconfiguration, including the target system's internal variables corresponding to quality attribute, and also the external context. In our work, we use probes to collect data from the executing system and its context about its current state.

Analyzer based on the high-level system requirement and the sensed information which is transferred from monitor, compares target system performance and desired state, determines whether a reconfiguration must be triggered, and infers the control error (delta) and forwards this error to Dynamic Configurator.

Planner selects strategies and generates the reconfiguration plan according to the error information. In this case, the plan refers to how to set the value of `concurrAcce_Threshold`. Meanwhile, planner computes the necessary control actions to be instrumented into the target system.

Executor interprets and executes each of the operations specified in the plan and instruments them into the target system. Thereby adapts target system to obtain the desired state.

#### IV. IMPLEMENTATION AND EVALUATION

We have designed our experimental study to evaluate our work. Our experiment environment is Apache Tomcat 8.0.5 running on a server with Intel(R) Core(TM) i7 CPU and 6G memory.

We implement our AOP-based dynamic reconfiguration approach based on Spring framework. And we adopt the `Interface ServletContextListener` and the `Class TimerTask` to implement the feedback control loop. We use `JMeter` to perform stress test and simulate the runtime environment. During the test, a lot of business operations are recorded in the Tomcat's configuration file (`server.xml`). We analyze the recorded data and produce more runtime state information, like the response time, the total number of requests, and the system's throughput, etc.

In our case study, we compared the performance of the original system which does not have the dynamic reconfigurable ability and the system which has been instrumented with our dynamic reconfiguration approach based on the original system. From the analysis data, we found that the response time of the dynamic system is a bit slower than the original one when the concurrent access less than 200. But the dynamic system has an obvious better performance than the original system when the concurrent access exceeds 500. It shows a significant improvement for highly concurrent application system.

#### V. RELATED WORK

This work has been inspired by many other approaches. For instance, [11] presented a component and aspect model that combines CBSD and AOSD disciplines. This model introduces aspects as special connectors between components. It achieves a high degree of independence between components and aspects, which makes them more reusable. Therefore, based on

this model, developers can build complete application systems in a short time.

In [12] Eddy Truyen and Wouter Joosen demonstrated where and how AOP can be applied in the architecture of self-adaptive systems. In [13], Tesanovic et al. proposed a novel concept of aspectual component-based real-time system development (ACCORD) and applied it successfully in the development of a real-time database system.

In [14], Shen JR et al. demonstrated how to use the model to guide the implementation of dynamic update in a J2EE application server. In our approach, we borrow the redirection idea from their Dynamic Update Connector (DUC) to implement self-healing dynamically.

D.Sykes et al. proposed a dynamic architectural reconfiguration approach in [15] where self-managed software architecture is one in which components automatically configure their interaction in a way that is compatible with an overall architectural specification and achieves the goals of the system.

D.Menasce et al. [16-17] proposed a model-driven framework (SASSY) for runtime self-architecting of distributed service-oriented software systems. This framework automatically generates candidate software architectures and selects the one that best serves stakeholder-defined, scenario-based quality-of-service (QoS) goals. Self-architecting occurs during initial system deployment and at runtime, thus making systems self-adaptive, self-healing, self-managing, and self-optimizing.

In [18], Yuriy Brun et al. explored feedback loops from the perspective of control engineering and within existing self-adaptive systems in nature and biology. At first, they investigated feedback loops as a key aspect of engineering self-adaptive systems. Then they outlined basic principles of feedback loops and demonstrated their importance and potential benefits for understanding self-adaptive systems. Finally, they described control engineering and biologically inspired approaches for self-adaptation.

In [1], A.Filieri et al. explored how continuous reconfigurations through dynamic binding can be obtained as the solution of a discrete-time feedback control problem. They regarded software system as a broken down into constituent blocks having no hard-wired connectors. Connector is created dynamically when the binding between two blocks is established. They applied control theory to automatically derive how the bindings must evolve over time.

[19] applied AI planning and implemented a general closed control loop for selecting a configuration and deciding how to change the system at runtime. Their approach worked well for small examples, but they encountered a critical complexity limit, depending on the number of used "objects" (resources) and the number of "exists" operators.

Danny Weyns et al.[20] investigated the question whether external feedback loops provide more effective engineering solutions than internal mechanisms. And they claimed that external MAPE loops do simplify the design in terms of control flow primitives for the processes. The significant reduction of

control flow complexity increases understandability of the design, and can improve maintainability and testability of the system. The use of external MAPE loops reduces fault density. Reduced fault density increases the quality and reliability of the software design, and adds to customer satisfaction. And external MAPE loops realize a separation of concerns, which yields easier to understand designs, having a positive effect on productivity.

## VI. CONCLUSION AND FUTURE WORK

In this paper, we presented an approach which combines AOSD technology and feedback control theory to implement a dynamic reconfiguration software system to achieve system's sustainability goal.

We have discussed how to implement dynamic reconfiguration under different application scenarios by using AOP and feedback control theory respectively. The main advantages include that: Firstly, our work complies with the "separation of concerns" principle. We separate the dynamic reconfiguration concern from the business function code by using AOP. It makes the system easier to maintain and reuse. Secondly, we use feedback control technology to set the values of key variables dynamically. Comparing to most of existing methods which set the values statically at design time, it can reflect the changing environment in real time and provide a more reasonable way to guide the system to run in an optimal state.

However, we just study how to reconfigure system at runtime driven by a single object. In real-world applications, we often need to take into account multiple objects at the same time, and even include conflict objects (e.g. security vs response time). Therefore, we will research runtime tradeoff about different quality objects and functional objects in our future work.

## ACKNOWLEDGMENT

This work is supported by the Innovation Program of Shanghai Municipal Education Commission under Grant No.11YZ251, the Startup Funding of SSPU Scientific Research under Grant No.A20XK11X003 and No.EGD14XQD13, the Shanghai University Scientific Selection and Cultivation for Outstanding Young Teachers in Special Fund under Grant No.ZZegd12008, and the Key Disciplines of Software Engineering of Shanghai Second Polytechnic University under Grant No. XXXZD1301.

## REFERENCES

[1] A.Filieri, C.Ghezzi, A.Leva, M.Maggio, "Reliability-Driven Dynamic Binding via Feedback Control", Proceedings of the 2012 international workshop on Software engineering for adaptive and self-managing systems (SEAMS '12), 4-5 June 2012, pp.43-52.

[2] G. Kiczales, "Aspect-oriented programming", In Proceedings of the European Conference on Object-Oriented Programming (ECOOP 1997), Invited presentation, 1997.

[3] Jeffrey O. Kephart, David M. Chess, "The vision of autonomic computing", *Computer* 36(1), 2003, pp.41-50.

[4] <http://www.cs.kent.ac.uk/people/rpg/cb492/saaf/concept.html>.

[5] M.Pinto, L.Fuentes, J.Troya, "A Dynamic Component and Aspect-Oriented Platform", *The Computer Journal* 48(4), 2005, pp.401-420.

[6] <http://docs.spring.io/spring/docs/2.5.4/reference/aop.html>.

[7] <http://aspectwerkz.codehaus.org/index.html>.

[8] <http://www.aosd.net/>.

[9] Villegas Machado, Norha Milena, Hausi A. Müller, and Gabriel Tamura Morimitsu, "On Designing Self-Adaptive Software Systems", *Software Systems, Revista S&T*, 9(18), 2011, pp.29-51.

[10] M.Shaw and D. Garlan, "Software Architecture: Perspectives on an Emerging Discipline", Prentice Hall, 1996.

[11] Mónica Pinto, Lidia Fuentes, José María Troya, "A Dynamic Component and Aspect-Oriented Platform", *The Computer Journal*, Vol. 48 No.4, July 2005, pp.401-420.

[12] E. Truyen and W. Joosen, "Towards an aspect-oriented architecture for self-adaptive frameworks", in proceedings of the 2008 AOSD workshop on Aspects, components, and patterns for infrastructure software (ACP4IS '08), 2008, pp.1-8.

[13] A. Tesanovic, R. Teanovic, D. Nystrom, J. Hansson, and C. Norstrom, "Towards Aspectual Component-Based Development of Real-Time Systems", in Proceedings of the 9th International Conference on Real-Time and Embedded Computing Systems and Applications, Springer-Verlag, 2003, pp.278-298.

[14] Shen JR, Sun X, Huang G, Jiao WP, Sun YC, Mei H, "Towards a unified formal model for supporting mechanisms of dynamic component update", ESEC/FSE-13 Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering, 2005, pp.80-89.

[15] Daniel Sykes, William Heaven, Jeff Magee, Jeff Kramer, "From Goals To Components: A Combined Approach To Self-Management", Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems (SEAMS '08), 2008, pp.1-8.

[16] Daniel Menascé Hassan Gomaa, Sam Malek, João Sousa, "SASSY: A Framework for Self-Architecting Service-Oriented Systems", *IEEE Software*, vol. 28, no. 6, Nov.-Dec. 2011, pp.78-85.

[17] Gomaa, H, Gomaa, H, "Dynamic self-adaptation for distributed service-oriented transactions", Proceedings of the 2012 international workshop on Software engineering for adaptive and self-managing systems (SEAMS '12), 4-5 June 2012, pp.11-20.

[18] Yuriy Brun, Giovanna Marzo Serugendo, Cristina Gacek, Holger Giese, Holger Kienle, Marin Litoiu, Hausi Müller, Mauro Pezzè Mary Shaw, "Engineering Self-Adaptive Systems through Feedback Loops", *Software Engineering for Self-Adaptive Systems*, Springer-Verlag, Berlin, Heidelberg, 2009, LNCS, vol. 5525, pp.48-70.

[19] Andrzejak, A., Herman, U., Sahai, A, "FEEDBACKFLOW - An Adaptive Workflow Generator for System Management", ZIB-Report 05-12 (Februar 2005), 2005, pp.1-10.

[20] Danny Weyns, M. Usman Iftikhar, and Joakim Soderlund, "Do External Feedback Loops Improve the Design of Self-Adaptive Systems? A Controlled Experiment", Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '13), May 18th-26th, 2013, pp.3-12.

# Persona Security: A Technique for Supporting the Elicitation of Security Requirements

Marco Santos, Jacilane Rabelo, Raimundo Barreto, Tayana Conte  
Universidade Federal do Amazonas (UFAM)  
Manaus, Amazonas – Brasil  
{marco.santos, jaci.rabelo, rbarreto, tayana}@icomp.ufam.edu.br

**Abstract**— Safety-critical embedded systems have assisted people in the execution of daily tasks, causing a search for security approaches in the initial phases of the development. The elicitation of security requirements in such systems is a key element for the definition of secure software. Nonetheless, security requirements are mostly ambiguous, incomplete or even not considered, which may be the reason of accidents. In this sense, it is essential to provide a good comprehension of peoples' roles within the development process, and the interactions with the safety-critical embedded systems. This paper presents the Persona Security technique which aims at aiding software engineers in the elicitation of security requirements. We have performed a pilot study to verify the perception and relationship between the easiness of eliciting security requirements, and find out improvement points. The initial results indicate that dividing the requirements into both system and users allows the software engineer to have a clearer and more general view of the system. However, we also identified improvement opportunities such as: (a) the need for identifying requirements related to users and generating their traceability with the software requirements; and (b) the need for the identification of the security requirements that are most relevant for the problem domain, showing their dependencies and their selection impact.

**Keywords:** Security Requirements; Safety-critical Embedded Systems; Software Requirements; Requirements Elicitation, Personas.

## I. INTRODUCTION

Safety-critical embedded systems (SCESs) are systems in which there is a possibility to cause accidents [1]. Increasingly, SCESs are assisting people in the execution of everyday tasks in areas such as [2]: aircraft flight control, medical devices, weapons, and nuclear systems. Safety-critical software is commonly embedded into some greater technical system [3]. Thus, they represent systems or applications in which failures and errors can lead to severe consequences, harm to properties or the environment, or even death [2].

The major cause for accidents pertinent to SCESs is associate to the poorly performed requirements elicitation or inconsistencies in what the software should do [4]. In this sense, the requirements elicitation is one of the knowledge areas in software requirements [5]. According to Lahoz *et al.* [6], the elicitation should be capable of producing requirements that preserve the security system's behavior at any stage, condition or circumstance. The application of good practices

for elicitation can avoid errors in the identification and comprehension of requirements [6].

Functional requirements specify what the system should do or should make happen. On the other hand, security requirements specify what the system should not do or what must be kept from happening [7]. In this sense, security requirements are considered the main non-functional requirements that must be provided in a project on SCESs [8]. According to Firesmith [9], security requirements are frequently ambiguous, incomplete or even not treated, which may be the cause of accidents. Consequently, such requirements are considered an essential element to prevent errors and security attacks [10].

In order to assist software engineers in the elicitation of security requirements for SCESs, we have proposed the Persona Security technique. This paper presents how we developed the Persona Security technique and how to apply it through an example. The initial results show that the fact of dividing the requirements into system and requirements related to users provides a general and clearer view of the software to be developed. The functional requirements and the requirements related to users were listed and also traced. Moreover, in the security requirements lists, we noted a difficulty in verifying which requirements were more important, since all requirements were considered representative due to the criticality of the software. We also verified the need for improving the security requirements list, by identifying the security requirements that are more relevant to the problem domain and providing their dependencies. Such improvements will aid software engineers in the selection of security requirements prioritized according to the analyzed critical system.

The remainder of this paper is organized as follows. Section II presents the theoretical background and related work, our findings suggest a deficiency to approaches elicitation of security requirements. In Section III we describe the proposed Persona Security technique. Then, in Section IV we present the results from a pilot study in order to identify improvement opportunities in our proposal. Finally, Section VI presents the final remarks for this paper.

## II. BACKGROUND

Safety-critical embedded systems are present in software for cars, trains, airplanes, and possibly within hundreds of

critical security components [10]. Due to the increasing demand on security software, eliciting security requirements has turned into an important task. In most cases, such requirements are included in a system in order to verify [11]:

- Authority problems: in which access to the system must only be granted to authorized users;
- Integrity problems: in which the integrity of the system must be maintained in order to avoid accidental or malicious attacks.

Generic requirements are one of the categories of security requirements [12]. In this sense, generic security requirements are a collection of characteristics and design restrictions to solve security problems in common software. Such requirements are used in different programs and environments, and can be extracted based on good practices and previous experience in well succeeded projects [13]. In this sense, Yang [14] classified generic security requirements into eight categories according to their content: dangerous command-related, initialization-related, data processing-related, detection of failure-related, control of safety critical functions-related, assurance of safety state-related, human-computer interaction-related, and other generic safety requirements.

According to Barry [15], “Requirements Engineering approaches assume that a system is designed for users”. Human Computer Interaction (HCI) covers getting acquainted with people as members of groups or organizations, the conditions under which subjects are likely to want to use their device, as well as the characteristics involved in this interaction [16]. In this sense, Personas is a technique proposed in HCI that offers an understanding of the users of the system regarding their characteristics, needs and goals [17]. Therefore, it defines a persona, which is a profile of a user that is described through data elicitation with users by applying techniques such as interviews or observations [15].

The Personas technique can improve the requirements engineering activities, by identifying the users’ needs [16]. In this context, Castro et al. [18] developed Persona\*, which consists of a set of inter-related activities that lead to the creation of personas. In this sense, Persona\* can allow the elicitation of generic requirements. However, it does not focus on the elicitation of security requirements.

Aiming at developing secure and useful systems, we identified two main challenges in the literature: (a) how can we understand the involved roles in the development process, and their interaction with the safety-critical embedded systems? [2][19] and, (b) how can we consider the security of the system in the requirements engineering process of critical systems? [15]. In order to meet these challenges, we have developed the Persona Security technique.

### III. THE PROPOSAL OF THE PERSONA SECURITY TECHNIQUE

The Persona Security technique is composed of a set of grouped activities that lead to the creation of personas and aid in the elicitation of security requirements. In order to minimize risks in the development of software, the Persona Security technique offers a categorized list of generic security requirements. Such requirements were extracted from the

Marshall Space Flight Center (MSFC) [12] and have been classified according to Yang’s [14] approach. It is noteworthy, that this list was checked by another experienced software engineer.

To facilitate its application, the Persona Security technique also provides an execution guide, which explains the steps to be followed when applying it. This way, whoever uses the document will have both instructions and examples within a single document. In this sense, Fig. 1 and 2 show part of the categorized list of the security requirements and part of the execution guide, respectively.

Category	ID	Generic Security Requirements
Human-Machine Interaction	SR01	The software must provide status on all controllable restrictions to the technical team, or to the control executive.
	SR02	The software must incorporate the capability to identify and show status alerts for each software restriction related to critical commands.
Dangerous Command-Related	SR06	Rewriting commands must require multiple actions from the operator.
	SR12	The software must provide a unique and independent command to control each controllable restriction.
	SR13	All software restrictions that are associated to a command must have a unique ID.
Initialization-Related	SR17	The software must start and restart replaceable units in a safe state.

Figure 1: Part of the categorized list of the security requirements

You received two documents: (1) a scenario composed of a list of functional requirements for the persona (profile) and the safety-critical embedded systems; (2) the categorization of the generic security requirements. Based on that, you must map the functional and security requirements using the instructions contained within this document.

#### Initial Procedures:

- Describe the persona profile – *see document (1) – scenarios with functional requirements*
  - << *The description of a persona is composed of: attitudes, goals, needs and behavior* >>
    - **Example:** “John is a 64 year old man who values his family. He is an experienced driver. Other people see him as a rational, practical and self-confident man with authority. He can easily get annoyed with drivers of other vehicles and (...).”
- Add the ID of the **Functional Requirements of the persona (Profile)** – *scenarios with functional requirements*
  - *Example: FR01, FR02.*
- List the ID of the **Functional Requirements for the safety-critical embedded systems** – *scenarios and functional requirements*
  - *Example: FR11, FR09.*
- Add the ID of the **Security Requirements for the safety-critical embedded systems** - *see document (2) - categorization of security requirements*
  - *Example: SR01, SR02.*

Figure 2: Execution guide for the Persona Security technique

Fig. 3 shows the execution process of the Persona Security technique, describing the interaction between its activities:

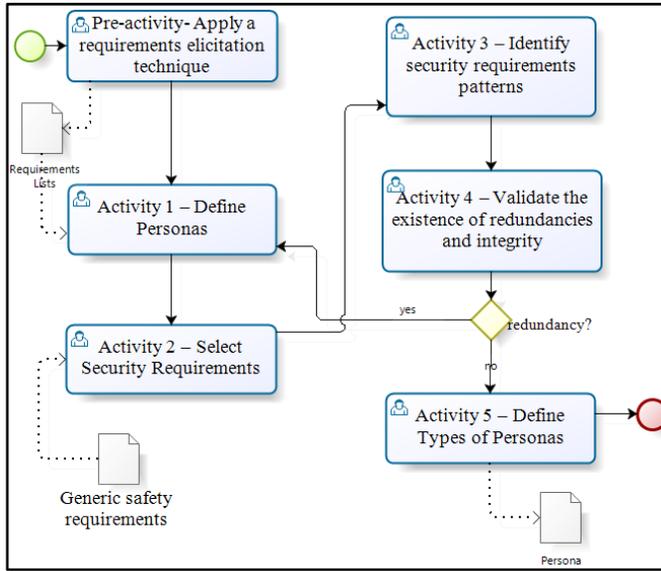


Figure 3- Activities for applying the Persona Security technique

In this sense, the activities for applying the Persona Security technique are:

- This activity is framed in the requirements elicitation phase [20]. The requirements elicitation aims at identifying the origin of the requirements and how to collect them [20]. According to Kasirun and Salin [21][1], there are several elicitation techniques: interviews, focus group, scenarios, discussion groups, prototyping and Joint Application Development (JAD);
- Activity 1 – Define Personas: Based on the results from the previous activity, the users’ needs from the critical system are groups. In the end, each profile will represent a persona and will be used to create a persona document. This activity is included in the following phases [20]: Requirements Elicitation, Requirements Analysis, Requirements Specification and Requirements Validation;
- Activity 2 – Select Security Requirements: During this activity, a list of categorized generic requirements (as shown in Figure 1) is provided to the interviewees. Then, the interviewees must select the requirements that can be applied to the system;
- Activity 3 – Identify security requirements patterns: The main goal of this activity is to identify a list of security requirements that is representative of the SCES;
- Activity 4 – Validate the existence of redundancies and integrity: During this activity, the analyst or the client’s representative will verify the generated documents in order to identify the existence of redundancy or omission. This activity includes the requirements validation phase;

- Activity 5 – Define Types of Personas: The created personas will be prioritized to determine the most important user profile, or the primary persona [16]. In the case that the critical system possesses only one user profile (persona), that will be the primary persona. Then, we must add the list of representative security requirements (see Activity 3) in order to create a security persona. In this sense, the definition of a persona involves the activities of [9]: requirements analysis and requirements specification.

#### IV. THE EXPERIMENT

Our main motivation for performing this study is to verify if the Persona Security technique aids in the elicitation of security requirements in safety-critical embedded systems. Since this was a preliminary evaluation of the technique, the pilot study was performed in an academic environment. We choose an academic environment since we would not require spending real software practitioner’s hour costs. Also, an academic environment is smaller and allows the observation and testing of new technologies to be transferred from academy to industry [22].

This pilot study was performed following three well-defined activities which will be explained below:

**Planning:** First, we defined the goal of this study based on the GQM (Goal-Question-Metric) [23] paradigm, as shown in Table I.

TABLE I. GOAL OF THE PILOT STUDY BASED ON THE GQM PARADIGM

Analyze	Persona Security Technique
For the purpose of	Characterize
With respect to	The perception over: (a) ease of use/assistance for the elicitation of security requirements; (b) aspects that aid or constrains in the application of the technique; (c) recommendations in the utilization of the technique to elicit security requirements
From the point of view	Software Engineering researchers
In the context of	A security requirements elicitation of a safety-critical embedded system performed by a high experienced doctorate student

After defining the goal of this study, we characterized it by describing the activities to be performed, the resources and the trainings to be ministered in order to carry out the study. Finally, we prepared the characterization form, the Consent Term, and post-experiment questionnaire. The characterization form serves to verify: a) knowledge level in requirements elicitation and critical systems; b) their working experience in projects and software development to critical systems (measured in number of projects); c) working time in software development. The consent term informs about data confidentiality and voluntary participation. Lastly, post-experiment questionnaire aims to gather the practitioners’ opinions.

**Execution:** We carried out the pilot study with a highly experienced doctorate student from Federal University of Amazonas (Ufam). He was enrolled in one of the Informatics

doctorate programs in Brazil. According to his characterization form, this student had more than 4 years' experience in developing software (both industrial and academic). Also he had worked in several development projects as an analyst and had prior knowledge with the development of non-functional requirements, mainly usability.

We handled the subject instructions on how to apply the technique. Also, we provided the guide for using the Persona Security technique, which is shown in Fig. 2. Such guide already illustrates an utilization example. The guide for using the Persona Security technique shows examples on how it should be filled. We also provided a scenario of a safety-critical embedded system composed of functional requirements and a persona (profile). Such scenario is from a real software development process from an Unmanned Ground Systems. In this sense, part of the scenario is shown in Fig. 4.

1. Description  
 This document describes the requirements for a robotic mobile vehicle, which represents an example of unmanned system where even if it has automated actions, it still depends on the control of an operator. (...)  
 The system shall meet the following functional requirements:  
 (...)

- FR08: The system shall allow the robotic vehicle to send data and that it is remotely controlled following wireless networks patterns (Wi-Fi);
- FR09: The system shall provide the operator with an adequate way to control the vehicle;
- FR10: The system shall allow the vehicle to be remotely controlled through a computer;
- FR11: The system shall allow the vehicle to be remotely controlled through a cellphone;
- FR18: The system shall allow the vehicle to transport an object in order to perform specific activities;
- FR24: The system shall allow the operator to remotely start, or interrupt the vehicle's functioning.

(...)

Figure 4: Part of the scenario of the robotic vehicle

Moreover, the subject received a list of categorized generic security requirements. Fig. 1 shows part of the categorized generic security requirements list. Then, we gave the subject the template of the Persona Security technique so that he could list: (a) the critical system functional requirements; (b) requirements for the personal profile; and (c) security requirements relevant to the analyzed scenario. Finally, the subject answered a questionnaire regarding ease of use, aspects that could aid and any constraints with the applicability of the Persona Security technique. Fig. 5 shows part of such questionnaire.

1. In your opinion, the Persona Security technique facilitated/assisted in the elicitation of security requirements? Please Comment.
2. Would you change anything in the Persona Security technique? Please Comment.
3. Would you apply the Persona Security technique again? Please Comment.
4. Would you recommend the Persona Security technique? Please

Figure 5: Evaluation Questionnaire

**Analysis and Results:** In this stage we analyzed the results from this study using qualitative analysis procedures. Qualitative studies allow a wider comprehension of the studied phenomenon, which is necessary for analyzing complex questions in software engineering [24]. Besides the possibility of answering research questions that involve variables that are difficult to quantify; and supporting answering the reason for research questions that were approached in other quantitative studies, qualitative methods allow the researcher to deepen into the complexity of the problem instead of abstracting it [24]. We have analyzed the data from the evaluation questionnaire using concepts from Grounded Theory (GT) [25].

GT is a qualitative method for data analysis, in which a theory is derived from data, systematically gathered and analyzed by means of the research process. The GT method is based on coding – the analytic processes through which data are fractured, conceptualized, and integrated to form a theory [25]. The data analysis began with the open coding of the questionnaires (Fig. 5). The qualitative analyses of the results from the study are described as follows.

## V. RESULTS FROM THE PILOT STUDY

During the execution of the study, two researchers observed the subject while he applied the Persona Security technique. Through observation, the researchers were able to verify and make notes regarding how the subject applied the technique. Right from the beginning of the application of the technique, the subject asked if he could relate (map) the requirements from the persona profile with the robotic mobile vehicle. Since, we were interested in verifying improvement opportunities and understand how would the Persona Security technique would be applied, the researchers answered that the subject could list the requirements in the way that would be more suitable and easier to use for him.

In this sense, the researchers notes that the subject applied the following strategy:

- First, the subject created a representation to relate the requirements using arrows allowing make the traceability between requirements involved, where:
  - a)  $A \rightarrow B$ : meaning that B is necessary in order to achieve A;
  - b)  $A \leftrightarrow B$ : meaning that A and B are mutually necessary for each other .
- Then, the subject identified the requirements regarding the persona profile and related such requirements using the approach described above;
- After that, the subject identified the requirements regarding the critical system and related them using the same approach. In this sense, the relationship was made for both the persona and the critical system functional requirements;

In Fig. 6 we show part of the document as how it was filled by the subject.

When analyzing the results from the elicitation and the relationships created by the subject, we noticed the importance of requirements traceability. The fact that it is necessary to first

identify the persona requirements and then map such requirements with critical system made it easier for the subject to understand how the system works and what it must do.

Document TEMPLATE for the Persona Security technique	
<b>Persona description</b>	Peter is a 37 year old man that works in research for NASA. Peter is very careful and is very solicitous about the equipments that are left under his supervision. (...)
<b>Functional Requirements of the Persona (Profile)</b>	<pre> graph TD     FR09 &lt;--&gt; FR24     FR09 &lt;--&gt; FR26     FR09 &lt;--&gt; FR27     FR10 &lt;--&gt; FR09     FR11 &lt;--&gt; FR09     FR12 &lt;--&gt; FR09           </pre>
<b>Functional Requirements of the Mobile Robotic Vehicle</b>	<pre> graph TD     FR10 &lt;--&gt; FR18     FR11 &lt;--&gt; FR18     FR12 &lt;--&gt; FR18     FR18 &lt;--&gt; FR20           </pre>
<b>Security Requirements for the Mobile Robotic Vehicle</b>	It will be marked the requirements directly into the list.

Figure 6: Requirements List on Persona Security

In this sense, when comparing the list of requirements from the subject with those in the oracle (in this context, a document containing the corrected elicited requirements, which is used to verify the subjects answers), we verified that the subject managed to identify all requirements from the persona and the robotic system correctly. On the other hand, the subject was not able to trace four requirements (three from the persona and one from the system). Therefore, one identified future research question for this research is: “Tracing the requirements from the persona with those for the critical system allows a better understanding of how the system works?” Moreover, another research question is: “Relating the requirements facilitates the identification of which security requirements are more critical for the system?”

When identifying the security requirements for the system, the subject ended up selecting and rejecting the requirements directly in the provided document. Thus, we verified that such document should be improved. Initially, we thought this document would be used as a reference document (see the original version in Fig. 1), however, it acts as a checklist and we must add check squares in order for it to be operationalized. By doing such change, software engineers applying the Persona Security technique will not waste time listing the requirements.

We noticed that the subject had difficulties in identifying which security requirements would be suitable for the system, as he considered that all of them were important. The following quote shows such concern:

*“(.) I don’t know where exactly the security requirements would fit, since I actually want to accept and include them all” – Subject*

Regarding **ease of use / assistance** of the Persona Security technique to elicit security requirements, the subject stated that the technique facilitates its use.

*“The fact of dividing the requirements both for the persona and the automated part allowed me to have a clearer view of the system...” – Subject*

It is noteworthy that when knowing about the criticality of the system, the subject also became insecure regarding the identification of security requirements.

*“(.) when I knew how critical the system was, and imagining use situations, I wanted to mark all security requirements” – Subject*

Based on these answers, we noticed that the list of generic security requirements will need to be modified. In this sense, when asked about if he would change the Persona Security technique, the subject answered:

*“Yes, I would change it. I would categorize the security requirements according to their priority and would verify with the client” – Subject*

To make this changes, we will map the security requirements that are more relevant to the problem domain, and which are its interrelated requirements and which are not necessary. For instance:

- For the mobile robotic system the SR01 requirement is essential. When selecting that requirement, the requirements SR08, SR09 and SR22 are mandatory. Moreover, requirements SR34 and SR35 are not important for this problem domain. Then, requirements SR01, SR08, SR09 and SR22 will be selected.

When asked about **Using the Persona Security technique Again**, the subject added:

*“Yes, I would use it to know the system. But I am not quite sure in which part the security requirements would fit...” – Subject*

The subject’s answer to this question allowed us to verify the need to improve the list of generic security requirements. In this sense, a further research question would be: “Categorizing the list of generic security requirements by essentiality and priority would facilitate the elicitation process?”

The subject also stated that he would **Recommend the Persona Security technique**:

*“Yes, but a generic approach to aware the development team of the importance of the security requirements, which can be defined based on the list” – Subject*

Although, the subject was able to correctly identify all requirements regarding the persona profile, it was not clear for him why this was necessary:

*“Why do you need a single persona? What is the influence of such persona in the elicited requirements? This was not clear.” – Subject*

The creation of a persona profile allows the presentation of the user needs and how an interface must be designed to meet them. The scenario presented in this pilot study was limited to just one persona, which restricted the analysis of the integration

of the Personas technique to the proposed Persona Security technique. After improving the Persona Security technique, we will perform a feasibility study to verify the acceptance of the approach. In this context, a new scenario will be investigated with more than one persona profile in order to verify the importance of such profiles in the elicitation of security requirements.

## VI. CONCLUSIONS

This paper presented the Persona Security technique, which proposes to support the elicitation process of generic security requirements of Safety-critical embedded systems. We have carried out a pilot study to verify the way in which the technique can be applied to identify the system's requirements of the persona profile, as well as the security requirements. Such study was performed with a doctorate student in the Informatics program at Federal University of Amazonas. The subject had knowledge in the requirements elicitation process.

We have analyzed the results using qualitative analysis approaches. Thus, we identified that the Persona Security technique aided the subject in understanding the critical embedded system as a whole, and even have a view of what risks can be involved in the chosen scenario. An interesting fact was that the subject traced/mapped the requirements, and this helped him to get to know the system. Moreover, the subject had difficulties in identifying which security requirement was needed in the critical embedded system, showing that there is still room for improvement in the development of the proposed technique.

As future work, we intend to: (a) perform a systematic review on security requirements elicitation in SCESs, to make possible to compare the technique with other proposals in the field of security requirements elicitation, and evolve the list of generic security requirements, providing an indicator of which security requirements can be applied depending on the specific context of the application; (b) carry out a new empirical study with a higher number of subjects to validate and evaluate the improvements on the Persona Security technique; (c) verify whether the addition of traceability to the technique is feasible; (d) figure out if it is possible to automate parts of technique. Thus, by adding such suggestions in Persona Security technique, we intend to improve its value, allowing software development teams may add greater certainty for safety-critical embedded systems.

## ACKNOWLEDGMENTS

We would like to thank the financial support granted by: CAPES (Doctorate); FAPEAM through process number 01135/2011 (PRONEX Project); and Project N.021/2011 (Universal Amazonas). We would like to thank Luis Rivero for the translation and textual revisions of this paper and also the practitioner who participated in this study.

## REFERENCES

[1] B. S. Medikonda, and P. S. Ramaiah "Integrated Safety Analysis of Software-Controlled Critical Systems", ACM SIGSOFT Software Engineering Notes, 35(1), 2010, pp. 1-7. J. C.

[2] Knight, "Safety critical systems: challenges and directions", ICSE '02: Proceedings of the 24th International Conference on Software Engineering, New York, NY, USA, 2002, pp. 547-550.

[3] S. Nejati, M. Sabetzadeh, D. Falessi, L. Briand, T. Coq, "A SysML-based approach to traceability management and design slicing in support of safety certification: Framework, tool support, and case studies", Information and Software Technology, v.54, n.6, 2012, pp. 569-590.

[4] N. G. Leveson. "The Role of Software in Spacecraft Accidents", AIAA Journal of Spacecraft and Rockets, Vol. 41, No. 4, July 2004.

[5] E. Nasr, J. Mcdermid J, and G. Bernat, "Eliciting and Specifying Requirements with Use Cases for Embedded Systems", Proceedings of the 7th International Workshop on Object-Oriented Real-Time dependable systems (WORDS), 2002, pp. 350 - 357.

[6] C. H. N. Lahoz, J. B. Camargo, M. A. D. Abdala, and L. A. Burgareli, "A Software Safety Requirements Elicitation Study on Critical Computer Systems", the 1st Institution of Engineering and Technology International, 2006, pp. 47-53.

[7] D. G. Firesmith, "A Taxonomy of Safety-Related Requirements", International Workshop on High Assurance Systems (RHAS'05), 2005.

[8] A. Armoush, E. Beckschulze, and S. Kowalewski, "Safety Assessment of Design Patterns for Safety-Critical Embedded Systems". Software Engineering and Advanced Applications (SEAA '09), 35th Euromicro Conference on, 2009.

[9] D. G. Firesmith, "Engineering safety-related requirements for software-intensive systems", Proceedings of the 27th international conference on Software engineering, St. Louis, MO, USA, 2005.

[10] D. Zeckzer, P. Liggesmeyer, Mickel, "Identification of Security-Safety Requirements for the outdoor robot RAVON using Safety Analysis Techniques", Software Engineering Advances (ICSEA), 2010, pp. 508 - 513.

[11] X. Chen, and J. Liu, "Eliciting Security Requirements in the Commanded Behavior Frame: An Ontology based Approach", Knowledge Systems Institute Graduate School (SEKE), 2012, pp. 61-65.

[12] NASA-GB-8719.13-2004, NASA Software Safety Guidebook [S], 2004.

[13] X. Xu, X. Bao, M. Lu, and W. Chang, "A study and application on airborne software safety requirements elicitation", 9th International Conference Reliability, Maintainability and Safety (ICRMS), 2011, pp. 710-716.

[14] C. Yang, X. Bao, D. Zhong, and Z. Li, "Designing generic safety test cases for airborne software", 9th International Conference Reliability, Maintainability and Safety (ICRMS), 2011, pp. 737-741.

[15] S. Faily, and I. Fléchaïs, "Barry is not the weakest link: eliciting secure system requirements with Personas", Proceedings of the 24th BCS Interaction Specialist Group Conference, United Kingdom, 2010.

[16] S. T. Acuña, J. W. Castro, and N. Juristo, "A HCI technique for improving requirements elicitation", Information and Software Technology, vol. 54, n. 12, 2012, pp. 1357-1375.

[17] A. Cooper, R. Reimann, and D. Cronin, "About Face 3.0: The Essentials of Interaction Design", Wiley Publishing, Indianapolis, 2007.

[18] J. W. Castro, S. T. Acuña, and N. Juristo, "Integrating the Personas technique into the requirements analysis activity", Computer Science, ENC '08, Mexican International Conference, 2008, pp. 104-112.

[19] R. R. Lutz, "Software Engineering for Safety: a Roadmap", Proceedings of the Conference on The Future of Software Engineering, Limerick, Ireland, 2000, pp. 213-226.

[20] IEEE Computer Society Professional Practices Committee, Guide to the Software Engineering Body of Knowledge (SWEBOK), v. 222010, edition 3.0, Los Alamitos, CA, 2004.

[21] M. Z. Kasirun, and S. S. Salim, "Focus Group Discussion Model for Requirements Elicitation Activity", Computer and Electrical Engineering, ICCEE, 2008, pp. 101-105.

[22] F. Shull, J. Carver, G. Travassos, "An empirical methodology for introducing software processes". ACM SIGSOFT Software Engineering Notes, v. 26, n. 5, 2001, pp. 288 -296.

[23] V. Basili, and H. Rombach, "The tame project: towards improvement-oriented software environments", IEEE Transactions on Software Engineering, v. 14, n. 6, 1988, pp. 758-773.

[24] C. B. Seaman, "Qualitative Methods in Empirical Studies of Software Engineering", IEEE Transactions on Software Engineering, v. 25, n. 4, 1999, pp. 557-572.

[25] A. Strauss, J. Corbin, "Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory", 2 ed. London, SAGE Publications, 1998, 400pp

# Runtime Code Reuse Attacks: A Dynamic Framework Bypassing Fine-Grained Address Space Layout Randomization

Yi Zhuang<sup>1</sup>, Tao Zheng<sup>1,2</sup>, Zhitian Lin<sup>1</sup>  
Software Institute<sup>1</sup>. National Key Laboratory for Novel Software Technology<sup>2</sup>  
Nanjing University  
Nanjing, China  
{mg1132019, zt, mg1232007}@software.nju.edu.cn

**Abstract**—Fine-grained address space layout randomization has recently been proposed as a method of efficiently mitigating ROP attacks. In this paper, we introduce a design and implementation of a framework based on a runtime strategy that undermines the benefits of fine-grained ASLR. Specifically, we abuse a memory disclosure to map an application’s memory layout on-the-fly, dynamically discover gadgets and construct the desired exploit payload, and finish our goals by using virtual function call mechanism—all with a script environment at the time an exploit is launched. We demonstrate the effectiveness of our framework by using it in conjunction with a real-world exploit against Internet Explorer and other applications protected by fine-grained ASLR. Moreover, we provide evaluations that demonstrate the practicality of run-time code reuse attacks. Our work shows that such a framework is effective and fine-grained ASLR may not be as promising as first thought.

**Keywords**—code reuse; security; dynamic; fine-grained ASLR

## I. INTRODUCTION

Software vulnerabilities have been a major cause of computer security incidents. Buffer overflows [3, 5], integer overflows and heap overflows were used to pose a significant threat to modern operating systems [1]. Format string vulnerabilities [2] allow an attacker to control the first parameter to a function of the printf-family, which can be used to store pointers to specific addresses if it is placed on the stack. Despite differences in the style and implementation of these exploits, they all share a same goal: to achieve the control-flow hijacking attempts within the vulnerable application. Nowadays, numerous defenses have been implemented to limit the scope of these attacks. However, well motivated attackers still succeed in their intent. So the cat and mouse game plays on.

To thwart such attacks, many mitigation techniques have been developed. Address Space Layout Randomization (ASLR) [9] and Data Execution Prevention (DEP) are very real thorns in the side of an attacker. DEP makes locating shellcode difficult; the attacker must find a page with executable permission and find a way to write to it when it has writable permission and figure out the location. Attackers then redirect to code reuse attacks. This new strategy utilizes code already present in memory, instead of relying on code injection. The canonical example is return-to-libc [6, 8], in which exploits redirect control-flow to existing shared-library functions. But if

the base address of the memory segment is randomized, then the success rate of such an attack significantly decreases. Shacham [7] introduced a new approach named return-oriented programming, which chains together short instruction sequences ending with a ret instruction (called gadgets) that already exists in the memory of the application and executes some specific computation. The key idea of ASLR is to randomize the base address of the stack, heap, code, and dynamic libraries at load and link time, which offered a plausible defensive strategy against these attacks. But a drawback of this approach is that not all memory regions have been protected with ASLR, the address space for 32bit binaries is small which opens the possibility of probabilistic attacks [17]. Besides that, ASLR on 32-bit architectures only leaves 16 bit of randomness, an attacker might attempt to perform a brute-force attack.

After that, smart defenders have been busily working to fortify perimeters by designing fine-grained randomization strategies [21] for repelling the next generation of wily hackers. Some approaches introduce randomness at compile time. For example, compilers can be modified to generate code without ret instructions [25]. But these mechanisms fail to handle attacks leveraging jmp instructions. Marlin, introduced by Gupta [18], randomizes the function-level structure of the executable code, so denying attacker the necessary a priori knowledge of instruction addresses for constructing such a desired exploit payload. Other approaches have also been proposed to randomize processes. STIR [19] and XIFER [26] defend against ROP by randomizing at the basic block granularity. ILR [24] randomizes the location of each instruction in the virtual address space and uses a process-level virtual machine to find the called code, which imposes a significant on-going performance cost. Therefore, the attacker is unaware of how exactly permutation is randomized for the currently executing process image. So the radiance of traditional ROP attack is fading away. We define these mitigations of ROP embodied by fine-grained ASLR.

In this paper, through memory disclosure, we implement a completely new variant of code reuse attack wherein we gather code chunks and retrieve them to the desired payload dynamically from memory layout during the vulnerable application is running. Then treating the calling of virtual

function as the trampoline, replace the address of function in vtable with gadget's first addresses in proper sequence. Finally trigger the call of function to complete our ROP attack on-the-fly. We show strong evidence that our variant ROP attack can entirely bypass all fine-grained randomization scheme and ROP mitigation. Based on the above findings, we argue that the fine-grained ASLR strategies still have loopholes. Meanwhile, we hope that our work will inspire others to explore more comprehensive defensive strategy than what exists today.

## II. BACKGROUND

We review the necessary technical background information before introducing the methodology behind our attack.

### A. Code Reuse Attacks

The fundamental factor for code reuse attack is that the relative offsets of instructions in the application's code are constant. That is to say, if an adversary knows any symbol's address in the application code, then the location of all gadgets and symbols in application's code is deterministic.

Return Oriented Programming [14] is generalization of return-to-libc attack [6, 8], which involves an adversary redirecting the program execution to an existing library function [12]. The general principle of any ROP attack is to combine short instruction sequences found in memory (or whatever code is not randomized), called gadget, and allowing an adversary to perform arbitrary computation. Recently, this concept was overthrown by removing the reliance on return instructions [13]. However, we show the basic idea of code reuse using ROP for simplicity in Figure 1. Steps ① to ⑦ show the entire procedure of ROP attacks.

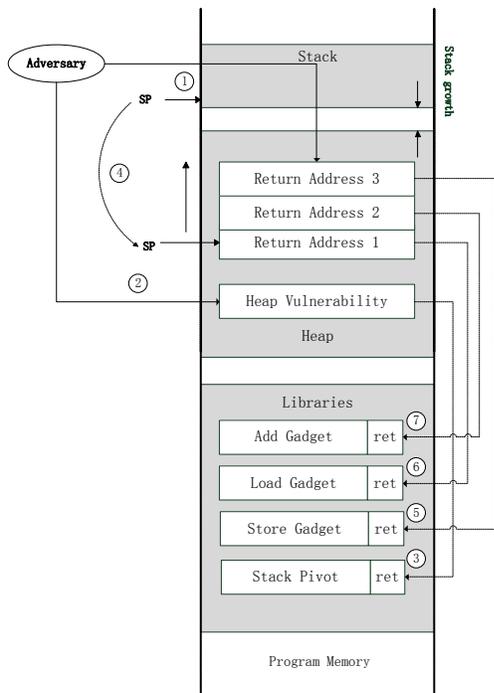


Figure 1. Layout of a sample ROP attack on the heap using a sequence of single-instruction gadgets.

Jump Oriented Programming (JOP) [10, 29] is similar to ROP in that JOP manipulates the control flow of the application. Jump oriented data is not limited to stack overflows but uses modified indirect control flow transfers to construct the chain of executed gadgets. Indirect control flow transfers are used in the application to support, e.g., library calls, function pointers, and object oriented programming. JOP has similar limitations like ROP. In addition, JOP needs to redirect control flow to the first JOP dispatcher. ASLR severely limits the initial redirection for JOP.

### B. Fine-Grained Randomization for Exploit Mitigation

A widely accepted countermeasure against code reuse attacks is that the defender randomizes the application's memory layout. This scheme randomizes the base address of segments such as the stack, heap, shared libraries, and the executable code itself. As is shown in Figure 2, the start address of an executable is changed and sequence of gadgets is shuffled between consecutive runs of the same application, which is not intended by the attacker. So the adversary must guess the location of the instruction sequences needed for deployment of their code reuse attack.

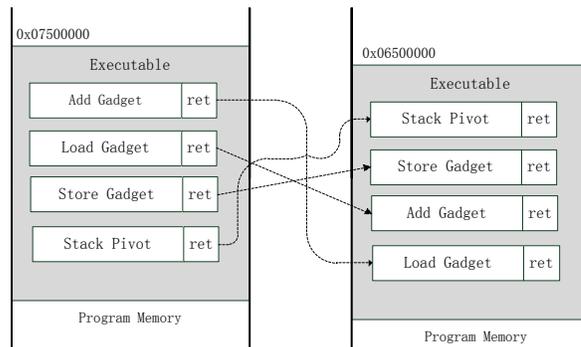


Figure 2. Mitigation of ROP attack by fine-grained memory and code randomization

Unfortunately, current ASLR implementations only randomize on a per-module level, which suffer from two main problems: first, ASLR can be bypassed by means of brute force attacks [15, 17] because the entropy on 32-bit systems is too low. Furthermore, upgrading to 64-bit is simply not feasible. Second, all ASLR solutions are vulnerable to memory disclosure attacks [11, 16] where the attacker gains knowledge of a single runtime address and uses it to re-enable code reuse. Therefore, some gadgets from a disclosed region can be organized deliberately by the adversary offline before deploying the exploit. To confound these attacks, Industrious defenders put forward a number of fine-grained ASLR and code randomization schemes [18, 19]. The broadly idea in these works is to randomize the data and code structure, for instance, by shuffling address of functions or basic blocks. Figure 2 depicts that the performance of these approaches are that the location of all gadgets is randomized.

We assume target platform uses the following mechanisms to mitigate the execution of malicious computations:

- **Non-Executable Memory:** We assume that the target vulnerable application is under the protection of non-executable memory (also called NX or DEP) applied to the heap and the stack. It will make the traditional code-injection attack ineffective. We also assume that this mechanism prevents adversary from tampering all executables and native libraries, in order to stop one from overwriting existing code.
- **Base Address Randomization:** We assume that the target platform randomizes base addresses of library and executable segments effectively. Mappings have been eliminated.
- **Fine-Grained ASLR:** We assume that the target platform deploys strong fine-grained memory and code randomization scheme on executables and libraries. First, target platform shuffles the order of functions [18] and basic blocks [19]. Second, target platform swaps registers and replaces instructions [20]. Third, target platform randomizes the location of each instruction [23] and performs randomization upon each execution of vulnerable application [19].

### III. OVERVIEW OF RUNTIME CODE REUSE ATTACK

Snow [30] has put forward the concept of runtime code reuse and implemented a framework that can launch exploit on-the-fly. But the drawback in this framework is very obvious. First, Snow did not give out the detail about how to obtain the entry point of memory disclosure. Second, Just-in-time compilation is simply too heavy-weight to be consider in a runtime framework. In our framework, we explain how to disclose code segment in detail, propose a new method to chain every gadget which getting rid of dependence on stack and proved to be light-weight.

The overall work flow of our runtime exploit is shown in Figure 3. In Step ①, we arrange objects on the heap in order by defragmenting system heap. Next, we perform an overflow on JS object to disclose the memory and construct our code chunks when target vulnerable application is running. Then we pick up some useful gadget sets from code pages. In Step ②, through setting up fake objects on the heap, we embedded serialized code chunks in our vtable. Through utilizing mechanism of virtual function call in target program, unordered gadget sets become malicious ROP payload. At last, we inject all above information in our exploit script to trigger our malicious attack in Step③.

#### A. Disclosing Memory

Because we assume our target application is running on the most fine-grained platforms, so memory layout of application is totally different at each execution. Therefore, the attempt to apply static code analysis is useless and futureless. The entire procedure of our attack is happening during the vulnerable application is executing. We need to arrange objects on the heap in the following order: (i) buffer to overflow, (ii) string object, (iii) JS object. The state of a process’s heap depends on the history of allocations and deallocations that occurred during the process’s lifetime. Unfortunately, we do not know the heap state resulting from a specific sequence of previous allocations.

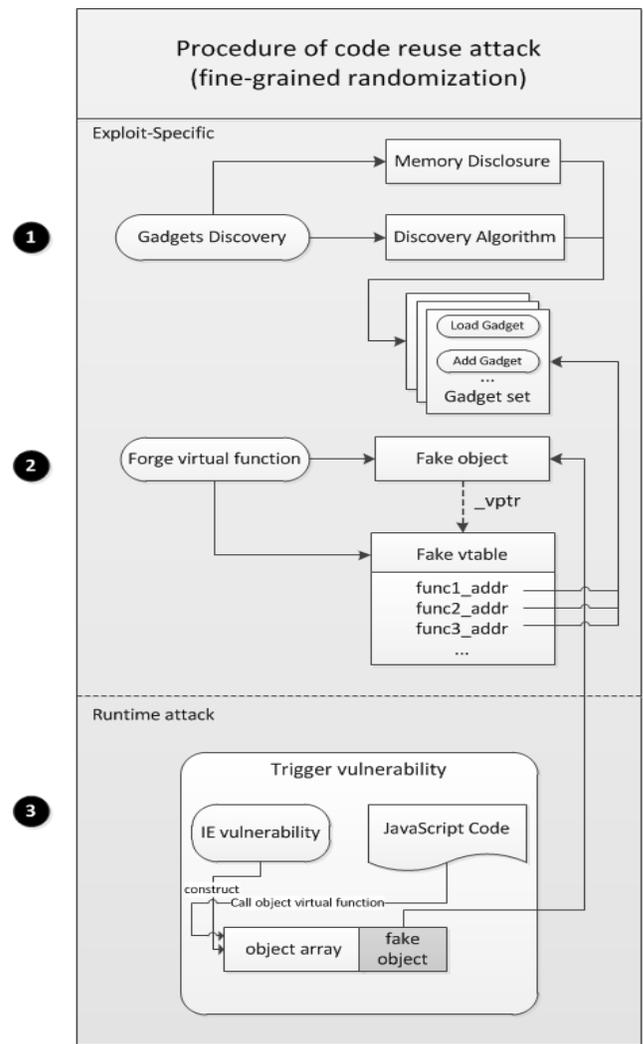


Figure 3. Workflow of memory code reuse attack against a script-enabled application protected by fine-grained ASLR

We can image that there are many holes of free memory. To overcome this hurdle, we must defragment the heap. Our purpose is to fill up these “holes” which are bigger than we allocate later on. This can be done by allocating a large number of blocks of the size we will use. After that, every time we allocate new blocks from the end of the heap. Thus the behavior of the allocator will be equivalent to starting with an empty heap.

One important exception is the data for JavaScript string object which is stored as BSTR [27] used by the COM interface. BSTR strings are stored in memory as a structure containing a four-byte size field, followed by the string data and a two-byte null terminator. The first block is the following string object. We take advantage of unsigned int overflow, so that actually used size in block is smaller than we allocated earlier. Then we fill the last four-byte with “0xffffffff”. So the following string object’s first four-byte is overflowed and string length will cover entire 32-bit address space layout. This will allow us to read bytes at any relative address in memory. As the relative offset from string object to button object is known, we can use the relative read to reveal the absolute

address through self-reference first 4-byte field in third block. We implement a method that, given a relative address, switches it to an absolute address and reveals the corresponding data. In short as memory is discovered, there will be many code chunks in code pages for us to build a useful payload, which we elaborate on in the next sections.

### B. Gadget Discovery

Now our task lies in finding out a number of gadgets to use as our code reuse payload. Not every instruction sequence can be used as a gadget. We choose our usable gadgets in the following three steps:

1) Upon with the code pages dynamically discovered in process's memory, we generate the sequence of instruction gadgets ended with `ret` by adapting the Galileo [4] algorithm and record them in a trie.

2) Discover the gadget that has the same semantic definitions [28] with our desired gadget of payload in the trie. If not found, we disclose the memory to glean new code pages and return to the first step.

3) We filter the gadgets discovered in step two using a kind of instruction dependence method which will be illustrated later. If no gadgets left after the filtration, we return to the first step.

Since fine-grained ASLR mitigation may change the module of code pages on each execution, we could not have much energy to analyze the code chunks offline. What we could do is dynamically collect gadgets at the time exploit runs. Furthermore, we must do this work as quickly as possible because our attack must run in real-time before the victim machine terminates the vulnerable application. Unfettered access to extensive memory address space enables us to search for gadgets. We iterates over the code pages to collect useful gadgets by adapting the Galileo [4] algorithm. After that we use semantic definitions to match gadgets and introduce a kind of instruction dependence method to filter them. In terms of the instruction dependence method, we define the dependence ( $Ins_i, Ins_j$ ) as following.

- If there exists a register  $Reg_a$  which serves as a destination register in  $Ins_m$  and as a source register in  $Ins_{m+1}$ , so the execution of  $Ins_m$  depends on  $Ins_{m+1}$ , we define  $Ins_m$  depends on  $Ins_{m+1}$ .
- If  $Ins_{m+1}$  is a jump instruction, we define  $Ins_m$  depends on  $Ins_{m+1}$ . For example, `jnz eax`,  $Ins_{m+1}$  strictly associates with  $Ins_m$ , which sets up flag register `%eflag`.
- If register is involved in accessing memory, that is,  $Ins_i, Ins_j$  corresponding to opcode `reg_src, [eax+Imm_s]`; opcode `[eax+Imm_t], reg_des`; if value in the register is not changed and  $Imm_s = Imm_t$ , we define  $Ins_i$  depends on  $Ins_j$ ; if value is changed, we define  $Ins_m$  depends on  $Ins_{m+1}$ .
- If  $Ins_m$  and  $Ins_{m+1}$  correspond to stack operation instruction sequence, such as `pop` and `push` instruction, we define  $Ins_m$  depends on  $Ins_{m+1}$ .

A gadget can be determined as usable or not after we check every adjacent two instructions' dependence according to the

above principle. If there are some gadgets left after the three steps, we choose them as our final gadgets and will use their addresses in the runtime attack.

### C. Forging Virtual Function

The next challenge is how to effectively use the collection of gadgets to satisfy the exploit target program. The traditional sense of the ROP attack lies in a stack-based buffer overflow. A ROP attack constructs a set of stack invocation frames that are popped one after another. Each stack invocation frame prepares a set of parameters on the stack and targets a gadget that uses the parameters and executes some malicious computation. ROP works around DEP but relies on static addresses for the stack and for the gadgets. In addition ROP needs to initially redirect control flow to the first ROP invocation frame.

The core notion of our attack is that we replace function addresses in virtual function table with some gadget's address. When the target application use the associated object, the first virtual function is been called, control-flow is redirected by sequentially executing the gadgets in memory which was found in previous step to fulfill adversary's goal. We do not need to arrange gadgets on stack that ends with `ret` instruction with the stack pointer, `%esp`, to execute the next gadget. This method can totally negate the effect of stack protection strategy.

We allocate one blocks of memory on the heap for a fake vtable. In figure 4, fake vtable is an object similar to String object. It has 4-byte head, following by 4-type padding content, next the virtual function addresses, and a 2-byte null terminator. Any virtual function call starts at offset 8 in the vtable. So if we put our gadget's first instruction address in this position, virtual function call through the vtable will jump to a location of our choosing.

Head 4 Bytes	Paddings 4 Bytes	Gadget1_ addr 4 Bytes	Gadget2_ addr 4 Bytes	Gadget3_ addr 4 Bytes	...	Paddings	Terminator 2 Bytes
-----------------	---------------------	-----------------------------	-----------------------------	-----------------------------	-----	----------	-----------------------

Figure 4. Data structure of fake vtable

Another data structure, called fake object, contains the base address of our fake vtable. This procedure has involved lookaside lists maintained by the system memory allocator. The cache consists with a number of bins, each holding 6 blocks of a certain size range. When a block is freed, it is stored in one of the bins. If the bin is full, the smallest block in the bin is freed and replaced with the new block. In terms of our needs, we apply a certain amount of blocks from memory, which size equals to our vtable, in order to make this size on the lookaside list empty. Now let's free this size of block containing a vtable, so there is a free block whose content is original vtable on the lookaside list and lookaside head will point to this block. We construct the fake object by assigning the lookaside head address to the fake object pointer.

### D. Trigger Vulnerability

We overwrite the object pointer with lookaside list base address called fake object pointer in Figure 5. Now the fake object pointer points to the virtual function address in vtable that has been overwrote with our first gadget's base address in memory. Every time we call a virtual function, corresponding

gadget will be executed. We put a number of object pointers into an array. So if an adversary wants to do high-level malicious computation, he just successively uses these pointers to call virtual functions.

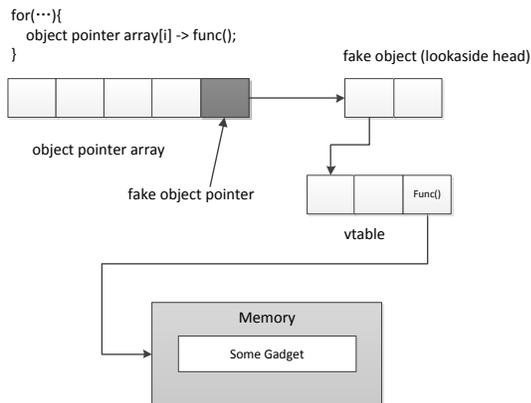


Figure 5. Workflow of virtual function call

Because the times of virtual functional calls determine the scale of our payload constructed by gadgets, if one object is not enough for us, we can construct more objects which include the virtual function calls to increase the power of our attack. After we finish the above steps, what we need is only to invoke vulnerable object which is used by the running application. Finally it will trigger our runtime attack.

### E. Implementation

To prove the power of our framework, we use it to exploit Internet Explorer (IE) 10 running on Windows 7 using CVE-2013-2551. The vulnerability is an integer overflow within certain object to achieve code execution in the context of IE 10 sandboxed process. We put our desired objects on the heap, dynamically traverse code pages, semantically pick up gadgets and executing a serialized payload useable through a series of virtual function calls. All performed at runtime in a script environment.

In Step ①, we develop a JavaScript library called HeapEasy that encapsulate the system allocation operations. For example, the implementation of HeapAlloc interface ensures each blocks allocation comes from the system heap, and implements a HeapLookAside interface to add blocks of the specified size to the lookaside list. Next, iteratively using the charCodeAt() function to read data from the memory code pages and implement a DiscloseAbsAddr interface that translates a relative address to an absolute address. We snapshot the memory state in our hashmap which contains the absolute address of every instruction. At the same time, we dynamically filter the useable gadgets which adapts the criteria from Schwartz [28]. After that, we construct the fake vtable by using HeapVTable interface in Step②.

Generally speaking, a virtual function call needs a MoveRegG to get the vtable address from register ecx, and a JumpG to invoke the function at offset 0x8 in the vtable. Our library enables us to figure out the heap base address and the absolute address of a jmp ecx or equivalent instruction through the debug library called DbgHelp. When IE 10 is running on

the target machine which uses the vulnerable ActiveX widget, our script, included in any HTML or other document-format supporting carriers, starts by an amount of bootstrap code. At last, our payload begins execution in Step③.

## IV. EVALUATION

We evaluated the proposed attack of our framework by applying it on real-world application under the protection of fine-grained randomization. Overall, the evaluation demonstrated the wide-scale applicability of our runtime code reuse framework against Internet Explorer in Windows and other popular applications.

### A. On Runtime Performance

We performed five tests of our framework. The first test uses integer overflow vulnerability in the context of IE10 sandboxed process. The second, third and fourth tests exploit the Internet Explorer plugin on windows. We use a DebugLog function that includes format string vulnerability in the second test. We employed the family of printf functions because of no warnings were reported. In the third test, we took advantage of an arbitrary sequence of commands. So we chose the Windows calculator program. The vulnerability in the fourth test is triggered by creating an ActiveX object and calling its KeyFrame method with an argument overflow larger than 0x07ffffff. The fifth test demonstrates the native performance of our framework.

### B. Evaluation of the Defense against Our Framework

Table 2 shows the popular mitigations used in current operation system. First, our framework does not need to arrange each gadget’s first instruction address on the stack and overflow the return address in EIP. So GS strategy loses her efficiency. The same principle applies to SafeSEH. Second, DEP strategy introduces a no-execute bit (NX) to prevent stack from code-injection attack. Our framework utilizes code already present in memory, so NX strategy does no harm to our framework. In the academic circles, randomization technique at various levels of granularity –function level, block level and instruction level were put forward recently. However, our framework glean code chunks at the same time, the target vulnerable application was loaded in memory. That is to say, the step of collecting gadgets happens after the application was randomized. The runtime feature of our framework determines the fine-grained ASLR strategy is short-sighted.

Table I RUNTIME FRAMEWORK AGAINST POPULAR ATTACK MITIGATIONS

Attack Mitigations		Runtime framework
GS	Stack cookies	✓
	Variable reordering	✓
SafeSEH	SEH handler validations	✓
	Permanent SEH	✓
DEP	NX	✓
ASLR	Function-level	✓
	Instruction-level	✓

## V. CONCLUSION

Fine-grained ASLR has been introduced as an efficient strategy to defend the code reuse attacks like ROP. In this paper, we introduce a dynamical framework that trigger a runtime code reuse by exploiting the virtual function call. Our framework consists of memory disclosure, gadget discovery and fake virtual function call construction. All work is executed after the vulnerable program begins, so the ASLR can hardly affect our experiment. We demonstrate the efficiency and complete some attacks using this framework. This kind of strategy may be a potential signal that future hacker will use the same or similar way to bypass the ASLR and do something dangerous. We hope our work will attract more researchers' attention and inspire them to bring forward more comprehensive defense.

## REFERENCES

- [1] E. H. Spafford, "The Internet Worm Program: an Analysis," SIGCOMM Computer Communication Review, vol. 19, no. 1, pp. 17–57, 1989.
- [2] V. van der Veen, N. dutt Sharma, L. Cavallaro, and H. Bos. Memory errors: The past, the present, and the future. In Symposium on Recent Advances in Attacks and Defenses, 2012.
- [3] Aleph One. Smashing the stack for fun and profit. Phrack Magazine, 49(14), 1996.
- [4] C. Cowan, C. Pu, D. Maier, H. Hintony, J. Walpole, P. Bakke, S. Beattie, A. Grier, P. Wagle, and Q. Zhang. StackGuard: Automatic adaptive detection and prevention of buffer-overflow attacks. In USENIX Security Symposium, 1998.
- [5] D. Litchfield. Defeating the stack based buffer overflow exploitation prevention mechanism of Microsoft windows 2003 server. In Black Hat Asia, 2003.
- [6] Solar Designer. Return-to-libc attack. Bugtraq, 1997.
- [7] H. Shacham. The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86). In ACM Conf. on Computer and Communications Security, 2007.
- [8] G. F. Roglia, L. Martignoni, R. Paleari, and D. Bruschi. Surgically returning to randomized lib(c). In Proceedings of the Annual Computer Security Applications Conference, pages 60–69, 2009.
- [9] —, "Address space layout randomization." [Online]. Available: <http://pax.grsecurity.net/docs/aslr.txt>
- [10] T. Bletsch, X. Jiang, V. W. Freeh, and Z. Liang. Jump-oriented programming: a new class of code reuse attack. In ACM Symp. on Info., Computer and Communications Security, 2011.
- [11] F. J. Serna. The info leak era on software exploitation. In Black Hat USA, 2012.
- [12] Nergal. The advanced return-into-lib(c) exploits: PaX case study. Phrack Magazine, 58(4), 2001.
- [13] S. Checkoway, L. Davi, A. Dmitrienko, A.-R. Sadeghi, H. Shacham, and M. Winandy. Return-oriented programming without returns. In ACM Conf. on Computer and Communications Security, 2010.
- [14] D. D. Zovi. Practical return-oriented programming. Invited Talk, RSA Conference, 2010.
- [15] L. Liu, J. Han, D. Gao, J. Jing, and D. Zha. Launching return-oriented programming attacks against randomized relocatable executables. In IEEE International Conference on Trust, Security and Privacy in Computing and Communications, 2011.
- [16] A. Sotirov and M. Dowd. Bypassing browser memory protections in Windows Vista, 2008.
- [17] H. Shacham, E. jin Goh, N. Modadugu, B. Pfaff, and D. Boneh. On the effectiveness of address space randomization. In ACM Conf. on Computer and Communications Security, 2004.
- [18] A. Gupta, S. Kerr, M. Kirkpatrick, and E. Bertino. Marlin: A fine grained randomization approach to defend against rop attacks. In J. Lopez, X. Huang, and R. Sandhu, editors, Network and System Security, volume 7873 of Lecture Notes in Computer Science, pages 293–306. Springer Berlin Heidelberg, 2013.
- [19] R. Wartell, V. Mohan, K. W. Hamlen, and Z. Lin. Binary stirring: Self-randomizing instruction addresses of legacy x86 binary code. In ACM Conf. on Computer and Communications Security, 2012.
- [20] V. Pappas, M. Polychronakis, and A. D. Keromytis. Smashing the gadgets: Hindering return-oriented programming using in-place code randomization. In IEEE Symposium on Security and Privacy, 2012.
- [21] C. Giuffrida, A. Kuijsten, and A. S. Tanenbaum. Enhanced operating system security through efficient and fine-grained address space randomization. In USENIX Security Symposium, 2012.
- [22] S.Krahmer. x86-64 buffer overflow exploits and the borrowed code chunks exploitation technique. <http://users.suse.com/~krahmer/no-nx.pdf>, 2005.
- [23] J. D. Hiser, A. Nguyen-Tuong, M. Co, M. Hall, and J. W. Davidson. ILR: Where' d my gadgets go? In IEEE Symposium on Security and Privacy, 2012.
- [24] Hiser, J., Nguyen-Tuong, A., Co, M., Hall, M., Davidson, J.W.: Ilr: Where' d my gadgets go? In: Proc. of the 2012 IEEE Symposium on Security and Privacy, pp. 571–585 (2012)
- [25] Onarlioglu, K., Bilge, L., Lanzi, A., Balzarotti, D., Kirda, E.: G-free: defeating return-oriented programming through gadget-less binaries. In: Proc. of the 26th Annual Computer Security Applications Conference, pp. 49–58 (2010)
- [26] Davi, L., Dmitrienko, A., Nurnberger, S., Sadeghi, A.R.: Xifer: A software diversity tool against code-reuse attacks. In: 4th ACM International Workshop on Wireless of the Students, by the Students, for the Students, S3 2012 (August 2012)
- [27] —, "BSTR." [Online]. Available: [http://msdn.microsoft.com/en-us/library/windows/desktop/ms221069\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/ms221069(v=vs.85).aspx)
- [28] E. J. Schwartz, T. Avgerinos, and D. Brumley. Q: exploit hardening made easy. In USENIX Security Symposium, 2011.
- [29] BLETSCH, T., JIANG, X., FREEH, V. W., AND LIANG, Z. Jump-oriented programming: a new class of code-reuse attack. In ASIACCS'11: Proc. 6th ACM Symp. on Information, Computer and Communications Security (2011), pp. 30-40.
- [30] Snow, Kevin Z., et al. "Just-in-time code reuse: On the effectiveness of fine-grained address space layout randomization." *Security and Privacy (SP), 2013 IEEE Symposium on.* IEEE, 2013.

# Industry-wise Analysis of Security Breaches in Data Loss Incidents

Rehab El-Kharboutly  
School of Business & Engr.  
Quinnipiac Univ.  
Hamden, CT 06518  
Ruby.ElKharboutly@quinnipiac.edu

Swapna S. Gokhale  
Dept. of CSE  
Univ. of Connecticut  
Storrs, CT 06269  
ssg@engr.uconn.edu

Lance Fiondella  
Dept. of ECE  
Univ. of Massachusetts  
Dartmouth, MA 02747  
lfiondella@umassd.edu

## Abstract

Cloud computing offers many conveniences at reduced costs, which has motivated many organizations to migrate their data and services to cloud platforms. Security breaches that plague these cloud transactions, however, can threaten widespread migration to the cloud. In this paper, we study data loss incidents over the last decade to discover the vulnerabilities that organizations face when moving their data and computations to the cloud. These data loss incidents are collected and documented in the DataLoss project led by the Open Source Foundation. Our industry-wise analysis focuses on understanding the types of breaches and data loss modes encountered by organizations in the business, educational, government, and medical sectors. For each industrial sector, we also measure the impact of these breaches in terms of the number of lost records and the estimated costs. We summarize the findings of our analysis into a set of recommendations for the four sectors. We expect that these recommendations will guide organizations in developing policies and practices that can prevent security breaches and stem the resulting losses.

## 1 Introduction

With the advent of cloud computing, many organizations have begun to store their data and applications in the cloud. A majority of these cloud transactions involve sensitive individual information including but not limited to name, social security number, credit card details, and health and medical data. Regardless of the involvement of such sensitive information, migration to the cloud exposes these organizations and their transactions to security breaches potentially resulting in a loss of valuable personal data of their clients. Such data loss can, at the very least, harm the reputation of these organizations, lead to financial consequences, and in the worst case invite potentially damaging lawsuits. A 2005 survey by the Federal Bureau of Invest-

igation [1] approximated the annual loss due to computer crimes at 67.2 billion dollars whereas a more recent study estimates this cost at 112 billion [2]. These two surveys clearly indicate the magnitude of the problem posed by data loss incidents. Instances of high-profile security breaches and resulting losses of sensitive data abound. For example, a recent security breach experienced by the South Carolina Department of Revenue exposed millions of Social Security numbers, hundreds of thousands of credit and debit card numbers and business tax records.

Organizations must therefore make an informed decision about migrating their data and transactions to the cloud by building an extensive knowledge of the types and likelihoods of the breaches they might face and the types and magnitude of data losses that these breaches might precipitate. The thousands of data loss incidents that have already occurred over the past decade may offer important clues into building this knowledge repository. Lessons drawn from these historical incidents can guide these organizations in the development of appropriate data handling and protection procedures. These lessons may also help cloud subscribers and providers to avoid future data loss incidents and mitigate the impact of such incidents should they occur.

In this paper, we analyze historical data loss incidents, documented in the DataLoss project [3], which is led by the Open Security Foundation (OSF) [4] with the goal of creating a database of physical and electronic data breaches experienced worldwide. Our analysis searches for the common types of security breaches and seeks to understand if particular data states or loss modes are specifically likely. Finally, we seek to find the impact of the breaches and loss modes in terms of the number of records and estimated costs. Along each of these dimensions, we explore and compare industry-specific trends for the business, educational, government and medical sectors. Our lessons indicate that: (i) the business sector is most vulnerable to databases and data storage hacks; (ii) the educational and government sectors are commonly victims of stolen user de-

vices; and (iii) the medical sector evenly experiences hacks, stolen devices, and data interceptions over the network.

The rest of the paper is organized as follows: Section 2 provides an overview of the DataLoss project. Section 3 analyzes the breach types. Section 4 analyzes the loss modes. Section 5 analyzes the impact of the breach types and loss modes. Section 6 summarizes our findings into a set of recommendations for each sector. Section 7 compares related work. Section 8 concludes the paper and identifies directions for future research.

## 2 DataLoss Project

The DataLoss project was established with several objectives [3]. Curators of the data wanted to raise awareness of the threats posed by data security and identity thefts to improve consumer protection. They believed that organized, unbiased public information would assist corporations and government organizations in making decisions and planning consumer protection efforts. They also wanted to provide citizens with objective data to evaluate the effectiveness of laws that require disclosure of breaches.

Data about breaches is collected by searching through news feeds, blogs, and other websites for data loss incidents. The incidents so compiled are either emailed to project members or posted on Twitter. The project's administrators also request breach notification documents from various states through the Freedom of Information Act (FOIA) [5]. Significant information about each incident is recorded, the details can be obtained from elsewhere [11]. We summarize the fields relevant for our study here.

- **Number of Records:** The number of records lost or affected by the breach.
- **Industry/Sector:** Primary organization impacted, from one of the following: business (Bus), education (Edu), government (Gov), medical (Med).
- **Incident Classification:** This classification details the method of breach and type of media compromised. We use the following framework to describe these categories,  $x \in \mathbf{X} = \{\text{Computer, Document, Drive, Laptop, Media, Mobile, Tape}\}$ . For example:
  - **Disposal**  $x$ : Data on  $x \in \mathbf{X}$  was not properly sanitized prior to disposal.
  - **Lost**  $x$ :  $x \in \mathbf{X}$  was lost with data still on it.
  - **Missing**  $x$ :  $x \in \mathbf{X}$  missing, but unknown or disputed whether lost or stolen.
  - **Stolen**  $x$ :  $x \in \mathbf{X}$  either reported or known to have been stolen by a third party.

- **Email/Fax/Snail Mail:** Email, fax, or snail mail exposed to unintended third party.
  - **Fraud/SE (Social Engineering):** A fraud or scam, often insider-related.
  - **Hack:** Computer-based intrusion, where data may or may not be exposed to public access.
  - **Skimming:** Theft of credit or bank card information with a small undetected electronic device (skimmer) used to steal the credit or bank card information of a victim when swiped.
  - **Snooping:** An employee exceeds privileges to access confidential records they were unauthorized to view.
  - **Unknown:** Unknown or unreported breach type.
  - **Virus:** Exposure of personal information from a virus or trojan such as a keystroke logger. Several incidents belonging to this class may also be classified as hacks.
  - **Web:** A computer or web-based intrusion performed with data commonly available to the public such as search engines and public pages.
- **Estimated Costs:** Estimated cost based on Ponemon Institute Direct Costs Estimate [6].

## 3 Analysis of Breach Types

Table 1 summarizes the different types of breaches, further classified according to industrial sectors. The table shows that the business sector has experienced three times as many breaches as compared to the educational, medical, and government sectors. Hacks is the most frequent type of breach experienced by the business and medical sectors; this highlights the vulnerability of sensitive personal, financial, and health-related data stored in electronic form. On the other hand, stolen laptop is the most frequent breach experienced by the educational sector and stolen drive is the most common breach in the government sector, suggesting that these two sectors are plagued more by physical rather than cyber breaches. For the business sector, Fraud and Social Engineering comes second, followed by Stolen Laptop. Hacks is the second most common form of breach for the educational sector, however, a particularly unsettling finding is that a significant percentage of breaches in this sector remain unclassified. The government and educational sectors rank stolen laptop and web as the second most frequent types of breaches, followed by disposal document.

## 4 Analysis of Loss Modes

The digital data in enterprises generally exists in three states, which correspond to three loss modes [7]. The first

**Table 1. Breach Types Per Industrial Sector**

Breach Type	Industry			
	Bus	Edu	Gov	Med
Disposal computer	7 (0.22%)	5 (0.45%)	5 (0.45%)	3(0.32%)
Disposal document	231 (7.24%)	81 (7.32%)	81 (7.27%)	38(4.10%)
Disposal drive	4 (0.12%)	5 (0.45%)	5 (0.44%)	1(0.11%)
Email	90 (2.81%)	43 (3.89%)	43 (3.86%)	53(5.71%)
Fax	0 (0.00%)	2 (0.18%)	2 (0.18%)	0(0.00%)
Fraud/SE	509 (15.88%)	92 (8.32%)	92 (8.26%)	35 (3.77%)
Hack	898 (28.02%)	117 (10.58%)	123 (11.04%)	261(28.13%)
Lost computer	7 (0.22%)	45 (4.07%)	45 (4.04%)	0(0.00%)
Lost document	54 (1.69%)	36 (3.26%)	36 (3.23%)	14(1.51%)
Lost drive	27 (0.84%)	12 (1.09%)	12 (1.08%)	19(2.05%)
Lost laptop	20 (0.62%)	53 (4.79%)	53 (4.76%)	1(0.11%)
Lost media	43 (1.34%)	18 (1.63%)	0 (0.00%)	12(1.29%)
Lost tape	48 (1.50%)	7 (0.63%)	18 (1.62%)	2(0.22%)
Missing document	8 (0.25%)	6 (0.54%)	7 (0.63%)	1(0.11%)
Missing laptop	4 (0.13%)	0 (0.00%)	0 (0.00%)	0(0.00%)
Missing media	6 (0.19%)	75 (6.78%)	6 (0.54%)	0(0.00%)
Snail mail	134 (4.18%)	44 (4.00%)	76 (6.82%)	29(3.13%)
Stolen computer	125 (3.90%)	48 (4.34%)	44 (3.95%)	54(5.82%)
Stolen document	68 (2.12%)	17 (1.54%)	49 (4.40%)	24(2.59%)
Stolen drive	32 (1.00%)	156 (14.10%)	17 (1.53%)	16(1.72%)
Stolen laptop	416 (13.00%)	12 (1.08%)	156 (14.00%)	110(11.85%)
Stolen media	14 (0.44%)	4 (0.36%)	12 (1.08%)	11(1.19%)
Stolen mobile	2 (0.06%)	58 (5.24%)	0 (0.00%)	0(0.00%)
Stolen tape	26 (0.81%)	13 (1.18%)	4 (0.36%)	2(0.22%)
Unknown	152 (4.74%)	155 (14.01%)	58 (5.2%)	14(1.51%)
Virus	42 (1.31%)	2 (0.18%)	13 (1.17%)	30(3.23%)
Web	237 (7.39%)	0 (0.00%)	157 (14.09%)	198(21.34%)

category labeled “Data in Motion” includes data that moves through the network to the outside world via email, instant messaging, peer-to-peer (P2P), FTP, or other communication mechanisms. The second category labeled “Data at Rest” includes data residing in file systems, distributed desktops and large centralized data stores, databases, or other storage centers. Finally, the third category labeled “Data in Endpoints” includes data residing at network endpoints such as laptops, USB devices, external drives, CD/DVDs, archived tapes, MP3 players, iPhones, or other highly mobile devices that are under users’ control. According to these definitions, we partitioned the incidents that involve electronic data into three modes as shown in Table 2. The remaining types of breaches did not involve electronic data, and hence, were excluded from the classification.

Table 3 shows the classification of incidents according to these three modes for each industrial sector. In the business sector, data residing in central storage is breached most commonly. This observation, combined with the findings from Section 3 suggest that the business sector is most plagued by database or data storage hacks. For the educa-

**Table 2. Breach Types Per Loss Mode**

Data in Motion	Email, Web Snail mail
Data at Rest	Hack, Virus
Data in Endpoints	Disposal computer Lost drive Lost computer Lost laptop Stolen computer Stolen drive Stolen laptop Stolen mobile

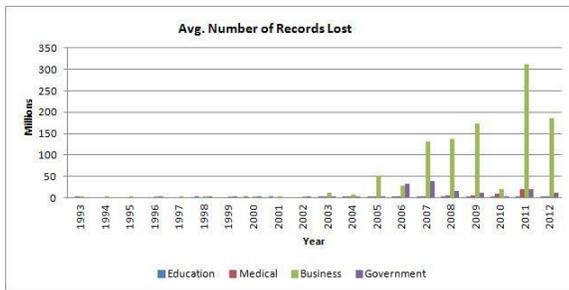
tional and government sectors, it is the data in the endpoints that is compromised most frequently. The common ground in both the loss modes and types of breaches between the educational and government sectors could be attributed to the lack of resources to ensure physical security allowing easy access to portable devices. Finally, data in the medical sector is breached almost evenly across all the three modes.

**Table 3. Loss Modes per Industrial Sector**

Mode	Industrial Sector			
	Bus	Edu	Gov	Med
Data in Transit	461 (22.46%)	87 (11.68%)	276 (36.51%)	280 (36.18%)
Data in Storage	940 (45.79%)	119 (15.97%)	136 (17.99%)	291 (37.60%)
Data in Endpoints	652 (31.76%)	539 (72.35%)	344 (45.50%)	203 (26.22%)

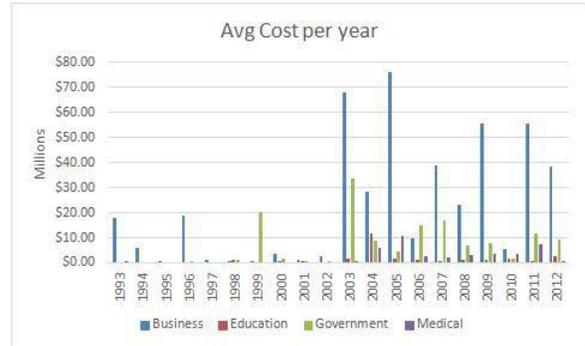
## 5 Analysis of Impact

Next, we analyze the impact of these incidents, measured in terms of the number of records lost and the estimated costs. Figure 1, which shows year-wise trends in the number of lost records, indicates that prior to year 2002, the business sector lost a modest number of records every year, the medical and educational sectors showed no loss of data, and the government sector recorded only sporadic losses. These trends could arise because while many businesses did have an online presence before 2002, educational and medical organizations have moved online only within the last decade. Overall, the number of records lost in all the sectors grew significantly after 2002, which could be attributed to the rapid migration of many organizations to the cloud. Figure 2 shows the year-wise estimated costs for each sector, which follow similar trends as the number of lost records shown in Figure 1.



**Figure 1. Sector-wise Yearly Avg. # Records Lost**

Table 4 shows the average number of records lost and average estimated cost per incident across the four sectors. The average number of records lost is the highest in the business sector, which is an order of magnitude higher than the educational and government sectors, which in turn are higher by an order of magnitude than the government sectors. The average estimated cost follows a different trend. Starting with the largest value for the business sector, the government sector ranks next followed by the medical and educational sectors. The business sector ranks at the top both in terms of the number of lost records and the estimated costs possibly because businesses are not fined for data loss and lobby vigorously against such penalties.



**Figure 2. Sector-wise Yearly Avg. Est. Cost**

**Table 4. Summary of Impact**

Sector	Average	
	Num. of Records	Est. Cost
Bus	611710	\$36,702,654
Edu	24341	\$1460518
Gov	698	\$11,330,302
Med	64263	\$3,855,785

## 6 Summary and Recommendations

In this section, we summarize the findings of our analysis into a set of recommendations for each industrial sector.

- **Business sector:** Hacks of databases or data storage is the most common type of breach in this sector. The number of records lost as a result of the breaches is two to three orders of magnitude higher than the other three sectors. Thus, the business sector can benefit significantly from prevention strategies for data in storage.
- **Educational and Government sectors:** Physical thefts of user devices such as a laptops and drives is most prevalent in these two sectors. Correspondingly, data in the endpoints is the most commonly compromised. Therefore, securing these endpoints by limiting access can offer a good pay off in alleviating the loss of data in these two sectors.
- **Medical sector:** This sector shows a uniform vulnerability to breaches such as hacks, physical violations caused by the thefts of devices, and interception of data while it is in transit. Thus, data in all three modes is

vulnerable, calling for uniform investments in preventing hacks on databases, stolen devices, and encryption of data as it passes over the network. Moreover, a deeper understanding of the significant percentage of breaches that remain unclassified is necessary in order to avoid them. Preventing the loss of medical data is especially important because this loss usually causes permanent damage by compromising users' privacy. As a result, this loss can potentially expose organizations to regulatory and litigation threats [7].

## 7 Related Research

Cloud computing paradigms have given rise to new security challenges and there is an ever growing body of literature on methods for ensuring privacy and security in the cloud. For example, Yang and Jia [8] developed an auditing framework for cloud storage systems and an accompanying privacy-preserving auditing protocol to reassure data owners that their data are correctly stored in the cloud. Seo *et al.* [9] propose a mediated certificateless encryption (mCL-PKE) scheme without pairing operations to improve efficiency while preserving confidentiality of the content and the keys with respect to the cloud. Ni *et al.* [10] demonstrated a flaw in the recently published dynamic auditing protocol of Yang and Jia [8] and suggested a revision, which preserves the properties of the original protocol.

While advanced techniques such as the above continue to evolve and mature, significantly less attention has been dedicated to broader studies to understand the computing industry's historical track record on data loss. Such analysis can offer valuable lessons to policy makers of public and private organizations to better defend themselves as well as their customers and clients from data theft. Our own prior work has taken a step in this direction by analyzing the incidents in the DataLoss project to explore data security, recovery, arrest, and lawsuit statistics in an effort to identify practices that could improve the chances of data recovery [11]. We extend this prior work with a focus on devising policies to prevent the occurrence of such incidents by understanding the likelihood of different types of breaches, modes of data loss, and their impact in terms of the number of lost records and estimated costs for each sector.

## 8 Conclusions and Future Research

This paper analyzed the incidents recorded in the DataLoss project to find common breaches, loss modes and their impact on organizations in the business, educational, government, and medical sectors. A set of recommendations based on our findings were offered for each sector.

Our future work will focus on mining the reports associated with the incidents to gain further insights into the de-

velopment of data protection policies and procedures. We will also study the correlation between data loss disclosures with short- and long-term loss of stock value. In the absence of legislation that holds organizations accountable for data loss, this analysis may motivate organizations to invest in the prevention of data loss incidents out of self interest, to protect their finances and reputation.

## References

- [1] D. Powner and K. Rhodes. Public and private entities face challenges in addressing cyber threats. Report to Congressional Requesters GAO-07-705, Government Accountability Office, <http://www.gao.gov/products/GAO-07-705>, jun 2007.
- [2] W. Jones. "How much does cybercrime cost? 113 billion". *IEEE Spectrum*, November 2013.
- [3] Open Security Foundation. DataLossDB. <http://datalossdb.org/>, Last accessed November 26, 2013.
- [4] Open Security Foundation. Open security foundation. <http://www.opensecurityfoundation.org/>, Last accessed November 26, 2013.
- [5] U.S. Dept. of Justice. Freedom of information act. <http://www.foia.gov/>. viewed on March 6, 2014.
- [6] 2011 Cost of Data Breach Study: United States. Technical report, Ponemon Institute LLC, Traverse City, MI, mar 2012.
- [7] S. Liu and R. Kuhn. "Data loss prevention". *IT Professional*, 12(2):10–13, March/April 2010.
- [8] K. Yang and X. Jia. "An efficient and secure dynamic auditing protocol for data storage in cloud computing". *IEEE Trans. on Dependable and Secure Computing*, 24(9):1717–1726, Sept. 2013.
- [9] S. Seo, M. Nabeel, X. Ding, and E. Bertino. "An efficient certificateless encryption for secure data sharing in public clouds". *IEEE Trans. on Knowledge and Data Engineering*, Aug. 2013.
- [10] J. Ni, Y. Yu, Y. Mu, and Q. Xia. "On the security of an efficient dynamic auditing protocol in cloud storage". *IEEE Trans. on Parallel and Distributed Systems*, Nov. 2013.
- [11] L. Fiondella, R. El Kharboutly, and S. Gokhale. "Data loss: An empirical analysis in search of best practices for prevention". In *Proc. of Intl. Workshop on Cloud Analytics, co-located with Intl. Conf. on Cloud Engineering*, March 2014 (To Appear).

# Bug Inducing Analysis to Prevent Fault Prone Bug Fixes

Haoyu Yang, Chen Wang, Qingkai Shi, Yang Feng, Zhenyu Chen\*

State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China

\*Corresponding author: zychen@software.nju.edu.cn

## Abstract

*Bug fix is an important and challenging task in software development and maintenance. Bug fix is also a dangerous change, because it might induce new bugs. It is difficult to decide whether a bug fix is safe in practice. In this paper, we conducted an empirical study on bug inducing analysis to discover the types and features of fault prone bug fixes. We use a classical algorithm to track the location of the code changes introducing the bugs. The change types of the codes will be checked by an automatic tool and whether this change is a bug fix change is recorded. We analyze the statistics to find out what types of change are most prone to induce new bugs when they are intended to fix a bug. Finally, some guidelines are provided to help developers prevent such fault prone bug fixes.*

**Keywords:** bug inducing, bug fix, mining software repository, software maintenance

## 1 Introduction

Bug fix is an important and challenging task for industrial and open source software. Developers may change some code in some specific files to fix bugs. These changes could be various, such as, changing certain lines in an existing function, changing the accessibility of a function from private to public or reorder the parameters of a function, etc. These modifications may bring potential danger into software. They may cause exceptional behavior later. In order to record and fix bugs well, software configuration management system and bug tracking system are used to control the process of constructing software and the flow of bugs respectively. software configuration management could cooperate with bug tracking tools and they indicate which commit fixes a specific bug in the bug tracking system.

It is valuable to automatically identify commits that may induce bugs. It enables developers to quickly and efficiently validate many types of bug fixes. However, it is challenging to find the bug inducing changes accurately. SZZ is introduced to automatically identify bug-introducing com-

mits [11]. This algorithm is improved to provide a process for automatically identifying the bug inducing predecessor lines that are changed in a bug-fixing commit [6]. The procedure of the SZZ algorithm firstly identifies the lines modified or deleted in a bug-fixing commit. It is easy to mine a software repository to find the commit that fixes a bug. We can finish it through examining the log message in each commit and check whether it has the keywords “Bug” and “Fixed” [8] or has symbols of bug reports like “#31245” [2] [4] [11], and then identify the bug-inducing change using annotation graph.

We adopt and improve the SZZ algorithm for our empirical study. We classify bug inducing changes into bug-fix ones or non-bug-fix ones. We analyse the change types of bug inducing changes to investigate the laws in the phenomena. We can figure out that what kinds of change types to fix a bug are more easily to bring in new bugs later.

The main contributions of this paper are as follows.

- We compare the proportion of change types between bug-inducing change which is bug-fix and bug-inducing change which is not bug-fix to figure out what kinds of change types are more dangerous in bug-fix change.
- We do not only focus on the common changes, but also study special changes appeared in object-oriented programs.
- Ratio of bug-fix bug-inducing changes is counted so that we could know what percentage of bugs are induced by bug fixes.

The rest of paper is organized as follows. Section 2 describes bug inducing analysis method in our study. The experiment design and the result analysis are presented in Section 3. We discuss the practical guidelines and related work in Section 4. The last section is the conclusion and future work.

## 2 Bug Inducing Analysis

### 2.1 Change and Bug Fix

Bug fix commits could be identified by examining the commit log messages that whether it includes the keywords like “Bug”, “Fixed” or has a specific bug report number corresponding to bug tracking system. Deleted changes and modified changes in bug fix commits are assumed as bug fix changes. Our experiment will use this method to find both bug fix changes and bug inducing changes.

Object-oriented programming languages are widely used in software development and they also present many challenges for software maintenance. There are some useful but risky characteristics, which are difficult to handle since some code changes may cause unexpected and non-local effects [10]. In change impact analysis, a key aspect is to transform source code edits into a set of atomic changes [10]. Some change types come from classical object-oriented change analysis, e.g., **lookupChange** [10] and changes related to variable initializers [9], which are checked manually. Others are distinguished by automatic tool. We combine these change types together, all change types are shown in Table 1. The second column shows that whether we check them manually or not.

### 2.2 Bug Inducing

The situation that developers do a lot of bad changes just to fix one tough bug occurs frequently especially when time is limited. Some complicated bugs are hard to fix. Developers may use special ways, which may not be consistent with the original design patterns involved. Obviously these kinds of change types are dangerous. As a consequence, more bugs will be induced and the whole software gets into chaos gradually. In this study, we trace a bug-inducing change and identify whether the change is a bug fix change. These changes are classified into two types as follows.

- **Bug Inducing Change (BIC):** A change that induced a bug and is a bug-fix change itself.
- **Non Bug Inducing Change (NBIC):** A change that induced a bug and is not a bug-fix change itself.

NBIC may be the Addition of new features, enhancement or other changes which are not aimed to fix a bug.

### 2.3 Fault Prone Analysis

The SZZ algorithm uses CVS annotation feature to trace bug-inducing changes. Developers could match a line

Table 1. Atomic Change Type Checked

Type Name	Manually
methodAdded	N
codeChanged	N
codeAdded	N
typeDeclarationAdded	N
innerClassAdded	N
fieldAdded	N
importAdded	N
parameterAdded	N
returnTypeChanged	N
accessChanged	N
constructorAdded	N
throwsAdded	N
parameterTypeChanged	N
variableChanged	N
importSectionAdded	N
parameterNameChanged	N
parameterReordered	N
methodBlockAdded	N
accessAdded	N
modifierChanged	N
lookupChanged	Y
instanceFieldInitializerChanged	Y
staticFieldInitializerChanged	Y
instanceInitializerAdded	Y
emptyInstanceInitializerDeleted	Y
instanceInitializerChanged	Y
staticInitializerAdded	Y
staticInitializerDeleted	Y
staticInitializerChanged	Y

which is changed in bug fix changes to its most recent modification so that they finally find where the bug inducing is. However, there may exist some biases judging bug inducing just by the most recent modifications since those modifications are likely to be non-behavior changes, e.g., format changes. The improvement of SZZ [6] removes biases caused by non-behavior changes in order to get a higher accuracy. And annotation feature is replaced with annotation graph at the same time, which is more precise. We use the improvement of SZZ in our empirical study.

SZZ algorithm is used to locate the bug inducing changes based on bug fix changes. When the commit has been located, we use the same method to judge whether the located change is a BIC or NBIC. This step does matter since when we find out that the change is BIC and we can analyze its change types. Some types of changes may be dangerous in a degree. We call them fault prone bug fix changes. Counting change types does let the developers know what kinds of bug fix change types are fault prone while others are

**Table 2. Project Information**

Project Name	Source	Lineofcode	NumofVersions	Brief Introduction
JEdit	SourceForge	186326	23520	lineJedit is a programmer’s text editor written in Java Protostuff is the stuff that leverages google’s protobuf. Machine learning framework
Protostuff	Googlecode	122277	1677	
Encog	Github	122723	3241	

relatively safe. Hence developers could fix a bug in the right way since this way prevents another bug from being induced.

### 3 Empirical Study

We conducted three detailed case studies for further investigate what kinds of change types are more dangerous in BIC than NBIC. These three projects are open source projects implemented by java, detail information of them is shown in the Table 2. As for JEdit<sup>1</sup>, there are too many revisions for us to examine all of them, so we choose to select revisions randomly. For Protostuff<sup>2</sup> and Encog<sup>3</sup>, we investigate the whole revisions of them.

#### 3.1 Experiment Procedure

Bug-fix commits are identified by their log message. Deleted and modified changes are being tracked backward using annotation graph so that we can find out where the corresponding bug-inducing change is. Then we judge whether the bug-inducing change is a BIC with the same method. Some bugs are induced in ordinary commit or initial import. They are caused by NBIC, while others are caused by BIC, which we count for percentage. After we locate the bug-inducing change, we use a tool *diffJ*<sup>4</sup> to judge what atomic change types it includes. However, some atomic change types from object-oriented perspectives cannot be distinguished by automatic tool. So we manually check them.

#### 3.2 Result Analysis

The statistics are shown in Table 3. The data tell us that in each project, there exist some BIC which lead to more bugs later.

Then we separate BIC from NBIC, comparing the differences between the proportion of each atomic change type in BIC and NBIC respectively. The results are shown in Figure 1, 2 and 3 in descending order. It could be concluded that

<sup>1</sup><http://sourceforge.net/projects/jedit/>

<sup>2</sup><https://code.google.com/p/protostuff/>

<sup>3</sup><https://github.com/encog/encog-java-core>

<sup>4</sup><https://github.com/jpace/diffj>

**Table 3. Overall Experimental Results**

Project Name	NBIC(#)	BIC(#)	BIC(%)
JEdit	466	144	23.61%
Protostuff	180	71	28.29%
Encog	298	39	11.57%

**codeAdded** and **codeChanged** appeared in BIC are much more than NBIC, which means that these change types are more prone to cause other bugs later if they appeared in BIC.

To discover more detailed information, we conduct further investigation on the atomic change types **codeAdded** and **codeChanged** since they appear more in BIC than NBIC. In that case, we manually check the BIC which includes atomic change type **codeAdded** or **codeChanged** in order to conclude, in which context these change types will show up. For example, adding or changing an *if* conditional statement, adding or modifying the parameters of the method, removing or changing the value of some local variables. Our statistics are shown in Table 4. We list the percentage of all the change types they related to. Some minor changes are combined together, for example, adding and modifying the parameters of method are combined to parameters. The total number of **codeAdded** and **codeChanged**, method invoke changes and return value are listed as *total,method* and *return* respectively.

#### 3.3 Threat to Validity

One threat to external validity is mainly due to the quality of open source project. Well managed project will do good to our research, while bad one will make biases. As for our experiment, the project *encog* initially imports 605 files together into repository, which leads to a lot of bugs tracked backward to the first revision. Obviously the first version is not a bug-fix commit. As a result, percentage of BIC is relatively small compared to the other projects.

Furthermore, we investigate only three projects and all of them are open source projects, which may not be enough to support our conclusion. Our experiment results may not be representative, because there may exist great difference in design between open source software and industrial software.

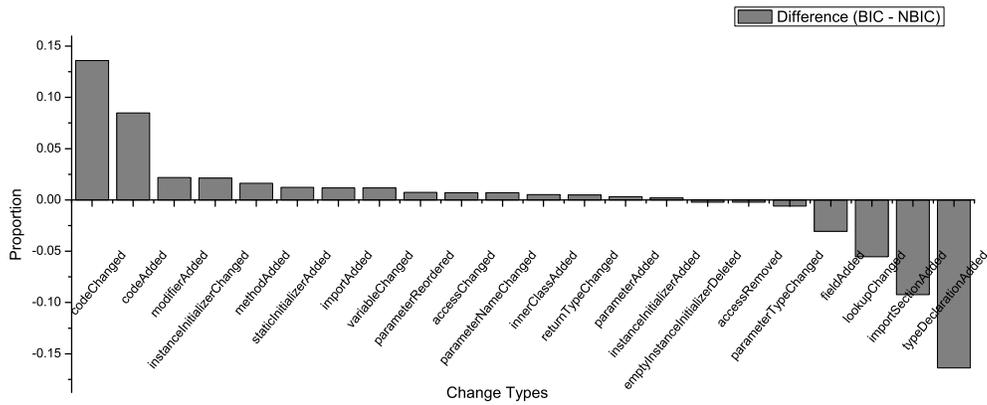


Figure 1. Proportion Difference of Each Change Type between BIC and NBIC in JEdit

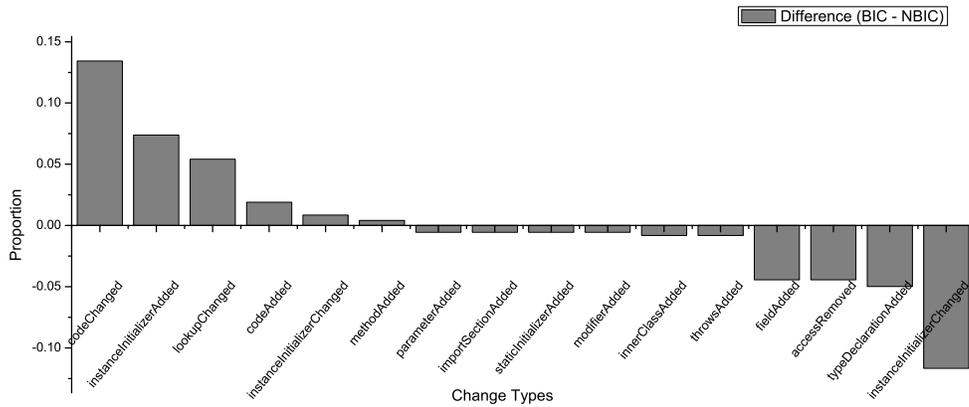


Figure 2. Proportion Difference of Each Change Type between BIC and NBIC in Protostuff

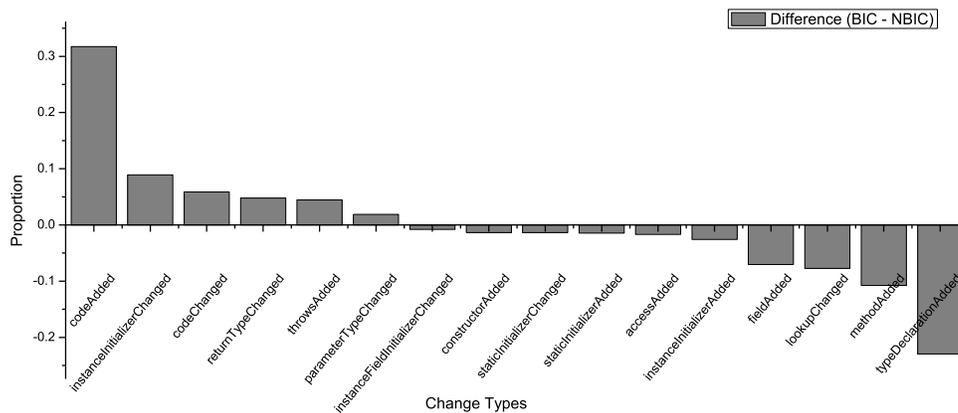


Figure 3. Proportion Difference of Each Change Type between BIC and NBIC in Encog

**Table 4. Detail statistics about codeAdded and codeChanged changes**

Project	total	if	method	local variable	return	type cast	parameters	file path	loop	exception
JEdit	68	42.6%	8.8%	23.5%	2.9%	1.5%	7.4%	1.5%	4.4%	2.9%
Protostuff	26	69.2%	15.4%	3.8%	3.8%	0	7.7%	0	0	0
Encog	21	38.1%	23.8%	9.5%	0	0	19%	0	9.5%	0
Overall	115	47.8%	13.4%	16.5%	2.6%	0.86%	9.5%	0.86%	4.3%	1.7%

Another threat is that all the three projects conducted are implemented by Java. Although Java is a representative object-oriented language, other languages may include some special features which java language does not include, and we do not take them into consideration.

We manually check some kinds of change types in object-oriented programs. However, manual checks may include errors.

## 4 Discussion

In this section, some interesting findings from our statistics will be discussed, in order to provide some guidelines in the software development and debugging practice.

### 4.1 If-Else Clauses Are Dangerous

As shown in the Table 4, changes related to *if* conditional statement occupy a large proportion of both **codeChanged** and **codeAdded**.

These changes include adding some new if-else blocks between the lines of a method, making some modifications of the conditional statements and applying combination or separation on the original statements. According to our statistics, the proportion of these changes on *if* conditional statements accounts for nearly 50% in both **codeChanged** and **codeAdded** changes, while other kinds of change types are in minority, which reveals that it should not be conceived an ignorable issue during the process of bug-fix. All of these changes on *if* conditional statements cast great danger on the lines just added or modified. Chances are that the conditional statements run normally under current circumstances, however, the coverage of the condition may not be integrate.

As the percentage of changes related to *if* conditional statements is so high in the bug-prone changes in our experiment, we sincerely suggest the programmers to avoid these kinds of changes in their software development practice. One effective solution is to apply widely recognized software design patterns and strict object-oriented rules on the program during the beginning period of the project in order to decrease the modifications on initial codes of a class or method. For example, using strategy pattern would

decrease the number of *if* conditional statements in one method. Thus, the complexity of the business logic of a class and the possibility of changing the conditional statements to fix a bug will be reduced.

### 4.2 Open/Closed Principle

From the statistics, we can see that the percentage of **typeDeclarationAdded** in BIC is much lower than N-BIC. Thus, we can make the assumption that **typeDeclarationAdded** may be a much safer way to fix a bug. There is a famous principle in object-oriented software development supporting our idea, and that is the so-called **Open/Closed Principle**.

The **Open/Closed Principle** states that "*software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification*". [7]. That means if we want to add some new functions or fix the existed bugs, such entities should allow these behaviours without altering its source code. If a project is constructed under the **Open/Closed Principle** strictly, there will be an increasing trend in BIC which can be completed by declaring new types or extending the current interface but not modifying them. As a result, changes like **codeChanged** or **codeAdded** which are prone to induce new bugs in BIC will be greatly prevented.

### 4.3 Related Work

As concluded in the work of Mockus [8], Cubranic [2] and Fischer [4], commit in software configuration management system can be judged whether it is a bug-fix one by the log message. Some key words, e.g., "Bug" or "Fixed" even bug report number "#31245" can serve as symbols of a bug-fix commit. Based on their work, Sliwerski et al. [11] presented SZZ algorithm to find out the position of bug-inducing changes according to the position of bug-fix changes. Kim et al. [6] improved SZZ algorithm by using annotation graph and ignore the changes which is meaningless just like blank line changes and format changes so that it became much more precise. The improvement of SZZ is currently the best available algorithm for automatically identifying bug-inducing commits. The algorithm does great contributions to related researches. Kim et al. [5] evaluate

whether a change is risky by using SZZ to track the original change. D’Ambros [3] visually reveals the relationship between bugs and software evolution. Williams et al. [13] revisited SZZ algorithm, using SZZ algorithm to track bug-inducing changes and identify change types of them, which is similar to our work. Ryder et al. [10] analyse the impact of object-oriented changes. Ren [9] does the similar work as Ryder, which defined some object-oriented change types that we use in our paper. Spacco [12] used lining mapping which make the tracking of bug inducing fix more precisely. A differencing technique and tool for object-oriented programs: JDiff [1] provide us the tool support. [12]

## 5 Conclusion and Future Work

In this paper, we apply the SZZ algorithm on a series of projects to find out what kinds of bug-inducing changes exist higher possibilities to become a great threat when they are marked as bug-fix changes. Based on the empirical studies operated on three open source projects, our statistics show that the **codeAdded** and **codeChanged** changes, especially addition or modification of the *if* conditional statements in bug-fix commits are apt to cause more bugs in the future.

Our future work may include the following aspects: A much wider selection of projects are going to be researched to further validate the conclusion we drew. Other change types except for **codeAdded** and **codeChanged** may also reveal some regular patterns as the number of projects grows.

## Acknowledgment

The work described in this article was partially supported by the National Basic Research Program of China (973 Program 2014CB340702), the National Natural Science Foundation of China (No. 61373013, 61170067).

## References

- [1] T. Apiwattanapong, A. Orso, and M. J. Harrold. Jdiff: A differencing technique and tool for object-oriented programs. *Automated Software Engineering*, 14(1):3–36, 2007.
- [2] D. Cubranic and G. C. Murphy. Hipikat: Recommending pertinent software development artifacts. In *Proceedings of the International Conference on Software Engineering*, pages 408–418, 2003.
- [3] M. D’Ambros and M. Lanza. Software bugs and evolution: A visual approach to uncover their relationship. In *Proceedings of the 10th European Conference on Software Maintenance and Reengineering*, pages 10–238, 2006.
- [4] M. Fischer, M. Pinzger, and H. Gall. Populating a release history database from version control and bug tracking systems. In *Proceedings of the International Conference on Software Maintenance*, pages 23–32, 2003.
- [5] S. Kim, E. J. Whitehead, and Y. Zhang. Classifying software changes: Clean or buggy? *IEEE Transactions on Software Engineering*, 34(2):181–196, 2008.
- [6] S. Kim, T. Zimmermann, K. Pan, and E. J. Whitehead. Automatic identification of bug-introducing changes. In *Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering*, pages 81–90, 2006.
- [7] B. Meyer. Object-oriented software construction. *Prentice Hall International Series in Computer Science*, 1988.
- [8] A. Mockus and L. G. Votta. Identifying reasons for software changes using historic databases. In *Proceedings of the International Conference on Software Maintenance*, pages 120–130, 2000.
- [9] X. Ren, F. Shah, F. Tip, B. G. Ryder, and O. Chesley. Chianti: a tool for change impact analysis of java programs. 39(10):432–448, 2004.
- [10] B. G. Ryder and F. Tip. Change impact analysis for object-oriented programs. In *Proceedings of the 2001 ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*, pages 46–53, 2001.
- [11] J. Śliwerski, T. Zimmermann, and A. Zeller. When do changes induce fixes? *ACM SIGSOFT Software Engineering Notes*, 30(4):1–5, 2005.
- [12] J. Spacco, D. Hovemeyer, and W. Pugh. Tracking defect warnings across versions. In *Proceedings of the 2006 international Workshop on Mining software repositories*, pages 133–136, 2006.
- [13] C. Williams and J. Spacco. Szz revisited: verifying when changes induce fixes. In *Proceedings of the 2008 workshop on Defects in large software systems*, pages 32–36, 2008.

# Development of A Sliding Window Protocol for Data Synchronization in a Flow Cytometer

Junhua Ding<sup>1,2</sup>

1) Dept. of Computer Science  
East Carolina University  
Greenville, NC 27587  
dingj@ecu.edu

Yuxiang Shao<sup>1,2</sup>

2) School of Computer Sciences  
China University of Geosciences  
Wuhan, Hubei, China  
shaocx25@163.com

Dongmei Zhang<sup>2</sup>

2) School of Computer Sciences  
China University of Geosciences  
Wuhan, Hubei, China  
jjielee@163.com

**Abstract** — Flow cytometers are instruments for analyzing blood cells individually and have been widely used for clinical diagnostics and research discovery. A flow cytometer is able to acquire signals of several independent physical or chemical parameters of each cell altogether. Therefore, synchronization of data acquisition among all parameters of each cell is required for a reliable flow cytometer. Most advanced flow cytometers use complex electronic design to ensure the data synchronization in the beginning of signal acquisition, but signals could still be corrupted during transmission or storing. In this paper, we present a software solution based on a dynamically sliding window protocol for data synchronization in a flow cytometer. Our solution is used for either a supplementary or a replacement of the hardware solution. The overhead due to the computing of sliding window is minimal. The design, implementation and testing of the solution is described in this paper, which shall be useful for others who are developing similar systems.

**Keywords**- sliding window protocol; flow cytometer; data acquisition; software testing

## I. INTRODUCTION

Flow cytometer is a high performance medical instrument for analyzing blood cells individually. Since it was invented in 1960s, it has been the premier instrument for analyzing single cells, and it is able to analyze blood cells from several hundred to 30 thousands every second [7]. Most flow cytometers detect blood cells with optical parameters coming from interaction between cells and laser beams. The optical parameters include forward scatters, side scatters, and different fluorescence signals. Flow cytometers have been widely used for clinical diagnostics and research discovery for over than 40 years. Because flow cytometers are able to analyze cells one by one with multiple parameters, they are effective and efficient for analyzing many immune related diseases such as HIV-infection and leukemia. For example, flow cytometers have been widely used for CD3+, CD4+ and CD8+ screening [6]. They are also used for apoptosis analysis and pharmaceutical discovery. The marketing value of flow cytometers including instruments, reagents, software and services in the world was over than \$2.6 billion in year 2010, and it will grow over than 10% annually in the next decade. However, flow cytometers are expensive, and the price of a cheap instrument with basic features could be over than \$50,000, and the service and reagents cost for each instrument could cost another \$30,000 easily each year. The price of an advanced flow cytometer system including a sample preparation station could be over than \$500,000. HIV-infection now is a serious problem in many developing countries, and

flow cytometers are the most effective and highly efficient device to detect the infection in the early phase. Unfortunately, flow cytometers are too expensive for users in the developing countries who are suffering of HIV-infection. Building an affordable flow cytometer is important to improve the life quality in developing countries. In this paper, we describe a flow cytometer we are building with affordable and comparable performance in mind. We particularly discuss the sliding window protocol that we designed for the data synchronization in the system.

A flow cytometer detects several parameters such as forward scatter and fluorescence for each cell to discover different information in the cell. Each parameter is detected by an independent signal detector like photomultiplier tube (PMT). The signals of the multiple parameters for a cell have to be acquired together. Missing a signal of any parameter will mess up the data of other cells and then the analysis result of the sample will be incorrect. The worst situation is that a user may unable to detect the error. Most existing flow cytometers have a complex electronic design for synchronizing the multiple parameters so that signals of all parameters have to be acquired together or none. Some inexpensive flow cytometers do not have any synchronization mechanism to ensure the data synchronization, which may produce potential error results.

In this paper, we present a sliding window protocol for synchronizing the data acquisition of multiple parameters in a flow cytometer. The protocol can be used as a supplementary of the hardware solution or an addition to a flow cytometer that does not implement data synchronization. The overhead of the calculation of the synchronization in the software is ignorable. Our experiments have shown that our design of the sliding window protocol for data synchronization is both effective and efficient.

The main contribution of this paper is due to the design and analysis of the dynamically sliding window protocol for the data synchronization in a flow cytometers. We introduce the background of flow cytometers and the issue regarding the data synchronization among multiple parameters. We describe the detail of design and implementation of the sliding window protocol, and we also validate the effectiveness and efficiency of the design and implementation using different techniques such as testing, simulation and model based testing. The design, implementation and validation of the sliding window protocol for the data synchronization could be useful to others who are building similar systems.

The rest of this paper is organized as follows: Section 2 presents a brief introduction of flow cytometers and data acquisition in flow cytometers. Section 3 describes the design, implementation and validation of the dynamically sliding window protocol. Section 4 is the related work, and Section 5 concludes this paper.

## II. BACKGROUND

In this section, we give a general description of flow cytometers.

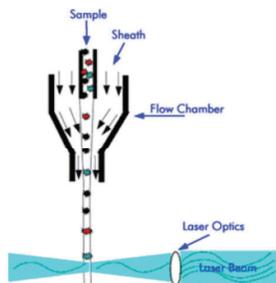


Figure 1. A schematic of a flow cell [10]

### A. Flow Cytometers

Although the origins of flow cytometers can be tracked to 19<sup>th</sup> century, the development of flow cytometers became feasible until late of 1970s when reliable lasers, commercially produced monoclonal antibodies and inexpensive computer became available [7]. Cytometry is about the measurement of physical and chemical as well as biological characteristics of cells, and flow cytometry is a process for measuring individual cells when they pass through detectors in a fluid stream [9]. Flow cytometer is an instrument of flow cytometry for measuring size, internal complexity, and fluorescence of cells based on measurement of electronic and optical parameters obtained as a result of the interrogation of flowing cells and lights [9]. A typical flow cytometer includes several major components: (1) a sample handling component for taking blood from sample tubes or wells; (2) a flow cell for building the blood sample into a fluid stream where the cells pass the light sources, (3) an optical component consisting of one or more laser light sources, (4) signal detection component for acquiring light scatter signals and fluorescence signals, and (5) a software system for instrument management and data analysis. A schematic representation of a flow cell for a typical flow cytometer is shown in Figure 1. The cell sample is sent to a tiny tube and then injected into the stream of sheath, which ensures the cell sample remain in the center according to laminar flow principle [8]. The cells in the sample interrogate with laser lights when they pass through them individually. Corresponding signal detectors acquire the scattered lights and fluorescence signals when a cell passes through a light source. Forward scattered lights are proportional to the surface area and size of the cell, and side scattered lights are proportional to the granularity and internal complexity of a cell. When a cell is prepared with monoclonal antibody and the antigen of the cell binds to the antibody, the cell will activate fluorescence in particular wavelength when it interacts with a laser light. Fluorescence signals are used for

recognizing specific antigens in the surface of cells. In a flow cytometer, fluorescence with each different wavelength is filtered with an optical lens and mirrored to a fluorescence detector, which normally is PMT. Figure 2 shows a typical configuration of a flow cytometer, where structure of the optical and signal detection components are shown in detail. An embedded software system is used to automate and control the instrument. Acquired signals in the instrument are sent to the application software system of the flow cytometer via network, and users manage the cell analysis and the instrument via the application software.

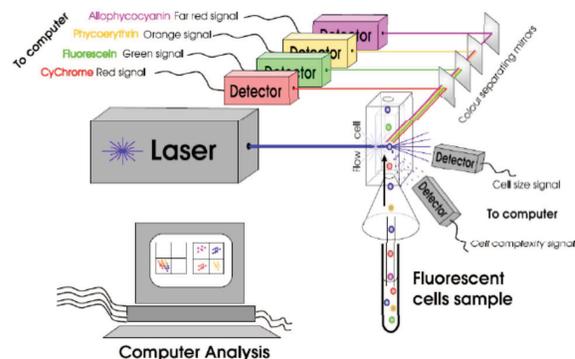


Figure 2. A schematic of a flow cytometer [10]

### B. Data Acquisition in Flow Cytometers

In this section, we describe the data acquisition and synchronization in flow cytometers.

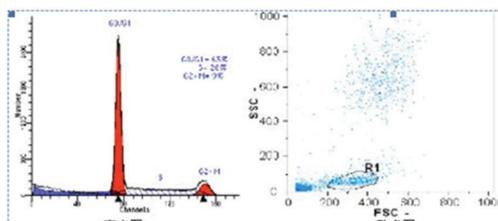


Figure 3. An illustration of data display and analysis

When a cell passes through the flow cell and interrogates with a laser beam, it generates signals like forward scatter signals or fluorescence signals. Each signal detector acquires the signals of a particular parameter. We build the set of signals of all parameters for each cell as a record, and each item in the record is a signal value of a parameter. If the value of an item is 0, then we know that the corresponding detector of the parameter did not get the signal of the cell. The 0 value of a parameter maybe caused by weak signal or problem of the detector, and it may occur many times. The system processes each record to exclude noise signals such as those coming from debris or other irregular particles. Each processed record is put into a buffer, and then the records are packaged and delivered to the workstation for data analysis and additional processing. The data analysis software stores the data records into a file or database in a standard format such as FCS 3.0 so that the data can be exchanged among different systems. At the same time, the data are displayed in different graphic formats such as two dimension dot plots or density plots or

one dimension histograms. Users are able to display the plots with different combinations of the parameters in the 2 dimension plots or histograms for different parameters. The classification of the cell population in the sample can be observed in the plots in some cases. In many cases, users need to select partial of the cell population or divide the cell population into several regions in the plots for further processing, and the procedure is called gating [9]. Statistics such as mean value, cell count, standard derivation or percentage of a cell type in a sample are calculated based on gating. Figure 3 shows a sample of data display and analysis plots in a flow cytometer. The left plot in Figure 3 is a histogram of a parameter, and the right plot is a dot plot of side scatter *v.s.* forward scatter, and it is gated with *RL*.

**Table 1: Sample data collected for each parameter**

	<i>FS</i>	<i>SS</i>	<i>FL1</i>	<i>FL2</i>	<i>FL3</i>
<i>cell1</i>	1	1	1	1	1
<i>cell2</i>	x	2	2	2	2
<i>cell3</i>	3	3	x	3	x
<i>cell4</i>	4	x	4	4	4
<i>cell5</i>	5	5	5	5	5

**Table 2: Sample records built for each cell in Table 1**

	<i>FS</i>	<i>SS</i>	<i>FL1</i>	<i>FL2</i>	<i>FL3</i>
<i>cell1</i>	1	1	1	1	1
<i>cell2</i>	3	2	2	2	2
<i>cell3</i>	4	3	4	3	4
<i>cell4</i>	5	5	5	4	5
<i>cell5</i>				5	

Although cells in a flow cytometer pass detectors individually, data analysis is still performed on a collection of cells like 1000 cells or 5000 cells. Therefore, the signals records are continuously acquired and stored when cells are passing in the fluid stream. Ideally, when a cell passes through a light beam, each detector detects one and only one signal of a particular parameter and all detectors get their signals altogether at the exact same moment and then reset themselves for next acquisition. However, one detector may not detect a passing cell, and other detectors do. In this case, the next signal acquired by the detector that missed the previous one will be accidentally put into the previous position that is supposed to be occupied by the missing value of last cell. The problem will only be found at the end of the sample analysis due to some missing values of the last several cells. Then the analysis data of the whole sample have to be discarded. In the worst situation, each detector missed the same number of signals, and then the problem even will not be found easily.

Let's explain the problem in details with examples. We assume the instrument collects 5-parameters signals for each cell. The 5 parameters are forward scatter *FS*, side scatter *SS*, fluorescence 1 *FL1*, fluorescence 2 *FL2*, and fluorescence 3 *FL3*. Each parameter has a signal detector for, and each

detector collects data independently when a cell interrogates a laser beam. Table 1 shows the acquired signals of 5 cells, where x represents a lost signal, and an integer number means a valid value that is collected. Then the program is going to build a record for each cell, and each record has 5 parameter values representing the analysis result. Since no one is aware of the lost values due to the high speed of data generation (most existing flow cytometers process around 10000 cells each second), the positions of the lost signals are filled by signals of coming cells. The 5 records built based on data collected in Table 1 is shown in Table 2, where we found the records of *cell2*, *cell3*, *cell4* and *cell5* are all wrong. Therefore, the analysis result of the whole sample has to be discarded.

Some hardware design can solve the data synchronization problem mentioned above. For example, one detector (it is the detector of *FS* in most cases since *FS* has the strongest signals) is designed as the master detector and it facilitates the synchronization among all detectors. When the master detector detects its signal, the corresponding position of each parameter in the record is filled with the signal values the detectors detected or a default value 0. If the master detector was not triggered by a passing cell, then none of other detectors will be triggered either. However, signals still could be lost during the data transmission or storing. For enhancing the quality of the instrument we developed, we designed a dynamically sliding window for implementing the data synchronization. It could be particularly useful for instruments that do not have a hardware based data synchronization design.

### III. SLIDING WINDOW PROTOCOL

In this section, we discuss the design, implementation and validation of the dynamically sliding window for data synchronization in a flow cytometer.

#### A. Design

When a cell interrogates a light beam, it may or may not trigger the signal detectors. If one detector loses a signal, then the data acquired for a sample are messed up. In some cases, the messed up of acquired data may even not be found at the end of the analysis. In order to solving the problem, we designed a dynamically sliding window for checking the data integrity at real time. We assume the instrument acquires values of 5-parameters signals of one light source for each cell. If the instrument has two laser light sources, then there are 10 signals total when a cell interacts with 2 laser lights each time. We discuss the instrument that has only one laser light, and the design can be easily extended for systems that have multiple laser lights. The 5 parameters we defined as *FS*, *SS*, *FL1*, *FL2*, and *FL3*. The missing of a signal could happen at any time. Therefore, the size of the sliding window is 1 in the beginning, and it increases if no missing of signals occurs next time, and it continuously increases the size until it reaches to a preset limitation as soon as no signal is lost. If a signal is lost, then the size of the sliding window is reduced until it reaches to 1 if losing of signals is continuously found. When the size of the window is larger than 1, and the numbers of missing signals for each parameter is same, then unfortunately our design is unable to detect the problem. However, the possibility of the case in a small size of window is extremely rare; we don't consider the

solution of the case. Here is the design of the dynamically sliding window:

1. As soon as the data acquisition of a sample starts, the size of the slide window is set to 1. A data buffer that has 5 fields is dedicated for storing data collected from each detector, and each detector is assigned with a field. Figure 4 shows how each detector stores its signal values into the fields in the buffer, where the arrow line means the direction of data pumping, and the dash line represents data stored in the buffer.

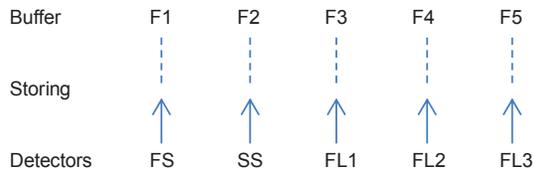


Figure 4: An illustration of data storing in a buffer

2. According to the speed setting of the fluid system in the flow cytometer and the sample type, we are able to calculate the minimal time interval of 2 cells passing through a laser beam. We set the time interval  $s$  as the cut off time for checking the sliding window. Since each detector gets its signal in near to light speed, we consider all detectors get their signals at the exact same moment.
3. When the first signal of  $FS$  is received and stored in the buffer, the program starts clock  $t$ . As soon as  $t = s$ , the program checks the buffer, if each field has one value, the analysis result is complete; otherwise, the fields that don't have signals are filled with value 0. This step is to ensure that no value is lost in the first line.

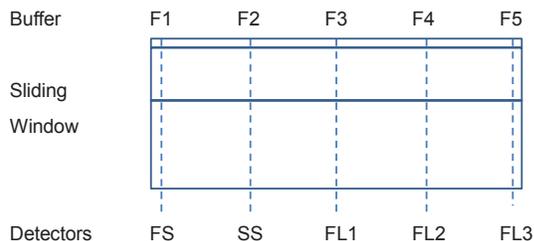


Figure 5: An illustration of sliding windows

4. The program increases the size of the sliding window to 10 so that the buffer will accept additional 10 cells. As soon as the program finds that the buffer already stored additional 10 values of  $FS$ , it immediately checks all other fields. During the time, if new signals are coming, they are stored in a temporary buffer that has the same data structure as current one. Normally it will not happen since it takes much less time for checking the synchronization in a window than the time interval  $s$  of two cells passing through the light.
5. The program counts the number of signals for each field in the buffer. If an inconsistency of number among fields is found, it means some detectors missed signals for some cells. Since  $FS$  is the master parameter, the number of

signals in other fields could be over than the size of the window if  $FS$  lost some values.

6. If the program finds a missing signal in the window, the analysis results of all cells in the window have to be discarded because the program does not know which cell that really missed a signal. The size of the window is reduced to half of the current window until the size is reduced to 1 if missing of values is keeping occurring.
7. If every field has the same number of signal values within the window, the size of the window is doubled until it reaches to 100, which is the maximal size of the window, as soon as no missing signal is found. A flow cytometer normally analyzes around 10000 cells for each sample; therefore, discarding 100 cells of 10000 cells is still acceptable. The users can decide whether to continue acquire data from the sample until it reaches to a preset criterion. The idea of the dynamically sliding window is shown in Figure 5, where the rectangles represent sliding windows.
8. If there are some items in the temporary buffer at the end of analysis, the program just simply counts the number of signals in each field. If the number is same for every field, the data is appended to the data in the original buffer. If inconsistency is found, the data in the temporary buffer are discarded.

#### B. Implementation

The main embedded program of the flow cytometer we developed has only one process that is implemented in an ARM single board computer, and the data acquisition program is implemented in a FPGA board. The sliding window program is implemented in the FPGA board with data acquisition programs.

The program sets a buffer according to the number of cells to be processed in a sample. For example, if a user needs to analyze 10000 cells from a sample, the buffer size is 200KB considering each value of a parameter is a 32 bits number. The real buffer is 400KB since each detector actually acquires two different signal values for each cell. We don't discuss the case since the sliding window only needs one signal for each parameter. Each acquired datum is processed for multiplying or digitalizing by the detector, and then the datum is stored in a particular position of the data buffer. The sliding window program monitors the number of  $FS$  data, stores and records the position of the current processing. As soon as the number of  $FS$  data in the window equals to the size of the window, which starts from the next position  $w_s$  of the last processed position in the buffer to position  $w_e$ , which is  $w_s$  plus the current size of window  $w_w$  ( $w_e$  shall be the last item in the current buffer). When the program counts the number of items in each field in the window, it counts all items from position  $w_s$  to the last item  $w_e$  of the field in the buffer. During the counting period, incoming data are stored in a temporary buffer. If a missing value is found, the data from position  $w_s$  to the last one in the buffer are deleted, and the size of the window is reduced to half of  $w_w$  or 1 if  $w_w$  is 10. If data in the window are complete, then  $w_s$  is set to the next position of  $w_e$ , and the size of the window is set to the smaller one between  $w_w$  and 100.

### C. Validation

Although implementing the dynamically sliding window is easy, testing the program is challenged due to several reasons: (1). Missing of values from each signal detector is very rare so that it is very difficult to reproduce missing of values; (2). The data generation from each detector is very fast, usually one signal for each parameter every 0.05 millisecond or less; (3). The data are produced at real time, and testing has to be performed at real time; (4). The time period between two subsequent values of a parameter could be less than 0.05 millisecond, and it is difficult to control the time period during the testing; and (5). The number of data items for a sample could be 10000 or more, which makes the verification of the testing result fairly difficult.

In order to test the implementation of the sliding window thoroughly, we test the program with three different ways [11]: (1). System testing; (2) Special cases based testing; and (3) Model based testing.

*System testing.* If some signal values of a parameter are lost in the early phase of the data acquisition, the analysis result of the sample will be messed up (see the explanation in section 2.B). We can prepare two exact same blood samples such as some standard quality control kits [7]. Then we run one sample in a reference flow cytometer that has been tested and calibrated, and run another sample in the flow cytometer that has the sliding window under test. We compare the analysis results such as patterns in the dot plots, histograms and statistics from two tests like those shown in Figure 6. If a significant difference of the two results is found, then something must be wrong in our program or design. However, it is not necessary that the messed-up data due to missing of values during the data acquisition will cause a significant difference of two analysis results. In many cases, the difference between two analysis results might be very difficult to be observed. Considering the fact that missing values in the data acquisition is fairly rare, system testing does not work well for testing the sliding window program.

*Special cases based testing.* It is impossible for us to interrupt some detectors to disable the data acquisition of some cells at real time due to the high speed of data acquisition. In addition, the rate of dropping signals is so low that no any signal may be lost even running the analysis for several days without interruption. It is difficult to change the firmware in the instrument for dropping signals. However, we can work around the problem via changing the setting of the instrument and testing. For example, we may set a very large buffer such as 1MB for storing the acquired data, and dilute the blood sample to such as 20% of the original one. Then it will take about 25 times of time to acquire enough data to fill the space in the buffer. We also set the size limitation of the sliding window to 1000 instead of 100. Based on above configuration, we have enough time to turn off one detector and then turn on it immediately during data acquisition. Then we can check whether the final result has discarded the cells that have incomplete data, and number of items for each field is same. If we turn off a detector before the data acquisition starts, we can check whether the missing value in the field of the first cell has been filled with 0 as we designed. The speed of fluid stream

can be adjusted in some instruments, and then we can also increase the time interval between two passing cells via reducing the speed of the fluid stream. However, it is still difficult to test a situation that a detector only drops one or two cells since many cells have passed the light during we turn on and off the detector even with above modifications. In addition, it is almost impossible to test the case that signals are dropped from two or more detectors randomly.

*Model based testing.* In order to thoroughly testing the sliding window protocol, we developed a program to simulate the data acquisition and the sliding window. The simulator generates a signal for each parameter and pumps it into the buffer every 0.05 millisecond or less, and randomly drops a datum in the buffer for any parameter. The simulator is able to simulate many different cases such as dropping a signal in the beginning or at the end of data acquisition, or dropping several signals of multiple parameters together. We run and test the simulation program in a PC environment. We shall test many different cases via combining different scenarios based on combinatorial testing technique [5]. The scenarios could include ones like missing data items in the beginning, missing just one data item for a parameter, missing several data items altogether for multiple parameters, missing 2 or more data items altogether for a parameter, and the time interval of generating signals of two cells could be set as 0.05 millisecond, 0.03 millisecond, 0.01 millisecond, and others between the range. However, it is difficult to know when the program is adequately tested. In this research, we used model-based testing technique to generate adequate tests.

The tool we used in this project is called MISTA [12], which is a model-based testing tool for automated generation of executable test code in model level and program level. It uses function nets (a type of PrT nets extended with inhibitor arcs and reset arcs) [12] for specifying test models so that complete tests can be automatically generated. The detail of MISTA can be found in [12]. Figure 6 is a function net model of the sliding window protocol in MISTA. In the model, place *Signals* and transition *Generator* are used for generating signals, and transitions *SendSS*, *SendFS*, *SendFL1*, *SendFL2*, and *SendFL3* are used for sending corresponding signals to the buffer of each parameter. Transitions *Drop1*, *Drop2*, *Drop3*, *Drop4*, and *Drop5* are used for simulating randomly dropping signals during transmission. When a signal is dropped, and the size of the sliding window is reached, transition *DoReset* is used for discarding signals in buffers within the sliding window. If no signal is lost within current sliding window, then transition *Sent* delivers all signals to place *Success*. The size of the sliding window is set by the guard conditions in transitions *DoReset* and *Sent*. After compiled and inspected executions of the model, we generated tests according to different adequate criteria including reachability tree, state overages and transition coverage. Through mapping the function net model to the Java implementation of the simulator with sliding window, MISTA automatically generated tests for testing the implementation based on the tests generated from the model. Due to the limitation of the space, we don't discuss the details of the tests in this paper.

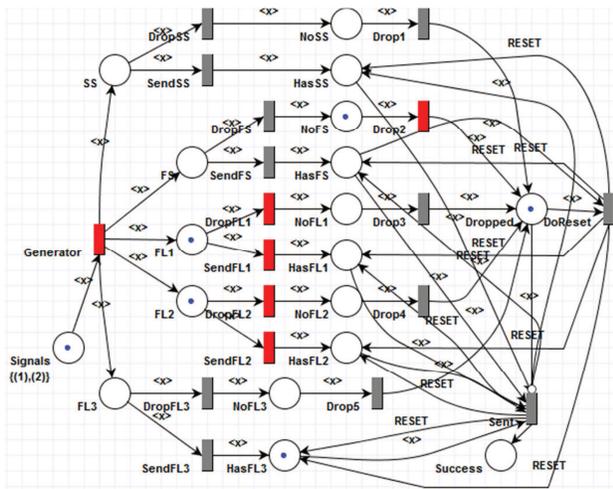


Figure 6: A function net model of the sliding window protocol

The result of the model based testing has shown effectiveness of the design and implementation of the sliding window protocol.

#### IV. RELATED WORK

There are many tutorials on flow cytometers, and the one [9] by Shapiro might be the most referred one. The quality of flow cytometers software is critically important to the quality of the instrument. The software normally is designed to enhance features and ensure the quality of the instrument. Therefore, it is important to develop a rigorous approach to ensure the software quality. In this paper, we designed a sliding window protocol for data synchronization in flow cytometers, and analyzed the design using different approaches to ensure the correctness. Sliding window protocols have been proposed for reliably transforming data packets in-order [2], but the sliding window protocol proposed in this paper is relatively simple since the delivering of signals in-order is not an issue. Formal modeling of software systems provides a solid foundation for formally analyzing software systems. Although formal analysis provides a rigorous way for verifying correctness of software systems, it is infeasible for analyzing large software systems due to the state explosion issue. Software testing is still the most widely adopted technique to check the quality of software, but it lacks rigorosity in many cases. Many researches have been done on combination of formal analysis and testing such as the one discussed in [3] and [4]. The analysis approach used in this paper combines both advantages of software testing and formal analysis. Tests were generated based on formal analysis results and testing bridges the gap between simulation and formal analysis. More important, tests generated from the model not only were used for testing the model, but also were used for generating code level tests so that the consistency between a model and its implementation is well ensured [12].

#### V. SUMMARY AND FUTURE WORK

Data acquisition and analysis is the most important part in a flow cytometer system [6]. In order to facilitate data synchronization among multiple parameters, hardware design has been used in many flow cytometers. We designed a sliding window protocol to ensure the data synchronization. The purpose of our sliding window is used for detecting missing of data during data acquisition. One of the difficulties comes from the high speed of data generation and the independence of data acquisition of each detector, and another difficulty comes from the requirements for rigorous techniques to ensure high quality of the system design and implementation. We address both of the difficulties in this paper. Our validation results show that our solution was able to be used for either a supplementary or a replacement of the hardware solution. The overhead due to the computing of sliding window is minimal. The design, implementation and validation of our solution described in this paper shall be useful for others who are developing similar systems.

#### ACKNOWLEDGMENTS

We thank Dr. Xinhua Hu at East Carolina University and Eric Statler from Beckman Coulter for numerous discussions. This research is supported in part by award #CNS-1262933 from the National Science Foundation. Junhua Ding's research was also partially supported by the guest professorship grant from school of computer sciences at China University of Geosciences.

#### REFERENCES

- [1] P. K. Chattopadhyay, M. Roederer, Cytometry: today's technology and tomorrow's horizons, *Methods*, Vol. 57, pp. 251-258, July 2012.
- [2] D. E. Comer, *Internetworking with TCP/IP, Volume 1: Principles, Protocols, and Architecture*, Prentice Hall, 1995.
- [3] J. Ding, X. He, Formal Specification and Analysis of an Agent-Based Medical Image Processing System, *Intl. Journal of Soft. Eng. and Knowledge Eng.*, Vol. 20, No. 3, pp. 1-35, 2010.
- [4] X. He, H. Yu, T. Shi, J. Ding, and Y. Deng, Formally Specifying and Analyzing Software Architectural Specifications Using SAM. *Journal of Systems and Software*, vol.71, no.1-2, pp.11-29, 2004.
- [5] R. Kuhn, R. Kacker, Y. Lei, J. Hunter, Combinatorial Software Testing, *IEEE Computer*, vol. 42, no. 8, August 2009.
- [6] E. Lugli, M. Roederer, and A. Cossarizza, Data Analysis in Flow Cytometry: The Future Just Started, *Cytometry A*. 77(7): 705-713., July 2010.
- [7] M. G. Macey (Ed.), *Flow Cytometry, Principles and Applications*, Humana Press, 2007.
- [8] F. F. Mandy, M. Bergeron, T. Minkus, Principles of Flow Cytometry, *Transfus. Sci.*, Vol. 16, No. 4, pp. 303-314, 1995.
- [9] H. M. Shapiro, *Practical Flow Cytometry*, 4<sup>th</sup> edn. Wiley-Liss, 2003.
- [10] <http://www.usuhs.mil/bic/cytometry/pdf/tutorial.pdf>, last accessed on March 16, 2014.
- [11] S. Vance, *Quality Code: Software Testing Principles, Practices, and Patterns*, Addison-Wesley Professional, 2013.
- [12] D. Xu, A Tool for Automated Test Code Generation from High-Level Petri Nets. *32nd Int. Conf. on Apps. and Theory of Petri Nets*, Newcastle, UK, June 20-24, 2011.

# An Empirical Study on the Test Adequacy Criterion Based on Coincidental Correctness Probability\*

Xiaoli Zhou<sup>†</sup>, Linzhang Wang<sup>†</sup>, Xuandong Li<sup>†</sup> and Jianhua Zhao<sup>†§</sup>

<sup>†</sup>State Key Laboratory of Novel Software Technology

Dept. of Computer Sci. and Tech. Nanjing University, Nanjing, Jiangsu, P.R. China 210093

Email: zhaohj@nju.edu.cn

**Abstract**—Coincidental correctness occurs when a fault is executed but no failure is detected. Coincident correctness is prevalent and adversely affects the effective of testing. In this paper, we improve the test adequacy criterion based on coincidental correctness probability which is proposed in our previous work and make an empirical study comparing our test adequacy criterion with two coverage-based test criteria. We take the Pearson Correlation Coefficient between the adequacy level and the capability of the test suite to find errors as a metric of the effectiveness. Our experiment involved 20 seeded versions of the programs from SIR. The results consistently approve that our test adequacy can effectively quantizes the capability of test suite to find errors.

## I. INTRODUCTION

Currently, coverage-based test adequacy criteria are widely used both in academia and industry. However, high code coverage does not mean high test adequacy regarding to the defect removing. An empirical study reports that there does not exit a causal relationship between code coverage and defect coverage [1].

Coverage-based testing for white box assume that the execution of faulty statement has a high probability to result in observable incorrect output. However, this is not necessarily always the case. A fault can be observed during testing, if and only if the following conditions all be met: 1) the defect is executed or reached, 2) the program has transitioned into an infectious state, 3) the infection has propagated to the output [2]. When 1) is met and 2) is not met, the coincidental correctness occurs [3][4]. The error may be hidden even if the faulty statement is executed. The main reason is the existence of coincidental correctness. Many experimental evidences show that, coincidental correctness is prevalent and responsible for reducing effectiveness of testing [3][4].

In our pervious work [5][6], we propose a test adequacy criterion based on coincidental correctness probability (CCP for short). In this paper, we make an improvement and an empirical study on the CCP-based test adequacy criterion. The main contributions of this paper can be concluded as follows:

- i) We improve the algorithm to evaluate the coincidental correctness probability with a refined control-flow analysis.
- iii) We take the Pearson Correlation Coefficient as an evaluation metric and make a comparison case study

against the test adequacy criteria based on the statement coverage and branch coverage. Experiments in this paper can more accurately reflect the situation in practice, and shows that CCP-based test adequacy criterion has a stronger correlation with errors than other criteria.

Section II shows the approach to estimate the CCP-based adequacy in detail. Section III presents the experimental work and results. Finally, Section IV discusses some related works and presents our conclusion and future works.

## II. AN APPROACH TO ESTIMATE THE CCP-BASED TEST ADEQUACY CRITERION

This section describes how to estimate the CCP, and presents the definition of CCP-based adequacy criterion. The basic idea is the same with previous works [5][6], and we improve the algorithm with a refined control-flow analysis.

An execution of a statement is called a statement instance. During the execution, statement instances load or store values in some memory units. Testers usually directly or indirectly check some memory units which are called check points at the end of the execution to determine whether the test is passed. The value stored in a memory unit is defined in some statement instances and then used in other statement instances. Thus these instances have a use-definition dependence for their correctness. If an error is triggered and generates an incorrect value, this incorrect value may be stored into some memory units. If other statement instances use these incorrect memory units, the incorrect values may be transferred till the end of this run or may be covered by other statement instances.

Before everything begins, here are two hypotheses:

- i) We estimate the test adequacy only for the test suite which is passed in testing.
- ii) We assume that a program has only one error.

### A. Instrumentation and static analysis

We identify variables and operations in statements for instrumentation. For every variable, we instrument to get its runtime variable values, its memory address and how it is related (use or definition). Also, we build the static control-flow graph of the program.

### B. Build dynamic dependencies

In this section, we described how to construct the dynamic use-definition and control-flow dependencies for runtime statement instances after running the instrumented program.

\*This work is supported by the National Natural Science Foundation of China (No.91118007, No.61328491, No.61170066), and by the National 863 High-Tech Programm of China (No.2012AA011205)

<sup>§</sup>Corresponding author

1) *Build the dynamic use-definition dependence graph:*

We build a dynamic use-definition dependence based on the memories used or defined by instances. The detailed method is described in [5].

2) *Build the control-flow dependence:*

Control-flow can indirectly affects the correctness of the output of a program. If the conditional expression generates an incorrect value, some statements will be mistakenly executed while some other statements will be mistakenly unexecuted. The influences on the mistakenly unexecuted instances are ignored in this paper as they are difficult to analysis.

Conditional transfer statements and loop statements are two kinds of typical control statements. For a statement instance  $R$ ,  $CE(R)$  denotes the set of control statements on which  $R$  is dependent.  $CE(R)$  is computed as follows:

- i) If  $R$  is an instance of a statement in a branch of an IF statement, the corresponding instance of conditional expression of this IF statement is in  $CE(R)$ .
- ii) If  $R$  is an instance of a statement in the body of a WHILE statement, all instances of the conditional expression of the WHILE statement from the start of the WHILE statement to  $R$  are in  $CE(R)$ .

If  $R$  is in more than one branches or bodies of IF/WHILE statements,  $CE(R)$  will be computed for every IF/WHILE statement according to the above rules.

C. *Impact factors for correctness possibility*

In this section, we describe how the use-define and control-flow dependencies have an impact on the correctness of the outputs of a program execution and specify the degrees of these effects.

1) *Impact factors of Operations:*

The impact of use-define dependence mainly arises from operations which may cover the incorrect intermediate results. The result of an operation has a probability to be correct even if its operands are wrong. We specify the probabilities that an operation generates an correct result under different situations. Because of space limitation, we only describe how the probabilities about binary operations are specified.

TABLE I. THE EFFECT DEGREES FOR BINARY OPERATORS

Correctness of Left Operand(a)	Correctness of Left Operand(b)	Effect degree
T	T	$Q_1 = 1.0$
T	F	$Q_2$
F	T	$Q_3$
F	F	$Q_4$

We use “ $a\#b$ ” to represent an binary operation with an operator “ $\#$ ” and two operands  $a, b$ . In Table I,  $Q_1, Q_2, Q_3$  and  $Q_4$  are the conditional probabilities that the result of “ $a\#b$ ” is correct under different situations. When  $a$  and  $b$  are both correct, “ $a\#b$ ” is definitely correct. So  $Q_1$  is always 1.0. The values of  $Q_2, Q_3$  and  $Q_4$  are differently specified for different operators.

Assuming that  $P_a, P_b$  and  $P_{a\#b}$  are the correctness probabilities of two operands and the result respectively,  $P_{a\#b}$  can be estimated by:

$$P_{a\#b} = P_a * P_b * 1.0 + P_a * (1 - P_b) * Q_2 + (1 - P_a) * P_b * Q_3 + (1 - P_a) * (1 - P_b) * Q_4 \quad (1)$$

2) *Impact factors of control-flow:* Control-flow dependence is used to revise the correctness probability estimated from use-definition dependence. For an instances  $R$ , the revision is:

$$P'(R) = P(R) \times (\prod_{R' \in CE(R)} P(R') + (1 - \prod_{R' \in CE(R)} P(R')) \times AdjED) \quad (2)$$

Here,  $P(R)$  and  $P(R)'$  are the correctness probabilities of  $R$  before and after the revision, and  $AdjED$  is the effect degree to adjust the probability. If any instances in  $CE(R)$  is incorrect, the result of  $R$  has a small probability to be correct. Thus, we set the  $AdjED$  as 0.2 to adjust the probability of  $R$  under the situation of incorrect runtime control-flows.

D. *An algorithm to estimate the coincidental correctness probability*

The algorithm to estimate the CCP of one program execution is described as Figure 1. For an execution, we estimate one CCP for each statement in the program assuming that this statement contains an error and all other statements are correct.

Given a serial program and a test case, we collect runtime information by executing the instrumented program. For a statement instance  $R$ , we use the  $UG(R)$  and  $CG(R)$  to denote the set of memory units that used or changed by  $R$ . We suppose that the  $UG, CG$  and  $CE$  of each instance are already computed as described in previous sub-sections. In the algorithm, we estimate the correctness probability of each instance by order. Finally, the CCP of an execution is the probability that all check points are coincidentally correct.

**Input:**

- $S$ : The statement is assumed to contain an error.
- $InstancesList$ : The runtime instances sequence.
- $MemorySet$ : The memory unit set used/updated by instances.
- $Checkpoints$ : the set of check points.

**Output:** CCP of the program execution

- 1: Initialize correctness probability of all memory units in  $MemorySet$  as 1.0;
- 2: **for each** node  $R$  in  $InstancesList$  **do**
- 3:   **if**  $R_i$  is an instance of  $S$  **then**
- 4:     /\*the result value of  $R_i$  is wrong by default.\*/
- 5:     Set the correctness possibility of memory units in  $CG(R)$  as 0;
- 6:   **else**
- 7:     /\* 1. Estimation based on use-define dependence\*/
- 8:     Estimate the correctness possibility of memory units in  $CG(R)$  based on the correctness probability of memory units in  $UG(R)$  and effect degrees according to the formulation (1);
- 9:     /\* 2. Revision with control-flow dependence\*/
- 10:     **if**  $CE(R) \neq \emptyset$  **then**
- 11:       Revise the correctness probability of memory units in  $CG(R)$  according to formulation (2);
- 12:     **end if**
- 13:   **end if**
- 14: **end for**
- 15: **return**  $CCP = \prod_{K \in Checkpoints} P(K)$
- 16: /\*  $P(K)$ : the correctness possibility of the value in check point  $K$ \*/

Fig. 1. The algorithm to estimate the CCP of a program execution

The CCP of a program execution w.r.t. a statement denotes the probability that there is an error in this statement and it’s triggered but not observed by testers. A higher CCP means that we need more test cases to verify the suspect statement. The granularity of our approach is statement level.

E. *The CCP-based adequacy criterion*

For a statement  $S$ , we get a set of CCPs, one for each test case. The test adequacy level of a test suite  $TS$  for statement  $S$  is defined as the following formulation:

$$AC(S) = 1 - \prod_{T \in TS} CCP(S, T) \quad (3)$$

Here,  $AC(S)$  is the adequacy level of the test suite for the statement  $S$ .  $CCP(S, T)$  denotes the probability that the error

in  $S$  is hidden by coincidental correctness after running a single test case  $T$ .  $\Pi_{T \in TS} CCP(S, T)$  is the probability that all the test cases in  $TS$  fail to find the error. The lower the  $\Pi_{T \in TS} CCP(S, T)$  is, the higher the probability of finding errors will be. Thus, the  $AC(S)$  denotes the probability to find the error in the statement  $S$  if the error exists. For a test suite, we collect a set of  $AC$ s, one for each statement in the program.

For an adequacy level  $AL$  of the whole program, the test suite satisfy adequacy level  $AL$  only when all  $AC$ s of statements is higher than  $AL$ . This means that, when a test suite satisfy the adequacy level  $AL$ , all the probabilities that an error is hidden in each statements are lower than  $1 - AL$ .

### III. EXPERIMENTAL WORK

#### A. Tool implementation and subject programs

Our experimental tool contains three modules: variable instrumentation module, runtime information analysis module and CCP-calculator. The variable instrumentation module is built on the grammar tool ANTLR. This module identifies variables and operations in statements and instruments the program for collecting runtime information. After running the instrumented program, runtime information analysis layer collects runtime information to build the dynamic use-definition and control-flow dependencies as described in the subsection II-B. The CCP-calculator computes the CCP of each test case assuming an error in each statement and finally gives the test adequacy level of the test suit. Because of the time and efforts limitation, programming language treated by our tool is limited to a subset of GNU C[7].

We involved 20 seeded versions of the selected two C programs with suitable size and the corresponding test suites from Software-artifact Infrastructure Repository (SIR)[8]. Table II provides the basic information about the subject programs and the test suites, including the sizes of the programs, test suites size( $|T|$ ), the number of failed test cases( $|T_F|$ ) and the percentage of failed tests. All seeded versions of programs contain an error.

TABLE II. SUBJECT PROGRAMS

Program	Size (LOC,Procedures)	$ T $	$ T_F $	$ T_F / T $
<i>tcas_v1.c</i>	173, 9	1608	308	0.19
<i>tcas_v2.c</i>			252	0.16
<i>tcas_v3.c</i>			158	0.1
<i>tcas_v4.c</i>			319	0.52
<i>tcas_v5.c</i>			139	0.09
<i>tcas_v6.c</i>			132	0.08
<i>tcas_v7.c</i>			36	0.02
<i>tcas_v8.c</i>			36	0.02
<i>tcas_v9.c</i>			126	0.08
<i>tcas_v10.c</i>			239	0.15
<i>tcas_v11.c</i>			128	0.08
<i>tcas_v12.c</i>			142	0.09
<i>tcas_v13.c</i>			190	0.12
<i>totinfo_v1.c</i>	565, 7	1052	29	0.03
<i>totinfo_v2.c</i>			199	0.19
<i>totinfo_v3.c</i>			36	0.03
<i>totinfo_v4.c</i>			227	0.22
<i>totinfo_v5.c</i>			145	0.14
<i>totinfo_v6.c</i>			45	0.04
<i>totinfo_v7.c</i>			71	0.07

#### B. Metric of Criterion Effectiveness and Experiment Design

Test adequacy aims to reflect that to what extent the test suite can prove the correctness of program. If the correlation

between the test adequacy level and the capability to find errors is strong, we think this test adequacy criterion can accurately reflect the capability of the test suite to find errors and also the capability to prove the correctness of the program. Hence, to investigate the effectiveness of our adequacy criterion, we take the correlation between the adequacy level and the probability to detect the error as a main metric.

1) *Foundation knowledge about correlation*: To quantify the correlation and dependence between two factors, we calculate the Pearson correlation coefficient(PCC for short). In statistics, Pearson correlation coefficient is widely used in the sciences as a measure of the strength of linear between two variables, giving a value between +1 and -1 inclusive.

*Definition 1*: Let  $X$  and  $Y$  be two zero-mean real-valued random variables. PCC is defined [9] as follows:

$$\rho_{X,Y} = \frac{cov(X, Y)}{\sigma_X \sigma_Y} \quad (4)$$

Where  $cov(X, Y)$  is the covariance of the two variables,  $\sigma_X$  and  $\sigma_Y$  is the standard deviations of  $X$  and  $Y$  respectively.

If  $|\rho_{X,Y}|=0$ , then  $X$  and  $Y$  are said to be uncorrelated. The closer the value of  $|\rho_{X,Y}|$  is to 1, the stronger the correlation between  $X$  and  $Y$ . Generally, when the absolute value of correlation is bigger than 0.8, this correlation is considered to be extremely strong.

2) *Experiment Design*: We compare the CCP-based adequacy criterion with the statement coverage-based and branch coverage-based adequacy criterion.

For every seeded version of programs, we get the PCC between the adequacy level and the capability to find errors, by the following steps:

- i) **Get the preliminary information**. We increase the adequacy level from 0.1 to 0.9 with a gap of 0.1. For each adequacy level, a test suite is built by adding distinct test cases from the test-case pool provided by SIR till the adequacy level is reached. For each adequacy level, we build 10000 test suites and record the percentage of test suites that can find the error. For each criterion (CCP-based criterion, statement coverage and branch coverage), we can get the 9 pairs of  $(AdqLevel, ErrFindProb)$ , where  $AdqLevel$  ranges from 0.1 to 0.9.
- ii) **Calculate PCC**. For each criterion, we calculate the PCC between the  $AdqLevel$  and the  $ErrFindProb$  using the formulation (4).

#### C. Result and analysis

This subsection shows the results of the experiments described in III-B2.

1) *The preliminary information of correlation*: The couples of  $(AdqLevel, ErrFindProb)$  of CCP-based criterion are showed in Figure 2. Every line in the graph presents the 9 couples of  $(AdqLevel, ErrFindProb)$  for a seeded version of programs. The detailed results for other two criteria is not presented here because of space limitation. For all seeded programs, the  $ErrFindProb$  grows consistently with the  $AdqLevel$ . For most seeded versions of programs, when the  $AdqLevel$  tends to be 1,  $ErrFindProb$  also tends to be 1.

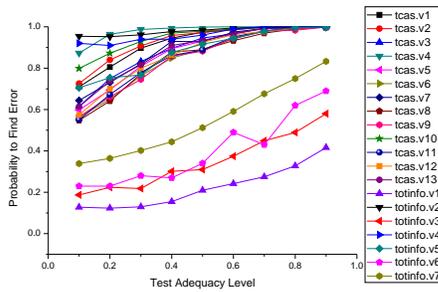


Fig. 2. Adequacy level and the Probability to find errors for programs

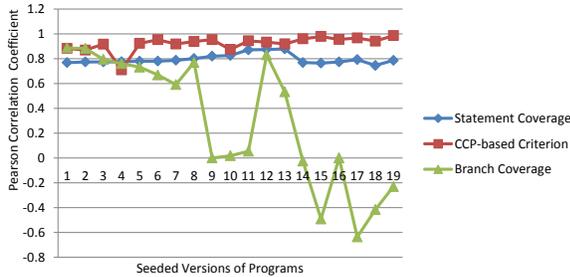


Fig. 3. Pearson Correlation Coefficient

2) *Pearson Correlation Coefficient*: Based on Figure 2, we can get a PCC for every seeded version of program. Every line in Figure 2 can be converted to a PCC value. p Figure 3 presents the PCC of different seeded versions of programs using three test adequacy criteria respectively. Table III provide a summary of the data in Figure 3.

TABLE III. THE OVERVIEW OF THE PCC RESULTS

Object	PCC with CCP-based Criterion			PCC with Statement Coverage			PCC with Branch Coverage		
	Average	Max	Min	Average	Max	Min	Average	Max	Min
tcas.c	0.9028	0.9535	0.7093	0.8084	0.8792	0.7680	0.5787	0.8864	0
totinfo.c	0.9637	0.9863	0.9418	0.7778	0.8115	0.7460	-0.2682	0	-0.6371
total	0.9241	0.9863	0.7093	0.7977	0.8792	0.7460	0.28	0.8864	-0.6371

From Figure 3, we can observe that for almost all seeded versions, CCP-based test adequacy criterion has a higher PCC than others. In Table III, the average, the maximum and the minimum of the PCC with CCP-based test adequacy are all the highest among the three test adequacy criteria. The average of PCC with CCP-based coverage is 0.9241 which means the correlation between CCP-based test adequacy and the probability to find errors is very strong.

The results in our experiment approve that, CCP-based test adequacy has a very strong correlation with the capability to find errors for a test suite and this correlation is stronger than the correlations of statement coverage and branch coverage.

#### IV. RELATED WORK AND CONCLUSION

An empirical study[3] shows that coincidental correctness is prevalent and is a safety reducing factor for coverage-based fault localization. Almost all the related works[4][10][11][12] focus on the improve efficiency and accuracy of coverage-based fault localization by taming coincidental correctness. Most of existing methods to identify coincidental correctness are selecting tests from the passed tests based on the similar behaviors between coincidental correct tests and failed tests.

These methods can not identify the coincidental correctness independently and their accuracy relies on the whole test suites. Boundary value analysis also suffers from coincidental correctness. Work in [13] studies the cases where shifts in boundaries can get undetected due to coincidental correctness and proposes a solution to generate tests that not suffer from predictable coincidental correctness.

In our previous work[5][6], we propose a new test adequacy criterion based on coincidental correctness probability. In this paper, we make an improve the estimation algorithm and make a empirical study which shows that our adequacy criterion has a stronger correlated relationship with bugs found than other criteria based on the statement coverage and branch coverage.

The effect degrees of operators and the control-flow in this paper are specified according to our intuitive knowledge. In the future, we will try to find more appropriate effect degrees for different operators and the control-flow using some approaches like data mining/machine learning.

#### REFERENCES

- [1] Lionel C. Briand. Using simulation for assessing the real impact of test-coverage on defect-coverage. *IEEE Transaction on Reliability*, 49(1):60–70, 2000.
- [2] Paul Ammann and Jeff Offutt. *Introduction to software testing*. Cambridge University Press, 2008.
- [3] Wes Masri, Rawad Abou-Assi, Marwa El-Ghali, and Nour Al-Fatairi. An empirical study of the factors that reduce the effectiveness of coverage-based fault localization. In *International Workshop on Defects in Large Software Systems, DEFECTS*, pages 1–5, New York, NY, USA, 2009. ACM.
- [4] Xinming Wang, Shing-Chi Cheung, Wing Kwon Chan, and Zhenyu Zhang. Taming coincidental correctness: Coverage refinement with context patterns to improve fault localization. In *Proceedings of the 31st International Conference on Software Engineering, ICSE '09*, pages 45–55, Washington, DC, USA, 2009. IEEE Computer Society.
- [5] Jie Chen, Qian Li, Jianhua Zhao, and Xuandong Li. Test adequacy criterion based on coincidental correctness probability. In *Proceedings of the Second Asia-Pacific Symposium on Internetware*, page 20. ACM, 2010.
- [6] Jie Chen, Qian Li, Jianhua Zhao, and Xuandong Li. Test adequacy criterion based on coincidental correctness probability. *Journal of Frontiers of Computer Science And Technology*, 5(7):602–612, 2011.
- [7] John Mitchell and Monty Zukowski. A complete GNU C parser and translator. 2003.
- [8] Hyunsook Do, Sebastian G. Elbaum, and Gregg Rothermel. Supporting Controlled Experimentation with Testing Techniques: An Infrastructure and its Potential Impact. *Empirical Software Engineering*, 10:405–435, 2005.
- [9] Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. *Pearson Correlation Coefficient*, volume 2. Springer Berlin Heidelberg, 2009.
- [10] Wes Masri and Rawad Abou Assi. Cleansing Test Suites from Coincidental Correctness to Enhance Fault-Localization. In *International Conference on Software Testing, Verification, and Validation*, pages 165–174, 2010.
- [11] Yi Miao, Zhenyu Chen, Sihan Li, Zhihong Zhao, and Yuming Zhou. Identifying coincidental correctness for fault localization by clustering test cases. In *The International Conference on Software Engineering and Knowledge Engineering*, pages 267–272, 2012.
- [12] Yi Miao, Zhenyu Chen, Sihan Li, Zhihong Zhao, and Yuming Zhou. A clustering-based strategy to identify coincidental correctness in fault localization. *International Journal of Software Engineering and Knowledge Engineering*, 23(05):721–741, 2013.
- [13] Robert M. Hierons. Avoiding coincidental correctness in boundary value analysis. *ACM Transactions on Software Engineering and Methodology*, 15:227–241, 2006.

# Forwarding Links without Browsing Links in Online Social Networks

Jing Jiang<sup>§†</sup>, Xiao Wang<sup>†</sup>, Li Zhang<sup>§</sup>, Yafei Dai<sup>†</sup>

<sup>§</sup>State Key Laboratory of Software Development Environment Beihang University, Beijing, China

<sup>†</sup>Department of Computer Science and Technology, Peking University, Beijing, China

{jiangjing, lily}@buaa.pku.edu.cn, {wangxiao, dyf}@pku.edu.cn

**Abstract**—Online social networks(OSNs) become important platforms of information diffusion. People forward links pointing to interesting content, and share them with friends. Browsing links is generally considered as the previous step for forwarding links. However, few studies have confirmed this hypothesis before. The deep understanding of forwarding links can help OSNs to improve current forwarding mechanism, and allow people to read more information and forward more links.

In this paper, we study the relationship between forwarding links and browsing links. We crawled a connected graph component of 42.1 million users, and 2.1 billion links forwarded by them. Moreover, we collected browsing records of 50,000 active users. Based on these datasets, we observe that 42% of users forward more than 50% of links without browsing links. Browsing links is often unnecessary for forwarding links. This finding changes the traditional concept of link dissemination in OSNs. We also find that when users directly forward links, they have more time to leave more comments and forward more links, which promotes the spreading of links. OSNs should encourage people to read snippets and directly forward links.

**Keywords**—Forwarding links, online social networks, measurement

## I. INTRODUCTION

Online social networks have brought to the public a new style of social lives parallel to offline activities, and they have the potential to alter the way people interact with the internet. The popularity of online social networks (OSNs) makes them major platforms of information diffusion. 1 million links are shared in Facebook every 20 minutes <sup>1</sup>.

The explosive growth in OSNs has given researchers opportunity to obtain massive quantities of data for empirical analysis. Initial studies have measured the characteristics of information diffusion in Flickr [1], Facebook [2], [3], Twitter [4], [5] and other online social networks. Some works focus on people and analyze user's uploading or retweeting activities [6], [7], [8], [9].

As shown in Figure 1, normal steps of link diffusion are as follow [2]: Firstly, users primarily interact with information through an aggregated history of their friends' recent activities, which is called as the news feed. Majority of these activities contain links to content, which are identified by URLs. According to the content, snippets are automatically generated and displayed below links. People may also find links by external events, such as e-mails or instant messages. Secondly,

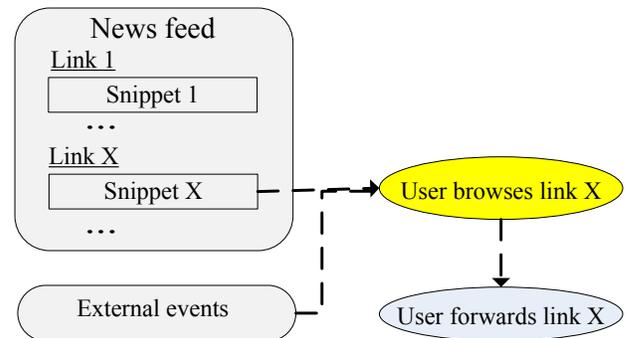


Fig. 1. Link diffusion

users read snippets and decide to browse some links. Finally, if the content is interesting, people forward links and recommend them to friends.

Search engines display snippets of search results. People sometimes find enough information from snippets, and do not take a further step to visit web pages. Since OSNs provide similar snippets, we ask the question: *Do users often forward links after browsing links, or directly forward links without visiting the links?* To our knowledge, no one has studied the relationship between browsing links and forwarding links before. Measurement results can help OSNs to improve current forwarding mechanism, and allow people to read more information and forward more links.

**Our study.** We collect and analyze large-scale traces in Renren social network <sup>2</sup>, one of the largest and most popular OSNs in China. We crawl an exhaustive snapshot of 42.1 million users and capture 2.1 billion links forwarded by these people. We also obtain browsing records of 50,000 active users. These datasets give us an opportunity to gain valuable insights in OSNs.

We provide a number of insights into the relationship between forward links and browsing links, including:

- 42% of users forward more than 50% of links without browsing links.
- From 6 AM to 8 AM, people forward 68% of links without browsing links. Users prefer to directly forward links in the morning, when they are busy at work.

<sup>1</sup><http://www.statisticbrain.com/facebook-statistics/>

<sup>2</sup><http://www.renren.com>

- When people forward links without browsing links, they leave more comments and forward more links.

Our findings change traditional concept of link dissemination: Normal steps in Figure 1 do not apply for all links. In fact, browsing links is sometime unnecessary before forwarding links. Moreover, forwarding links without browsing content is beneficial for link dissemination. It saves user's time of browsing web pages, and provide users more time to read snippets, forward links and leave comments. OSNs should generate good snippets, provide enough information, save user's time of browsing links, and allow them to forward more links.

## II. COLLECTION METHODOLOGY

Before diving into detailed measurement, we provide background information about the Renren social network. Then we introduce our collection methodology.

Renren is one of the biggest and oldest OSNs in China [10]. Renren can be best characterized as Facebook's Chinese twin, with a similar user interface and most or all of Facebook's features [11]. Users maintain personal profiles and establish bidirectional friend relationships. They publish different types of content, such as blogs, photos and albums. When people find interesting content, they forward links of the content. This function is similar to the 'share' function in Facebook. In user's news feed, Renren automatically displays snippets below links. Snippets are generated based on the content type, which is similar to the search engine. More specifically, a small part of texts are extracted for blogs, and thumbnails are generated for photos and albums.

In order to study the relationship between forwarding behavior and browsing behavior, we find opportunities to collect datasets from Renren: Firstly, people forward links in Renren, and these links are organized into completely open pages. Therefore, we collected 2.1 billion links forwarded by 42.1 million users. We use these datasets to understand forwarding behavior. Secondly, we build long-time collaboration with Renren. We directly obtain browsing records of 50,000 active users from Renren, and use them to understand browsing behavior. Different from forwarding records, browsing records are not displayed in profiles and they cannot be collected by traditional measurement techniques. Therefore, few works have studied whether browsing links is necessary before forwarding links. The collaboration with Renren provides us this special opportunity.

**Crawling forwarding records.** Friend relationships were public in the past. This allowed us to perform an exhaustive crawl of the largest connected component with 42.1 million users [10]. We seed crawlers with these 42.1 million users and proceed to crawl their links. We record the username, the link identifier, the forwarding time and the number of comments on this link. Renren labels the link's type, such as blog, photo and album. The type of link is gathered too. In total, 42,115,509 users forward 2,052,205,513 links. Some links are same, because they are forwarded for by different

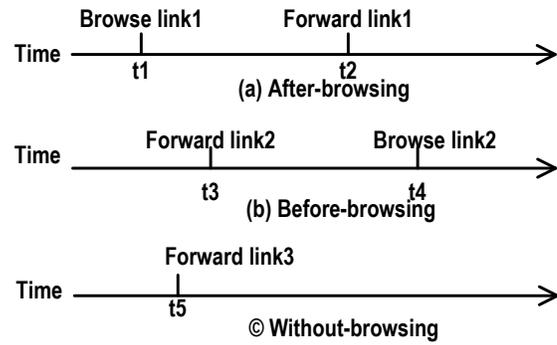


Fig. 2. Definition of three forwarding modes

people. After filtering out same links, we gather 118,228,021 unique links.

**Collecting browsing records.** Our group has built long-term collaboration with technical teams in Renren. We randomly select 50,000 active users who forwarded at least 20 links from January to June of 2012. We manually check 100 users, and none of them are spammers. Then Renren provides browsing records of 50,000 users between January and June of 2012. The browsing record includes the username, the link identifier and the browsing time. These datasets provide us an opportunity to gain valuable insights of user behavior.

We focus on user behaviors and do not need any actual content. Therefore, we wait for crawls to complete, then go through data to anonymize user IDs and strip any private data from our datasets to protect user privacy. Moreover, all user IDs are hashed to random IDs. link addresses are mapped to hash values. All timestamps are replaces with relative sequence numbers.

## III. FORWARDING LINKS WITHOUT BROWSING LINKS

In user's news feed, OSN automatically displays snippets of links recommended by friends. People read snippets and choose to visit some web pages pointed by links. Then people find some interesting web pages and forward their links. Figure 1 describe these normal steps of link dissemination. However, snippets already provide summary of web pages, and we doubt whether users must browse links before forwarding them. We ask the question: Do users often forward links after browsing links, or directly forward links without visiting links? In order to answer this question, we study the relationship between forward links and browsing links.

### A. Definition and Basic analysis

As shown in Figure 2, we classify forwarding behaviors into three modes:

- After-browsing: The user browses the content of the link, and then forwards the link. This mode is considered as normal.
- Before-browsing: The user forwards the link, and visits the link later. When the user reads the interesting snippet, he or she may not have enough time to browse the link

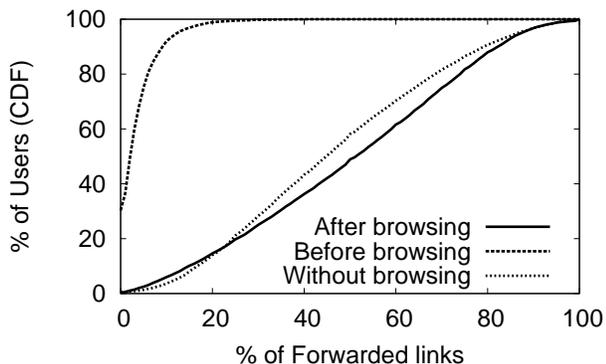


Fig. 3. Distribution of forwarding modes

immediately. Therefore, the user forwards the link, and later visits the link.

- Without-browsing: The user directly forwards the link, without browsing the link. If snippet provides enough information, the user immediately shares the link with friends, but never visits the original web page.

If the user browses the same link for several times, we consider the first visit here. For example, a user browses the link at first, then forwards the link, and later browses the link again. This forwarding behavior is still classified as the after-browsing mode.

As described in section II, our datasets include forwarding records and browsing records of 50,000 active users. We use these datasets and classify forwarding behaviors into appropriate forwarding modes. According to username and the link identifier, we match forwarding record and browsing record. If the forwarding record does not match any browsing record, it means that the link is directly forwarded without browsing content. Therefore, the forwarding activity belongs to without-browsing mode. If the forwarding record matches a browsing record, then we compare the forwarding time and the browsing time. If the forwarding time is before the browsing time, the forwarding activity belongs to before-browsing mode, otherwise the forwarding activity belongs to after-browsing mode.

For every user, we compute the percentage of links forwarded in each mode. Then we aggregate all users the percentage and plot results in Figure 3. 49% of users forward less than 50% of links after browsing links, while other 51% of users forward more than 50% of links after browsing links. The after-browsing mode is common in online social networks. In contrast, the forwarding before browsing mode is rare. 92% of users forward less than 10% of links before browsing links. Surprisingly, 42% of users forward more than 50% of links without browsing links. This important finding shows that the forwarding without browsing mode is common in online social networks. Some forwarding activities do not follow normal steps. Users read snippets of links, and then directly share links without browsing links. This finding has significant implication: Snippet generation becomes important in the forwarding mechanism. Good snippets can provide

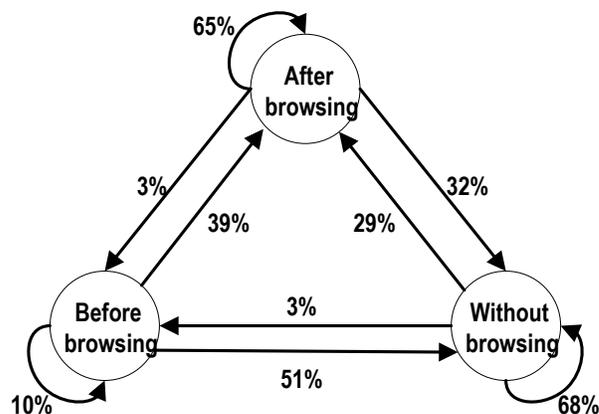


Fig. 4. Transition probability between forwarding modes

enough information and replace content pointed by links.

### B. Transitions between different modes

In this subsection, we take a further step and explore transitions between different modes. We ask the question: If the user shares the link in a forwarding mode, will this user forward the next link in the same mode? For every user, we compute the transition probability from a forwarding mode to a forwarding mode. For example, the user forwards 5 links, and their forwarding modes are: after-browsing, after-browsing, without-browsing, after-browsing, without-browsing. The user has 3 after-browsing behaviors. 1 after-browsing behavior is followed by the after-browsing behavior, and 2 after-browsing behaviors are followed by without-browsing behaviors. Therefore, the transition probability from the after-browsing mode to the after-browsing mode is 33%, and the transition probability from the after-browsing mode to the without-browsing mode is 67%. Since the user does not forward links before browsing links, we do not compute the transition probability from the before-browsing mode to any modes.

For all users, we calculate these transition probabilities. Then we compute average value of all users and plot results in Figure 4. The transition probability from the after-browsing mode to the after-browsing mode is 65%. If people forward links in the after-browsing mode, they are likely to forward next links in the after-browsing mode. The transition probability from the without-browsing mode to the without-browsing mode is 68%. Results show that both after-browsing and without-browsing activities are stable. When people are free, they may continually forward links after they browse links; when they are busy, they may also successively forward links without browsing links. The before-browsing mode has 39% probability of moving to the after-browsing mode, and 51% probability of moving to the without-browsing mode. The before-browsing mode is unstable, and quickly moves to other modes.

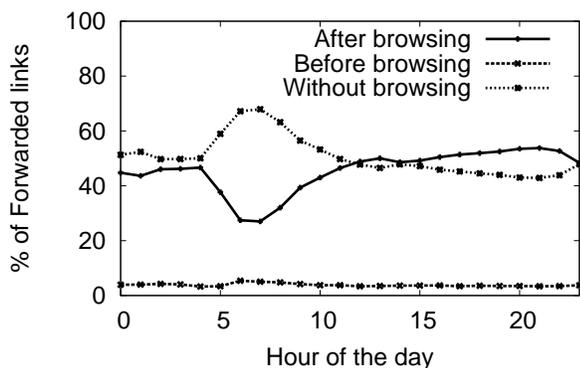


Fig. 5. The daily distribution of forwarded links

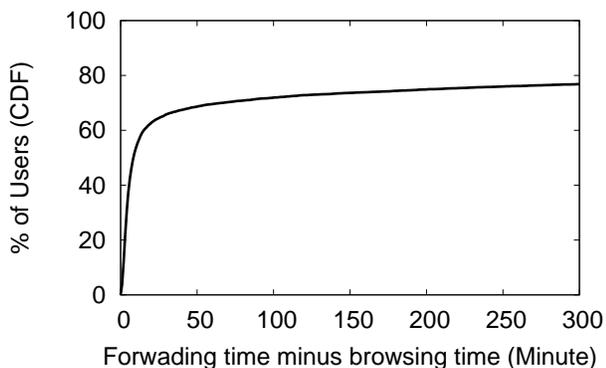


Fig. 6. Interval time for links which are forwarded after browsed

### C. Time Distribution

Next, we study the time distribution of forwarding modes. First of all, we study daily patterns of links forwarded in different modes. We group links by the hour of day, and plot the percentage of links forwarded in each mode in Figure 5. For example, 0 in the horizontal axis stands for links forwarded between 0 AM and 1 AM. From 0 to 4, the percentage of links in the without-browsing mode is slightly larger than that of links in the after-browsing mode. From 5 to 11, the percentage of links in the without-browsing mode is much larger than that of links in the after-browsing mode. More specifically, people forward 68% of links in the without-browsing mode from 6 AM to 8 AM, While people only forward 28% of links in the after-browsing mode from 6 AM to 8 AM. From 12 to 17, the percentage of links in the without-browsing mode is slightly smaller than that of links in the after-browsing mode. From 17 to 23, the difference value between without-forwarding curve and after-forwarding curve is bigger than 5%. The percentage of links in the before-browsing mode is the smallest all the day. We guess this phenomenon is probably because that people are busy at work in the morning, and they prefer to quickly read snippets and forward links directly. People are free in the evening and have enough time. They prefer to browse links and forward links. The time of the day has the significant effect on forwarding modes.

Some links are are forwarded after browsed. For these links, the interval time is calculated as the forwarding time minus

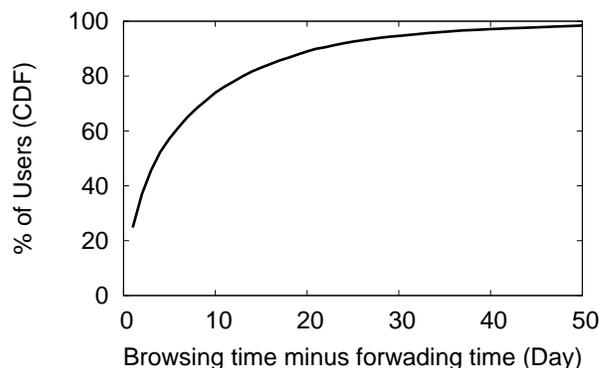


Fig. 7. Interval time for links which are forwarded before browsed

Forwarding mode	Number of comments
After-browsing	0.15
Before-browsing	0.18
Without-browsing	0.27

TABLE I  
AVERAGE NUMBER OF COMMENTS PER LINK

the browsing time. For example, the interval time is  $t_2 - t_1$  in Figure 2 (a). For every user, we compute the average interval time of all links which are forwarded after browsed. Then we aggregate all users the interval time and plot results in Figure 6. 63% of users have the interval time less than 20 minutes, but 23% of users have the interval time more than 300 minutes (5 hours). A majority of people forwards links immediately after they browse links, while a small part of people share links a long time after they visit links. This minority of users do not forward links immediately after they browse links. Their activities are also strange and need further study in future work.

Some links that are forwarded before browsed. For these links, the interval time is calculated as the browsing time minus the forwarding time. In Figure 2 (b), the interval time is  $t_4 - t_3$ . Figure 7 shows the CDF of interval time for links that are shared before visited. 25% of users have the interval time less than 1 day, but 26% of users have the interval time more than 10 days. People may forward links to make bookmarks. They firstly forward links and add links as favorites. Then they visit links when they are free.

### D. Activity

In this subsection, we analyze activities related to different forwarding modes. When people forward links, they sometimes write comments to express opinions and attract friends' attention. For every user, we compute the number of comments for links forwarded in each mode, divided by the number of links forwarded in each mode. The result is the average number of comments per link in each mode. Then we compute the average value of all users and plot results in Table I. For links in the after-browsing mode, people only leave the 0.15 comments per link. For links in the before-browsing mode, people leave 0.18 comments per link. For links in the without-browsing mode, people leave 0.27 comments per link, which is

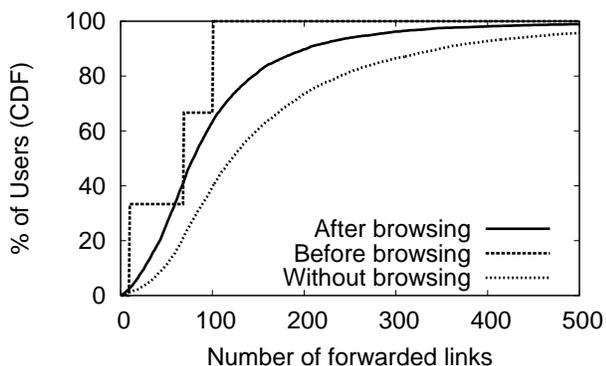


Fig. 8. Number of forwarded links for different types of users

nearly twice as the value of links in the after-browsing mode. In the without-browsing mode, people do not spend time in browsing links, but have more time to express opinions and leave comments. In news feed, comments are displayed in links' snippets and attract friends' interest. Results show that the without-browsing mode promotes the spreading of links in OSNs.

Next, we explore how forwarding modes impact the frequency of forwarding behaviors. For every user, we compute the percentage of links forwarded in each mode. We find the maximum percentage and the corresponding forwarding mode. Then we classify the user into the type of maximum forwarding mode. For example, the user forwards most of links in the without-browsing mode, and the user is classified as the without-browsing type. For users in each type, we plot the number of forwarded links in Figure 8. People seldom forward links before browsing, and only 3 people are classified into the before-forwarding type. For the after-browsing type, 63% of users forward less than 100 links, and other 37% of users forward more than 100 links. For the without-browsing type, only 39% of users forward less than 100 links, and other 61% of users forward more than 100 links. Users in the without-forwarding type share more links than users in the after-forwarding type. When people prefer to forward links directly, they spend little time in visiting links, and have additional time to read more snippets and share more links with friends. Results further show that the without-browsing mode is beneficial for the link dissemination in OSNs.

### E. Content type

Links point to different types of content. We wonder whether people prefer to forward links belonging to the same type. 97.8% of links belong to blogs, photos or albums, while only 2.2% of links belong to other types. Therefore, we focus on these 3 types and analyze their impacts on forwarding modes.

First of all, we examine whether people mainly forward a certain type of links. For every user, we rank the type based on the number of links forwarded in each type, and calculate the percentage of links in the biggest type. For example, a user forwards 3 blogs, 10 photos and 7 album. The biggest type is

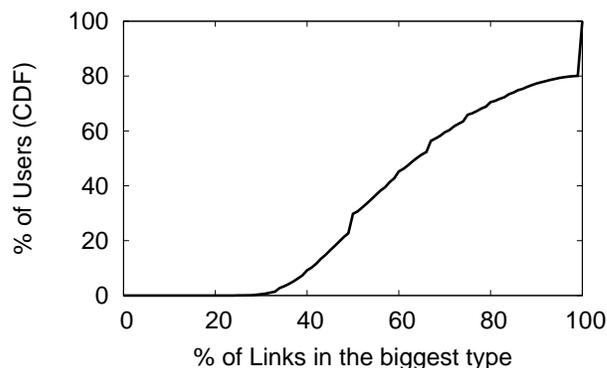


Fig. 9. Concentration of content type

Type	% of Users
Blog	80%
Photo	12%
Album	8%

TABLE II  
THE DISTRIBUTION OF THE BIGGEST TYPE

photo, and the percentage of links in the biggest type is 50%. We compute the percentage of forwarded links in the biggest type, and plot results in Figure 9. 30% of users forward less than 50% of links in the biggest type, while other 70% of users forward more than 50% of links in the biggest type. People concentrate links into the biggest type.

We take a further step and find the biggest type for every user. Then we calculate the percentage of users who like each type most and plot results in Table II. The blog is the most popular type. 80% of people forward most of links pointing to blogs. 12% of users prefer to spread photos, and other 8% of users like albums most. In order to satisfy user's interests, OSNs can identify user's favorite type and recommend links belonging to the corresponding type.

Finally, we study the distribution of forwarding modes in various content types. For every user, we compute the number of blogs' links forwarded in the after-browsing mode, divided by the total number of blogs's links of this user. We also compute the percentage of blogs' links in other forwarding modes. Then we calculate the average value of all users. For links pointing to photos and albums, we also compute the percentage of links forwarded in each forwarding mode, and plot results in Figure 10. For links pointing to blogs, 45% of links are forwarded after browsed, while 52.3% of links are forwarded without browsing content. When people forward blogs, they are more likely to directly read snippets and forward links. For links pointing to photos, 57.8% of links are in the after-browsing mode, while 36.8% of links are in the without-browsing mode. Users are more likely to visit links and then forward links. For links pointing to albums, the percentage of links in the after-browsing mode is as high as 66.5%. We compare different types and observe that people mostly prefer without-browsing mode for blogs, and like after-browsing mode for albums. The type of content has obvious

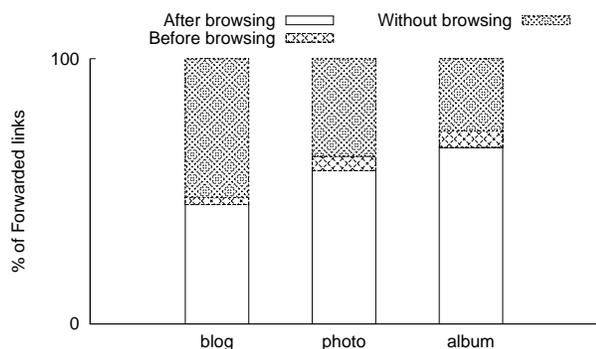


Fig. 10. Distribution of forwarding modes in the content type

impacts on forwarding activities.

#### IV. RELATED WORK

Much effort has been put into understanding of information propagation in online social networks. Photo's spread in Flickr [1], [12], dissemination of pages in Facebook [13], [3], product adoption in Instant Messaging [14], individual invitation in games [15], 'gesture' diffusion in Second Life [16], linking pattern through the blog [17], propagation of tweets in Twitter [4], [5] and forwarding emails in a social sensor system [18] all report on information dissemination. These works mainly explore characteristics of information cascades, such as dissemination scope and speed.

Some studies analyze user's uploading or forwarding behaviors. Ding et al. examine the uploading behavior of YouTube users [6]. Yang et al. understand retweeting behaviors in Twitter [7]. Previous works [8], [9] compare different kinds of content recommendation, and study their effects on user's forwarding behaviors.

We measure the relationship between forwarding behaviors and browsing behaviors. To our knowledge, this is the first study of forwarding links without browsing links in OSNs.

#### V. CONCLUSIONS

In this paper, we study the relationship between forwarding links and browsing links. We observe that 42% of users forward more than 50% of links without browsing links. People sometimes read snippets and directly forward links, without browsing links. This finding changes the traditional concept of link dissemination in OSNs. We take a further step and observe characteristics of the without-browsing mode: 1) Users prefer to directly forward links in the morning, when they are busy at work. 2) In the without-browsing mode, people leave more comments and forward more links. These findings suggest that without-browsing mode is beneficial for link dissemination. Users do not spend time to browse links, and have more time to read snippets and forward links, especially when they are busy at work. It implies that snippet generation becomes important in the forwarding mechanism. OSNs should generate good snippets to provide enough information of links, save user's time of browsing links, and encourage the without-browsing mode.

#### ACKNOWLEDGMENT

This work is supported by the National Science Foundation of China under the National Basic Research Program of China grant No. 2011CB302305, National Natural Science Foundation of China under Grant No.61300006, and the State Key Laboratory of Software Development Environment under Grant No.SKLSDE-2013ZX-26.

#### REFERENCES

- [1] M. Cha, A. Mislove, and K. P. Gummadi, "A measurement-driven analysis of information propagation in the flickr social network," in *Proc. of World Wide Web Conference*, Madrid, Spain, April 2009.
- [2] E. Bakshy, I. Rosenn, C. Marlow, and L. Adamic, "The role of social networks in information diffusion," in *Proc. of WWW*, Lyon, France, April 2012.
- [3] M. S. Bernstein, E. Bakshy, M. Burke, and B. Karrer, "Quantifying the invisible audience in social networks," in *Proc. of CHI*, Paris, France, April 2013.
- [4] F. Toriumi, T. Sakaki, K. Shinoda, K. Kazama, S. Kurihara, and I. Noda, "Information sharing on twitter during the 2011 catastrophic earthquake," in *Proc. of WWW Companion*, Rio de Janeiro, Brazil, May 2013.
- [5] K. Lee, J. Mahmud, J. Chen, M. Zhou, and J. Nichols, "Who will retweet this? automatically identifying and engaging strangers on twitter to spread information," in *Proc. of IUI*, Haifa, Israel, February 2014.
- [6] Y. Ding, Y. Du, Y. Hu, Z. Liu, L. Wang, K. W. Ross, and A. Ghose, "Broadcast yourself: Understanding youtube uploaders," in *Proc. of ACM Internet Measurement Conference*, Berlin, Germany, November 2011.
- [7] Z. Yang, J. Guo, K. Cai, J. Tang, J. Li, L. Zhang, and Z. Su, "Understanding retweeting behaviors in social networks," in *Proc. of CIKM*, Toronto, Canada, October 2010.
- [8] J. Chen, R. Nairn, L. Nelson, M. Bernstein, and E. H. Chi, "Short and tweet: Experiments on recommending content from information streams," in *Proc. of CHI*, Atlanta, USA, April 2010.
- [9] S. A. Macskassy and M. Michelson, "Why do people retweet? anti-homophily wins the day!" in *Proc. of International AAAI Conference on Weblogs and Social Media*, Barcelona, Spain, July 2011.
- [10] J. Jiang, C. Wilson, X. Wang, P. Huang, W. Sha, Y. Dai, and B. Y. Zhao, "Understanding latent interactions in online social networks," in *Proc. of ACM Internet Measurement Conference*, Melbourne, Australia, November 2010.
- [11] C. Wilson, B. Boe, A. Sala, K. P. N. Puttaswamy, and B. Y. Zhao, "User interactions in social networks and their implications," in *Proc. of ACM Eurosys*, Nuremberg, Germany, March 2009.
- [12] M. Cha, A. Mislove, B. Adams, and K. P. Gummadi, "Characterizing social cascades in flickr," in *ACM SIGCOMM WOSN*, Seattle, USA, August 2008.
- [13] E. Sun, I. Rosenn, C. A. Marlow, and T. M. Lento, "Gesundheit! modeling contagion through facebook news feed," in *Proc. of International AAAI Conference on Weblogs and Social Media*, San Jose, USA, May 2009.
- [14] R. Bhatt, V. Chaoji, and R. Parekh, "Predicting product adoption in large-scale social networks," in *Proc of ACM Conference on Information and Knowledge Management*, Toronto, Canada, October 2010.
- [15] X. Wei, J. Yang, and L. A. Adamic, "Diffusion dynamics of games on online social networks," in *Proc. of the 3rd Workshop on Online Social Networks*, Boston, USA, June 2010.
- [16] E. Bakshy, B. Karrer, and L. A. Adamic, "Social influence and the diffusion of user-created content," in *Proc. of ACM conference on Electronic commerce*, Stanford, USA, July 2009.
- [17] J. Leskovec, M. McGlohon, C. Faloutsos, N. Glance, and M. Hurst, "Cascading behavior in large blog graphs patterns and a model," in *SIAM International Conference on Data Mining*, Minneapolis, USA, April 2007.
- [18] D. Wang, Z. Wen, H. Tong, C.-Y. Lin, C. Song, and A.-L. Barabasiand, "Information spreading in context," in *Proc. of World Wide Web Conference*, Hyderabad, India, April 2011.

# Accurate Local Estimation of Geo-Coordinates for Social Media Posts

Derek Doran and Swapna S. Gokhale  
Dept. of Computer Science & Eng.  
University of Connecticut  
Storrs, CT, 06269  
{derek.doran,ssg}@engr.uconn.edu

Aldo Dagnino  
Industrial Software Systems  
ABB Corporate Research  
Raleigh, NC, 27606  
aldo.dagnino@us.abb.com

## Abstract

*Associating geo-coordinates with the content of social media posts can enhance many existing applications and services and enable a host of new ones. Unfortunately, a majority of social media posts are not tagged with geo-coordinates. Even when location data is available, it may be inaccurate, very broad or sometimes fictitious. Contemporary location estimation approaches based on analyzing the content of these posts can identify only broad areas such as a city, which limits their usefulness. To address these shortcomings, this paper proposes a methodology to narrowly estimate the geo-coordinates of social media posts with high accuracy. The methodology relies solely on the content of these posts and prior knowledge of the wide geographical region from where the posts originate. An ensemble of language models, which are smoothed over non-overlapping sub-regions of a wider region, lie at the heart of the methodology. Experimental evaluation using a corpus of over half a million tweets from New York City shows that the approach, on an average, estimates locations of tweets to within just 2.15km of their actual positions.*

## 1 Introduction

The information shared by users over Online Social Networks (OSNs) such as Facebook and Twitter offer unique insights into their thoughts, emotions, and opinions. The richness of these posts has motivated numerous organizations to harvest the content embedded within them in support of value-added services. Associating geographic locations with the information extracted from these posts can offer both theoretical and practical benefits. Theoretically, this association can facilitate sociological studies to examine how online behaviors, relationships, and interactions are influenced by their offline socio-spatial counterparts [17]. Practically, the linking of location to content can enhance existing services such as location-based advertising [12]

and disaster response [20] and conceive novel ones.

Social media posts may be tagged with location information in two ways. First, users may choose to automatically tag their posts shared via GPS-enabled mobile devices. Second, many OSNs allow users to include their current location [16] through fields such as “location” or “from” in their social media profiles. If users diligently and authentically use one of these two methods, then accurate location information can be extracted easily. However, currently, a vast majority of users do not enable tagging of their mobile posts [4] and choose not to include their locations in their profiles, perhaps for privacy reasons. Some users who do populate this field may specify it broadly in terms of a state or a country, while some may intentionally provide inaccurate or fictitious positions [9]. Thus, in practice, only a small percentage of social media posts are accompanied by rich and accurate location data. To alleviate this shortcoming, contemporary approaches that need the location of posts estimate it by analyzing their content. These approaches, however, estimate broad regions of the order of a city, or location “types” such as restaurants, offices, homes, or stores [6, 13]. Finally, a few efforts that try to estimate the actual positions or geo-coordinates of social media posts are accurate within a radius of 80-100 km [4, 14, 5], essentially identifying only broad regions.

Once the broad region from where a social media post has originated is identified either through tagging, by finding keywords corresponding to famous landmarks and interesting events, or by using an aforementioned contemporary approach, pinning it down narrowly to within a small radius around its actual geo-coordinates may add significant value. For example, such local geo-tagging can shed light on how people within a neighborhood think similarly as they are exposed to common events, and participate in richer and meaningful offline friendships [15]. Identifying localities can also provide more accurate information on an event or a disaster which can be highly beneficial to first responders [20]. Law enforcement can also use such fine geo-tagging to approximate the location of a suspect, who

is known to be present in a town or a city. Finally, it may be feasible to identify the geo-coordinates of a post with high accuracy once its broad region is known because this prior knowledge limits the range of possible positions.

In this paper, we present a methodology to accurately estimate the geo-coordinates of a social media post based on its content, once the broad region from where it originates is known. The methodology consists of partitioning the broad region into a grid of non-overlapping sub-regions, building probabilistic language models over each sub-region, and then applying geo-smoothing to improve the accuracy of the location estimates. We train and evaluate the language models over a corpus of tweets collected across downtown and midtown Manhattan, and find that the approach, on an average, pinpoints the positions of tweets to within 2.15km, or just 4% of the size of the total region from which they are known to originate.

This paper is organized as follows. Section 2 describes our estimation methodology. Section 3 presents experimental evaluation. Related work is compared in Section 4. Conclusions and future work are offered in Section 5.

## 2 Estimation Methodology

In this section, we describe the two steps in the methodology to estimate geo-coordinates.

### 2.1 Building Language Models

The topics, thoughts, words, and expressions embedded within social media posts are influenced by the inherent properties and circumstances of the locality from where users share these posts. For example, people may share their opinions of a restaurant while seated there. They may also share about an accident or a noteworthy public event as it occurs. The culture and social norms of a local area may also modulate these posts. As an example, posts from Little Italy in New York City may be pre-dominantly influenced by Italian norms and culture, while those from Times Square may instead overwhelmingly share the excitement of visiting the city. Thus, both the language and the content of social media posts shared from different, smaller sub-regions within a broad region will be varied.

To expose these local variations, we partition a broad region of interest  $\mathcal{L}$  into a collection of equally sized, non-overlapping sub-regions  $\ell_i$ , which are defined by a  $g \times g$  grid. We then build an ensemble of models; one per sub-region to represent the language of its posts. This ensemble is inspired by recent approaches including our own [7, 8], that have demonstrated its promise in capturing the linguistic variations in the content of social media posts. A language model defines a probability distribution over  $n$ -grams, where an  $n$ -gram is an ordered sequence of  $n$  words

$(w_1, \dots, w_n)$ . The maximum likelihood estimate of an  $n$ -gram, computed over a corpus of posts within  $\ell_i \in \mathcal{L}$ , is given by [1]:

$$P_{\ell_i}(w_1, \dots, w_n) = \frac{c_{\ell_i}(w_1, \dots, w_n)}{c_{\ell_i}(w_1, \dots, w_{n-1})}$$

where  $c(\cdot)$  is the number of times the sequence appears in the posts. The probability that a sub-region generates a phrase  $T = (w_1, \dots, w_k)$  is computed as the product of the probabilities of the  $n$ -grams that comprise  $T$ :

$$P(T|\ell_i) = \prod_{j=1}^{k-n+1} P_{\ell_i}(w_j, w_{j+1}, \dots, w_{j+n-1})$$

Contextual information increases with  $n$  because longer sequences of words can be considered. However, because social media posts are short, specific long word sequences appear with low frequency, and hence, prevalent approaches use only unigrams to model these posts. Although unigrams or 1-grams model the distinct vocabulary, independent of the order of words [2], they lack the ability to capture context within a language. For example, a unigram model trained over “going to work” can represent how one discusses the concept of “work”, and the action of “going”, but cannot associate the concept with the action. Language models trained over bigrams “going to” and “to work”, however, can capture additional context of going somewhere, and applying an action or a verb to the concept of “work”. We limit to bigrams although higher order models can capture even more details, because estimating higher order models may be inaccurate using a corpus of social media posts that are typically short but refer to a broad variety of topics.

To improve the accuracy of the language models, we interpolate the probability of a bigram with the probability of the unigram that completes it. This interpolation compensates for the low count of a bigram by incorporating the expected higher count of the unigram that completes it. For example, if the unigram “driving” is used frequently in a training corpus, we should expect that bigrams completed by this word (e.g. “love driving”) are more likely to be seen even if the bigram does not appear often. Thus, for a sub-region  $\ell_i$ , the probability of observing the bigram  $(w_{j-1}, w_j)$  is given as:

$$P_{\ell_i}(w_{j-1}, w_j) = \lambda_1 \frac{c(w_{j-1}, w_j)}{c(w_{j-1})} + \lambda_2 \frac{c(w_j)}{|W(\ell_i)|}$$

where  $\lambda_1 + \lambda_2 = 1$ ,  $|W(\ell_i)|$  is the number of distinct words in all posts in  $\ell_i$  and  $c(w_j)/|W(\ell_i)|$  is the estimate of the unigram that completes the bigram [1].

We further compensate the language model to account for future unseen bigrams by diverting some of the prob-

ability of the training bigrams to those that are as yet unobserved. We use the Modified Kneser-Ney (MKN) algorithm [11] for this compensation because it offers the best performance for interpolated language models [3]. The MKN algorithm subtracts a constant  $\hat{d}$  from the observed frequency of every known bigram. It then estimates the likelihood that an unknown bigram  $(w_{j-1}, w_j)$  will appear with a modified estimate of the unigram  $w_j$ , where only the number of *distinct bigrams* that  $w_j$  completes is considered:

$$P_c(w_j) = \frac{|\{w : c(w, w_j) > 0\}|}{\sum_v |\{w : c(w, v) > 0\}|}$$

$P_c(w_j)$  is then weighted by the probability mass  $\lambda(w_{j-1})$  that is taken by subtracting  $\hat{d}$  from the counts of known bigrams:

$$\lambda(w_{j-1}) = \frac{\hat{d}|\{w : c(w_{j-1}, w) > 0\}|}{c(w_{j-1})}$$

Thus, under the MKN algorithm the probability of observing a bigram becomes:

$$P_{\ell_i}(w_{j-1}, w_j) = \frac{\max(c(w_{j-1}, w_j) - \hat{d}, 0)}{c(w_{j-1})} + \lambda(w_{j-1})P_c(w_j)$$

If  $(w_{j-1}, w_j)$  is unknown, the probability is just given by  $\lambda(w_{j-1})P_c(w_j)$ , and if it is known, the probability is given as a linear interpolation of the modified bigram and unigram estimates. Note that the modified unigram estimate  $P_c(w_j)$  is superior to  $c(w_j)/|W(\ell)|$  because under  $P_c(w_j)$ , words that appear frequently but within few distinct contexts will not strongly influence the probability of the bigram. We estimate  $\hat{d}$  such that the log-likelihood that the model generates a given bigram is maximized:

$$\hat{d} = \arg \max_d \sum_v c(v, w_j) \log P_\ell(v, w_j)$$

This has a closed form approximation depending on whether  $c(w_{i-1}, w_i)$  is equal to 1, 2, or  $\geq 3$  [19]. Using these approximations, we set  $\hat{d}$  equal to  $d_1, d_2$ , or  $d_3$  respectively:  $d_1 = 1 - (2n_2/(n_1 + 2n_2))$ ,  $d_2 = 2 - (3n_3n_1/(n_2(n_1 + 2n_2)))$ , and  $d_3 = 3 - (4n_4n_1/(n_3(n_1 + 2n_2)))$  where  $n_i$  is the number of bigrams that appear with frequency  $i$ . Subsequently, we define the probability that a social media post  $T$  is generated from a sub-region  $\ell_i$   $P(T|\ell_i)$  as:

$$P(T|\ell_i) = \prod_{j=2}^k P_{\ell_i}(w_{j-1}, w_j)$$

## 2.2 Estimating Geo-Coordinates

After training the language models over tweets from each sub-region, the ensemble is queried to compute the

probability that a social media post  $T$  is generated from a sub-region  $\ell_i$  using Bayes rule:

$$P(\ell_i|T) = \frac{P(T|\ell_i)P(\ell_i)}{\sum_j P(T|\ell_j)P(\ell_j)}$$

$P(\ell_i)$  is the prior probability that a social media post is from sub-region  $\ell_i$  and is given by  $N(\ell_i)/N(\mathcal{L})$ .  $N(\ell_i)$  is the number of posts in  $\ell_i$  and  $N(\mathcal{L})$  is the total number of posts in the entire city  $\mathcal{L}$ . The geo-coordinates of a post  $T$  may be estimated as the center of the sub-region whose posterior probability  $P(\ell_i|T)$  is the highest, that is, we may choose the center of  $\ell^*$  where  $\ell^* = \arg \max_i P(\ell_i|T)$ .

Previous works suggest that the proximity to an object increases the propensity of the users to post about it [6, 18]. In other words, it is feasible that the language of a sub-region may be influenced by the landmarks and events within its neighboring sub-regions. Thus, although we can naively use the highest  $P(\ell_i|T)$  to estimate the geo-coordinates of a post, we introduce a *geo-smoothing* function  $\Theta^\circ(\ell_i|T)$ , which combines the posterior probabilities of the neighboring sub-regions to capture their influence on the language in  $\ell_i$ . Based on this geo-smoothing function, we select  $\ell^*$  as  $\ell^* = \arg \max_i \Theta^\circ(\ell_i|T)$ . Popular functional forms for  $\Theta^\circ$  include a decay component that reduces the contribution of neighbors as they get increasingly away from  $\ell_i$  [13]. Such geo-smoothing performs best when the decay component takes a polynomial form [6, 18]. Thus, in this preliminary study, we consider the simplest polynomial shown to be effective in geo-locating documents [18]. Letting  $\Omega_k(\ell_i)$  be the set of neighbors of  $\ell_i$  whose distance is  $k$  cells away, and  $P_{\ell_i}(T) = P(\ell_i|T)$ , our geo-smoothing function is defined as:

$$\Theta^\circ(P_{\ell_i}(T); \alpha, d) = (1 - \alpha)P_{\ell_i}(T) + \alpha \sum_{k=1}^d \sum_{\omega \in \Omega_k(\ell_i)} \frac{P_\omega(T)}{(2k+1)^2 - 1}$$

where  $\alpha \in [0, 1]$  is the smoothing weight and  $d$  is smoothing diameter, that is, the largest distance from which a neighbor can be located.

It is important to note that the accuracy of the estimated geo-coordinates is limited to the resolution of the  $g \times g$  grid chosen to divide the region into sub-regions. Increasing  $g$  will decrease the size of the sub-regions and allow for more accurate estimation, however, the number of posts available within each sub-region may be insufficient to train the models. On the other hand, decreasing  $g$  increases the size of individual sub-regions so that they contain more posts, but limits the estimation accuracy. Similarly, increasing  $\alpha$  and  $d$  respectively increase the importance and number of the neighboring sub-regions. If  $\alpha$  and  $d$  are very high, neighboring sub-regions may dwarf the candidate sub-region. However, if they are too low, they may not adequately capture

users’ reactions on the local events and objects. We empirically choose the values of the three hyperparameters  $g$ ,  $\alpha$  and  $d$  to balance these competing concerns.

### 3 Experimental Evaluation

In this section, we describe the data, its pre-processing, hyperparameter fits, and evaluation results.

#### 3.1 Data Pre-Processing

We collected over half-million geo-tagged tweets using Twitter’s Streaming API <sup>1</sup> across New York City over a three-month period (January 29th - April 7th, 2013). The tweets were collected across a 51.44km<sup>2</sup> region that includes downtown and midtown Manhattan because it includes popular residential and commercial districts as well as tourist destinations. We expect that because of this diversity tweets from this region will capture varied thoughts representing the perspectives of long-term city residents, commuters, and visitors.

For every tweet, we eliminated all non-English words and characters. We also eliminated hashtags because although they may indicate topic and content, they may also include shorthand or concatenated words (i.e. “#WestEnd”) that the language models cannot decipher. We further pre-processed the tweets by converting all words to lowercase and by stripping punctuation, username replies, and links to Web pages. We also produced a stopwords list of the 200 most frequently used words such as “at”, “the”, and “or”, which lack contextual information, and hence, introduce noise into the estimation of bigrams. We choose a limited stopwords list that is approximately equal to 1% of the number of distinct words across the data collection region. We also include a “catch all” unigram “<misc>” to aggregate the probability of words that occur only once. This term thus accounts for the many miscellaneous, shorthand, mis-spelled, and other user-specific notations that are uniquely common to Twitter. Of the 574,948 tweets, we reserved 408,095 (70%) for training the language models, 83,990 (15%) for evaluation, and another 82,863 (15%) as hold-out data for fitting the hyperparameters.

#### 3.2 Hyperparameter Fits

We find values for the three hyperparameters, namely, the grid size  $g$ , smoothing weight  $\alpha$ , and smoothing diameter  $d$  such that the average estimation error of  $\Theta^\circ$  across the set of hold-out tweets is minimized. We define the average estimation error as the geo-distance (in km) between

<sup>1</sup><https://dev.twitter.com/docs/api/1.1/post/statuses/filter>

the actual GPS coordinates of a tweet to the center of the sub-region that the model estimates it is from. The parameters were fit via a standard grid search where  $\alpha$  and  $d$  were varied within their range of possible values, namely,  $\alpha \in \{0.1, 0.2, \dots, 1.0\}$  and  $d \in \{1, \dots, g\}$ . We chose to vary  $g \in \{5, 6, \dots, 15\}$  because the estimation error increased when  $g$  was outside this range regardless of  $\alpha$  and  $d$ .

As we simultaneously varied  $\alpha$ ,  $d$ , and  $g$  in these ranges, we found that irrespective of  $g$  and  $d$ ,  $\alpha = 0.9$  consistently minimized the estimation error. In other words, only 10% of the posterior probability that a tweet  $T$  originated from a sub-region  $\ell_i$  can be attributed to its own language model  $P_{\ell_i}(T)$ , while the rest is contributed by the language models of the neighboring sub-regions. Furthermore, for every value of  $g$ , estimation error over the hold-out set is minimized at  $d = g$ . We thus set the grid size parameter  $g$  equal to  $d$  and  $\alpha = 0.9$ . Figure 1 shows the mean error for different values of  $g$ . We achieve the best performance across the hold-out data when we set  $g = 8$ , where the city is partitioned into 64 sub-regions each with an area 0.803km<sup>2</sup>. With  $g$  set to 8 and  $\alpha = 0.9$ , we then estimate the parameters of the interpolated bigram language models using the method described in Section 2.

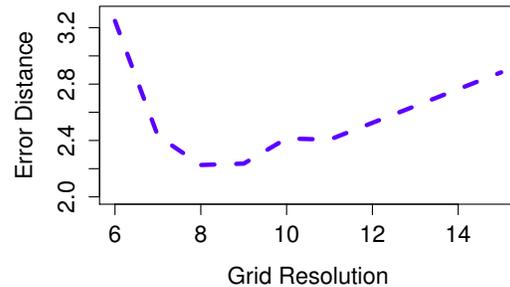


Figure 1. Mean Hold-out Errors –  $\alpha = 0.9, d = g$

#### 3.3 Evaluation Results

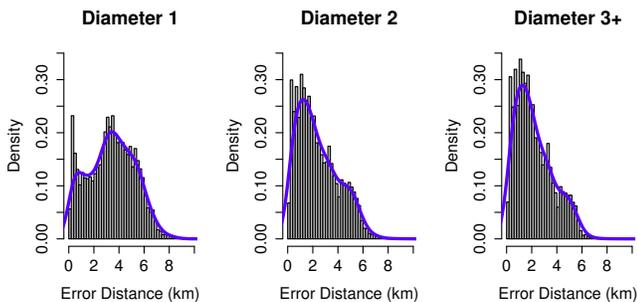
We experimentally explore the influence of  $d$  on the overall estimation error over the test set comprising 82,863 tweets. Table 1 shows that the average estimation error decreases as  $d$  rises. Thus, even a simple polynomial function can appropriately decay probabilities from sub-regions as they get farther away from  $\ell_i$  and significantly enhance estimation accuracy. We note that the mean error does not change for  $d \geq 5$  because as the diameter of geo-smoothing increases to include sub-regions  $d = g/2$  cells away, central sub-regions in the city begin to consider almost every other sub-region in its set of neighbors. Further increases in  $d$  thus do not change  $P_{\ell_i}(T)$  for a growing number of

sub-regions, causing the mean estimation error to converge.

Diameter	1	2	3	4	5+
Mean Error	3.38	2.39	2.18	2.16	2.15

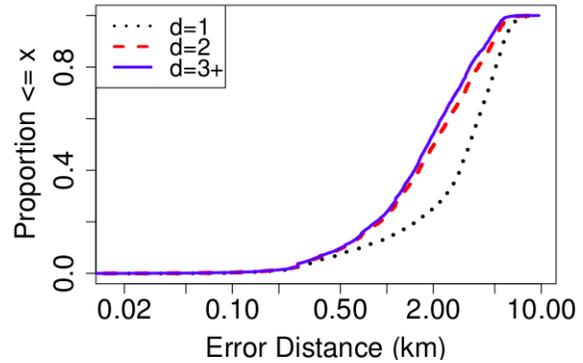
**Table 1. Mean Estimation Error (km)**

We evaluate the distribution of error estimates as a function of  $d$  in in Figure 2. At  $d = 1$ , where only directly adjacent neighbors are considered, the estimation errors are bi-modal with small peaks at approximately 0.7km and 3.8km. The error distribution has a very wide variance; except for a decrease between 1 and 3km, the error terms are generally distributed uniformly in the range 0 and 6km. The bi-modal behavior disappears for  $d = 2$  and most of the mass accumulates at errors less than than 2km. For  $d = 3$ , the peak sharpens even further at approximately 1.75km, which is less than the mean estimation error of 2.18km. The densities for  $d \geq 3$  are nearly identical because the estimates change only for a very small number of tweets as  $d$  increases from 1 to 3.



**Figure 2. Error Densities vs.  $d$**

Finally, we evaluate how frequently our model is accurate to within a given distance in Figure 3, where we plot the CDF of estimation errors for  $d = 1, 2$ , and  $\geq 3$  on semi-log scale. The shape of the distribution function becomes linear for  $d > 2$ , which suggests that the density function takes an exponential form as the smoothing diameter increases. We also find that the estimation accuracy increases only marginally beyond  $d \geq 2$ , suggesting that smoothing over neighbors more than 2 sub-regions away offers diminishing returns. Furthermore, the semi-log plots confirm monotonic behavior. In other words, there is no special case or specific instance for which a smaller value of  $d$  outperforms a larger value  $d$ . The overall accuracy of our approach is promising; at  $d = 3$  the model can estimate the geo-coordinates of a tweet to within 4km with probability 80%, to within 2km over 50%, and to within 1km at 20%.



**Figure 3. Error Distributions**

## 4 Related Research

In this section, we review contemporary techniques according to whether they estimate the locations or geo-coordinates of social medial posts.

Broadly, the techniques that identify user locations either zero in on the “home” locations of users or location “types” from where users check-in and update. These methods rely on spatial word usage and language models [6], posting behaviors and external data on locations [14], and inferences based on unified discriminative models [13]. Estimation of specific geo-coordinates also use varied techniques including language models [18, 4], spatial word distributions [4], sequences of check-ins from social network friends [16], integrating mobile phone data [5], and mapping latent topics from across regions [10].

Despite the integration of data from separate sources, and the aid of sophisticated probabilistic models, both types of approaches can identify only broad areas. By contrast, the methodology proposed in this paper can narrowly estimate geo-coordinates while relying solely on the content of the posts and on prior knowledge about the wide area from where they originated.

## 5 Conclusions & Future Work

In this paper, we presented a methodology to narrowly estimate the geo-coordinates of a social media post, given the knowledge of the much broader region from where they originate. An experimental evaluation using tweets collected from New York City shows that on an average the methodology can estimate geo-coordinates of social media posts to within 2.15km, or just 4% of the size of the broader region. Future work will examine the accuracy of the approach over regions with distinct geographic features, sizes, and population distributions. We also propose to investigate the accuracy of alternative geo-smoothing methods.

## References

- [1] L. R. Bahl, F. Jelinek, and R. L. Mercer. A maximum likelihood approach to continuous speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (2):179–190, 1983.
- [2] S. Chandra, L. Khan, and F. Muhaya. Estimating twitter user location using social interactions—a content based approach. In *Intl. Conf. on Social Computing*, pages 838–843. IEEE, 2011.
- [3] S. Chen and J. Goodman. An empirical study of smoothing techniques for language modeling. In *Proc. of Association for Computational Linguistics Annual Meeting*, pages 310–318. Association for Computational Linguistics, 1996.
- [4] Z. Cheng, J. Caverlee, and K. Lee. You are where you tweet: a content-based approach to geo-locating twitter users. In *Proc. of Intl. Conference on Information and knowledge management*, pages 759–768. ACM, 2010.
- [5] E. Cho, S. A. Myers, and J. Leskovec. Friendship and mobility: user movement in location-based social networks. In *Proc. of Intl. conference on Knowledge discovery and data mining*, pages 1082–1090. ACM, 2011.
- [6] N. Dalvi, R. Kumar, and B. Pang. Object matching in tweets with spatial models. In *Proc. of Intl. Conference on Web search and data mining*, pages 43–52. ACM, 2012.
- [7] D. Doran, S. Gokhale, and A. Dagnino. Human Sensing for Smart Cities. In *Proc. of Intl. Conf. on Advances in Social Network Analysis and Mining*, pages 1323–1330, 2013.
- [8] D. Doran, S. Gokhale, and A. Dagnino. Understanding Common Perceptions from Online Social Media. In *Proc. of Intl. Conference on Software Engineering and Knowledge Engineering*, pages 107–112, 2013.
- [9] B. Hecht, L. Hong, B. Suh, and H. E. Chi. Tweets from justin bieber’s heart: the dynamics of the location field in user profiles. In *Proc. of Intl. Conference on Human Factors in Computing Systems*, pages 237–246. ACM, 2011.
- [10] L. Hong, A. Ahmed, S. Gurumurthy, A. J. Smola, and K. Tsioutsoulouklis. Discovering Geographical Topics in the Twitter Stream. In *Proc. of Intl. Conf. on World Wide Web*, pages 769–778, 2012.
- [11] R. Kneser and H. Ney. Improved backing-off for m-gram language modeling. In *Intl. Conference on Acoustics, Speech, and Signal Processing*, volume 1, pages 181–184. IEEE, 1995.
- [12] B. Kolmel and S. Alexakis. Location based advertising. In *Intl. Conference on Mobile Business*, 2002.
- [13] R. Li, S. Wang, H. Deng, R. Wang, and K. C.-C. Chang. Towards social user profiling: unified and discriminative influence model for inferring home locations. In *Proc. of Intl. Conference on Knowledge discovery and data mining*, pages 1023–1031. ACM, 2012.
- [14] J. Mahmud, J. Nichols, and C. Drews. Where is this tweet from? inferring home locations of twitter users. In *Proc. of Intl. Conference on Weblogs and Social Media*, volume 12, pages 511–514, 2012.
- [15] D. Mok, B. Wellman, and J. Carrasco. Does distance matter in the age of the internet? *Urban Studies*, 47(13):2747–2783, 2010.
- [16] A. Sadilek, H. Kautz, and J. Bigham. Finding Your Friends and Following Them to Where You Are. In *Proc. of Intl. Conference on Web Search and Data Mining*, pages 723–732, 2012.
- [17] S. Scellato, A. Noulas, R. Lambiotte, and C. Mascolo. Socio-spatial properties of online location-based social networks. In *Proc. of Intl. Conference on Weblogs and Social Media*, volume 11, pages 329–336, 2011.
- [18] P. Serdyukov, V. Murdock, and R. Van Zwol. Placing flickr photos on a map. In *Proc. of Intl. Conf. on Research and Development in Information Retrieval*, pages 484–491, 2009.
- [19] M. Sundermeyer, R. Schlüter, and H. Ney. On the estimation of discount parameters for language model smoothing. *Interspeech*, 2011.
- [20] J. Yin, A. Lampert, M. Cameron, B. Robinson, and R. Power. Using Social Media to Enhance Emergency Situation Awareness. *IEEE Intelligent Systems*, 27(6):52–59, 2012.

# Two-Level Smart Search Engine Using Ontology-Based Semantic Reasoning

Haiping Xu and Arturo W. Li

Computer and Information Science Department  
University of Massachusetts Dartmouth, North Dartmouth, MA 02747, USA  
{hxu, ali}@umassd.edu

**Abstract**—Traditional search engines are typically keyword-based, which cannot understand the semantics of a query or relationships among queried concepts, causing much of the related information to be absent from the search results. As opposed to traditional keyword-based search engines, in this paper, we introduce an approach to developing a smart search engine using ontology-based semantic reasoning. The search engine can understand the semantics of a concept or multiple concepts entered by a user, and classify and reason about the concepts within a knowledge domain specified using ontology. The search engine consists of two levels, namely the semantic reasoning level and the traditional keyword searching level. The semantic reasoning level supports reasoning about subsumption, supersumption, semantic equivalency and property relationships among concepts. Once new concepts related to the queried concepts have been inferred from the ontology, they can be further matched using a traditional keyword-based search mechanism. To demonstrate the feasibility of our approach, we adopt the domain of computer science courses for semantic search, and show that our approach may produce more relevant search results to user queries.

*Keywords*—Search engine; smart search; semantic reasoning; ontology; domain knowledge.

## I. INTRODUCTION

The growth of Internet has made search engines one of the most frequently used web applications over the past decades. Search engines can be used to search information on the web, including documents, images, audios, videos, and related web pages. Traditional search engines such as Google search engine, typically use the keyword-matching approach and ranking algorithms to retrieve the desired information for user queries. They have been quite successful by providing users simple search interface and useful search results. However, since traditional search engines cannot understand the semantics of a query or relations among queried concepts entered by a user, they often offer low recall and precision, with much of the related information being absent from the search results or too many irrelevant and ambiguous results being presented [1],[2]. For example, the word “Football” may refer to the sport played in North America with an oval ball or the Olympic sport played with a spherical shape ball known as soccer in America. When a user types the query “Football Leagues” into a traditional search engine, expecting some search results related to the Olympic sport, the highly ranked search results are unfortunately all about American football

leagues with no relation to the Olympic sport. In order to return desirable results (e.g., the European Football League) for this user, it is required that the search engine can clearly understand the ambiguous meaning of the word “Football.” In another example, suppose a computer science student wants to search for information related to a concept called “pushdown automaton.” Unfortunately, the student cannot remember this terminology, though he knows that such a model contains some states and a stack. Using semantic search, when the student types a pair of concepts “State” and “Stack” within the domain of computer science courses, the reasoner should efficiently infer the concept of “pushdown automaton” and present it to the student for further querying. Different from traditional search engines, semantic search engines utilize semantic information of certain domain knowledge specified using ontology. With the support of domain knowledge, user queries are precisely and unambiguously analyzed in order to provide better precision and recall rates for search results. Thus, ontology-based semantic search can highly improve search accuracy of the query and deliver results that are more relevant to the user queries.

In this paper, we propose an ontology-based methodology combined with traditional keyword-matching approaches to obtain more accurate and relevant search results for user queries. The proposed smart search engine model consists of two levels, namely the semantic reasoning level and the traditional keyword searching level. As one of the advantages of our approach, users are not required to be familiar with the exact concepts to be searched when formulating queries. Instead, the smart search engine can automatically infer the desired concepts by analyzing and reasoning about user-provided concepts within the given knowledge domain. To demonstrate the advantages of our approach, we develop a prototype smart search engine that supports reasoning about subsumption, supersumption, semantic equivalency and object property relations among concepts, and adopt the domain knowledge in computer science courses to demonstrate how semantic search may benefit computer science students.

Although there have been many efforts on semantic search, research on this topic is still in its premature stage. SHOE search engine [3], introduced by Heflin and Hendler, has a standalone architecture and is one of the first form-based semantic search engines. It provides sophisticated web forms that allow users to specify queries. Similar to SHOE, the OntoIR system [4] proposed by Garcia and Sicilia also belongs to the form-based semantic search engine category. It improves the SHOE approach by providing a selection-based interface

for the users. Although the above form-based approaches can be useful for those who are familiar with the domain ontology, they are not quite usable for typical Internet users. In contrast, our approach is not form based and does not require users to be familiar with the domain knowledge; thus, our approach allows users to interact with the system in a similar way to traditional search engines. There are also a few semantic search engines with standalone architecture and a RDF-based querying language. Swoogle is a crawler-based semantic web search engine that discovers and indexes documents containing RDF data [5]. When a new semantic web document is discovered, Swoogle analyzes it and extracts the needed data, and then it computes the metadata and derives the statistical properties. Similar to Google, Swoogle uses two ranking algorithms, namely OntoRank and Term Rank, to rank search results. Likewise, SWSE (Semantic Web Search Engine) consists of components that support web crawling, data enhancing and indexing, search interface, and browsing and retrieval of information [6]. It operates over RDF web data, and adapts large-scale web search engines to the case of structured data. Although the semantic search engines mentioned above can operate on RDF data, they do not support reasoning about relations among concepts in order to infer new knowledge. In contrast, our approach uses the Web Ontology Language (OWL) to specify domain ontologies, which is a knowledge representation language for authoring ontologies or knowledge bases. Thus, our approach supports reasoning about user queries, and can classify and infer related concepts to enhance search performance.

## II. ONTOLOGY DEVELOPMENT

Ontology can be used to formally and explicitly specify a conceptualization, and precisely describe the concepts and relationships that exist in a particular domain of knowledge. Ontology allows computers to understand and infer about meaning of information by providing an organized framework for classification and reasoning of given concepts and relationships. In the following sections, we describe important relations between concepts, and show how to use object property restrictions to define new concepts.

### A. Concept Definition and Class Hierarchy

Ontology is typically organized as a hierarchical structure, which contains a set of concepts and relationships that can be inherited to their child elements. The combination of all concepts and their relationships constitute the knowledge base of the domain. Concepts can be represented in the form of classes organized within a class hierarchy, where classes are connected with each other by class-level relationships, namely, subsumption relation (i.e., *is-subsumed-by* or *is-subclass-of*, denoted as  $\sqsubseteq$ ), supersumption relation (i.e., *is-superclass-of*, denoted as  $\sqsupseteq$ ), and semantic equivalence (i.e., *is-equivalent-to*, denoted as  $\equiv$ ) [7]. We follow a top-down approach to develop the class hierarchy for the domain knowledge. The top-down development approach starts with the definition of the most general concepts in the domain and subsequent specialization of the concepts. In an ontology-based class hierarchy, class-level relations between concepts are explicitly defined using the keywords `subClassOf` and `equivalentClass`. For example, the class `FiniteAutomaton` can be defined using

ontology language OWL as a subclass of the `Automaton` class. Since it has the same semantic as a finite state machine, it is also defined as an equivalent class of the `FiniteStateMachine` class. Based on the above definitions, a reasoner can infer that all properties of the `Automaton` class and the `FiniteStateMachine` class are also properties of the `FiniteAutomaton` class due to subsumption and semantic equivalence relations among the classes, respectively.

### B. Relation Between Concepts

Concepts defined as classes in a hierarchical order are not sufficient for describing certain domain knowledge. Properties between instances of classes may also be needed to describe further relations between different concepts. Relations between concepts can be specified using object properties in OWL, which are usually defined as actions. For example, the two concepts `Automaton` and `Language` can be defined as the domain and the range of the object property relation `recognizes`, respectively. Since all subclasses inherit the properties of their superclasses, when defining the domain and the range of an object property relation, we usually choose the most general classes that satisfy the relation from the class hierarchy. In addition, we also define the object property `recognizes` as an inverse relation of the object property `isRecognizedBy` since a language can be recognized by an automaton. Furthermore, the object property `recognizes` can be defined as a functional property because each automaton recognizes only one language, i.e., the language of an automaton is unique.

### C. Class Definition Using Property Restriction

When we declare that class  $a$  satisfies condition  $\varphi$ , it is equivalent to say that  $a$  is a subclass of  $a'$  that satisfies a necessary and sufficient condition  $\varphi$ . This is because all instances of  $a$  must also be instances of  $a'$  since they all satisfy the necessary and sufficient condition of class  $a'$ , namely, condition  $\varphi$ . Similarly, when concept  $c$  has property  $p(c, c1)$ , where  $c$  has an object property relation  $p$  with concept  $c1$ , concept  $c$  can be defined as a subclass of concept  $c'$  with property  $p(c', c1)$  as its necessary and sufficient condition. In OWL, concept  $c'$  is called a *property restriction*, which is usually kept as an anonymous class. For example, when we define `PDA` that contains some `Stack`, we could specify `PDA` as a subclass of property restriction  $r_1$  that satisfies property `contains( $r_1$ , Stack)`. Similarly, we can specify `PDA` as a subclass of a property restriction  $r_2$  that satisfies property `isCoveredBy( $r_2$ , CIS361)` since the concept `PDA` is covered by undergraduate course `CIS361`.

## III. ONTOLOGY-BASED TWO-LEVEL SMART SEARCH ENGINE

### A. Architectural Design

The framework for the two-level smart search engine is illustrated in Fig. 1. The system consists of a user interface module, a query handler, the domain ontology and two major levels, called the semantic reasoning module (Level 1) and the keyword-based query matching module (Level 2), respectively. In this paper, we focus on the first level, i.e., the semantic reasoning, and adopt an existing keyword-matching traditional

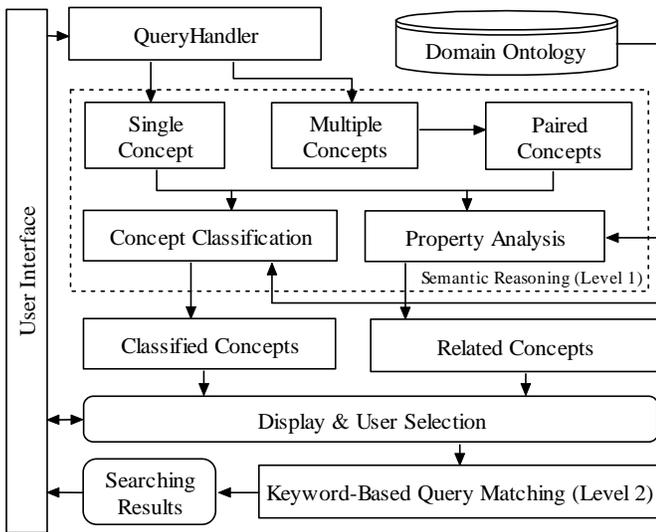


Figure 1. A framework for the two-level smart search engine

search engine, such as Google, Yahoo, Bing or DogPile, as the searching mechanism at the second level.

As shown in Fig. 1, the user interface allows a user to type in a single or multiple concepts in the form of string(s). The user inputs are filtered and parsed by the query handler, and matched with concepts defined in the domain ontology. When the user inputs are matched with a single concept, it is processed by the semantic reasoning module directly. On the other hand, when the user inputs are matched with two or more than two concepts, paired concepts are generated, and each pair of concepts is processed by the semantic reasoning module individually before the reasoning results are combined. For either a single concept or paired concepts, the semantic reasoning module first classifies the concept within the class hierarchy of the domain ontology, and infers the concepts that have the subsumption, supersumption, and semantic equivalence relations with the input concept(s). We called such inferred concepts *classified concepts*. Then it analyzes the properties of the input concept(s), and infers any concepts that are related to the input concept(s) by object properties. We call such inferred concepts *related concepts*. Once we derive all classified concepts and related concepts, they are displayed on screen to allow a user to select the most desired concept for further concept matching. In the second level of the smart search engine, a user-selected concept can be again sent to the semantic reasoning module (this option is not shown in Fig. 1) or it can be sent to a keyword-based query matching module, e.g., the Google search engine, to search for relevant information to the user query from the Internet.

### B. Semantic Reasoning for Single-Concept Query

When a query contains a single concept, we call it a *simple query*. For example, if a user types in “Automaton,” we consider it a simple query since “Automaton” represents a single concept. For a single concept  $sc$ , an inferred concept can be either a classified one or a related one. A classified concept subsumes, supersumes or is semantically equivalent to  $sc$ ; while a related concept could be inferred due to a *someValuesFrom* or *allValuesFrom* restriction. In the

following defined enumerated single concept relation, the aforementioned five different types of relations are denoted as SUB, SUP, EQ, SOME or ALL, respectively,

```
enum SingleConceptRelation { // for a single concept
    SUB, SUP, EQ, SOME, ALL }
```

The SimpleRelation and InferredConcept classes are defined in the following. Note that we also define an abstract class Relation, which serves as a superclass of the SimpleRelation class as well as more complicated relations described in Section III.C.

```
class SimpleRelation extends Relation {
    SingleConceptRelation sRel;
    OntClass queriedConcept;
    Relation getRelation() { ... }
}
```

```
abstract class Relation {
    abstract Relation getRelation();
}
```

```
class InferredConcept {
    OntClass concept; // the ontology class
    Relation relation; // relationship with the
                      // concept(s) to be queried
}
```

The algorithm for inferring new concepts for a single concept is presented as Algorithm 1. When a user inputs a single concept, it is first processed (e.g., removing the space in the input string “Finite Automaton”), and matched with a predefined concept (i.e., *FiniteAutomaton*) within the domain ontology. Then the system uses a semantic reasoner to infer all classified concepts that subsume, supersume, and are semantically equivalent to the input concept. In the following steps, the algorithm checks all restrictions that are superclasses of the input concept, and infers all related concepts that have object property relations with the input concept by *someValuesFrom* or *allValuesFrom* restriction. Once all classified and related concepts are identified, they are returned as a list of inferred concepts for further processing.

---

#### Algorithm 1: Inference of New Concepts (Single Concept)

---

**Input:** User input representing a single concept  $sc$

**Output:** A list of inferred concepts  $lic$  related to  $sc$

---

1. match user inputs with single concept  $sc$  in the domain ontology
  2. initialize  $lic$  to an empty list
  3. infer all concepts such that each concept  $c \sqsubset sc$ ,  $c \sqsupset sc$  or  $c \equiv sc$ , and store them in sets  $sSub$ ,  $sSup$ , and  $sEq$ , respectively.
  4. **for each**  $c$  in  $sSub$
  5.   set  $c.relation.sRel$  to SUB, and add  $c$  into  $lic$
  6. **for each**  $c$  in  $sSup$
  7.   set  $c.relation.sRel$  to SUP, and add  $c$  into  $lic$
  8. **for each**  $c$  in  $sEq$
  9.   set  $c.relation.sRel$  to EQ, and add  $c$  into  $lic$
  10. let  $lres$  be the list of restrictions from the set of superclasses of  $sc$
  11. let  $inferc$  be an instance of *InferredConcept*
  12. **for each** restriction  $res$  in  $lres$
  13.   **if**  $res$  is a *someValuesFrom*  $inferc.concept$  restriction
  14.     set  $inferc.relation.sRel$  to SOME, and add  $inferc$  to  $lic$
  15.   **else if**  $res$  is an *allValuesFrom*  $inferc.concept$  restriction
  16.     set  $inferc.relation.sRel$  to ALL, and add  $inferc$  to  $lic$
  17. **return** list  $lic$
-

### C. Semantic Reasoning for Multi-Concept Query

When a user types in a query that involves two or more concepts, we call it a *complex query*. In this case, the semantic reasoning module first generates all possible paired concepts for efficient processing, and for each pair of concepts ( $c1$ ,  $c2$ ), it checks the following special cases that require only searching for a single concept.

**Special Case 1 (SP1).** When concept  $c1$  is equivalent to concept  $c2$  (i.e.,  $c1 \equiv c2$ ), we only need to search for single concept  $c1$  or  $c2$ , and all classified concepts and related concepts for both  $c1$  and  $c2$  will be inferred.

**Special Case 2 (SP2).** When concept  $c1$  and  $c2$  have a subsumption relation (i.e.,  $c1 \sqsubseteq c2$ ), we only need to search for single concept  $c1$  since  $c1$  inherits all properties of  $c2$ . In this case, all classified and related concepts of  $c2$  will be automatically inferred for concept  $c1$ . Similarly, when  $c2 \sqsubseteq c1$ , we only need to search for single concept  $c2$ .

**Special Case 3 (SP3).** When there exists an object property relation  $p(c1, c2)$  between paired concepts  $c1$  and  $c2$ , we only need to search for single concept  $c1$  since in this case,  $c2$  will be automatically inferred as a related concept of  $c1$ .

On the other hand, if the relation between  $c1$  and  $c2$  does not belong to any of the above special cases, new concepts must be inferred using both of the two concepts. Four typical relations between an inferred concept  $c$  and the pair of concepts ( $c1$ ,  $c2$ ) are defined as follows.

**Common Supersumption (CSUP).** Concept  $c$  is an inferred classified concept such that  $c \sqsupset c1$  and  $c \sqsupset c2$ , i.e.,  $c$  is a proper superclass of both  $c1$  and  $c2$ . Note that in this case,  $c \neq c1$  and  $c \neq c2$ ; otherwise, the relation must belong to either *SP1* or *SP2*. For example, if  $c \equiv c1$  and  $c \sqsupseteq c2$ , we have  $c1 \sqsubseteq c2$ , i.e.,  $c2 \sqsubseteq c1$ , which is defined as *SP2*.

**Common Subsumption (CSUB).** Concept  $c$  is an inferred classified concept such that  $c \sqsubset c1$  and  $c \sqsubset c2$ , i.e.,  $c$  is a proper subclass of both  $c1$  and  $c2$ . Note that in this case,  $c \neq c1$  and  $c \neq c2$ ; otherwise, the relation must belong to either *SP1* or *SP2*. For example, if  $c \equiv c1$  and  $c \sqsubseteq c2$ , we have  $c1 \sqsubseteq c2$ , which is defined as *SP2*.

**Property Relation (PROP).** Concept  $c$  is an inferred related concept, and  $\exists p_1(c, c1)$  and  $p_2(c, c2)$ , where  $p_1$  and  $p_2$  are object property relations between  $c$  and  $c1$ , and  $c$  and  $c2$ , respectively. Note that  $p_1$  and  $p_2$  can be either the same or different object property relations.

**Property-Subsumption Relation (PROP-SUB).** Concept  $c$  is an inferred related concept such that  $c \sqsubset c2$  and  $\exists p_1(c, c1)$  or  $p_1(c1, c)$ , where  $p_1$  is an object property relation between  $c$  and  $c1$ ; or  $c \sqsubset c1$  and  $\exists p_2(c, c2)$  or  $p_2(c2, c)$ , where  $p_2$  is an object property relation between  $c$  and  $c2$ .

Note that we do not need to consider the case of *Property-Supersumption Relation*, where an inferred related concept  $c$  is a superclass of  $c1$  (or  $c2$ ), and there exists an object property relation between  $c$  and  $c2$  (or between  $c$  and  $c1$ ). This is because in such cases, the relation must belong to the special

case *SP3*. For example, in a paired concept ( $c1$ ,  $c2$ ), if concept  $c$  is a superclass of concept  $c2$ , and there exists an object property relation  $p_1(c, c1)$ , there must exist an object property relation  $p_2(c2, c1)$  since  $c2$  inherits all properties of  $c$ . This is exactly the case defined in special case *SP3*; therefore, we only need to search for single concept  $c2$ .

The algorithm for inferring new concepts from multiple concepts can be designed in a similar way by considering the special cases and the CSUP, CSUB, PROP, and PROP-SUB relations (due to page limitation, the algorithm is not presented in this paper). Briefly, the algorithm first generates a list of paired concepts, and then infers classified and related concepts for each pair of concepts ( $c1$ ,  $c2$ ). Note that in the special cases when  $c1$  is equivalent to  $c2$ ,  $c1(c2)$  subsumes  $c2(c1)$ , or  $c1$  and  $c2$  has an object property relation, the algorithm automatically executes Algorithm 1 that infers classified and related classes for a single concept. Otherwise, the algorithm first infers classified concepts, namely, the common superclasses *csup* that subsume both  $c1$  and  $c2$ , and the common subclasses *csub* that supersume both  $c1$  and  $c2$ . Then the algorithm checks whether there exists any concept  $c$  that has object property relations with both  $c1$  and  $c2$ . Finally, it infers any possible concept  $c$  that has a property-subsumption relation with the pair of concepts ( $c1$ ,  $c2$ ). All inferred concepts will be added into a list of inferred concepts, and returned as output of the algorithm.

## IV. CASE STUDY

To demonstrate the feasibility of our proposed approach to developing the two-level smart search engine using ontology-based semantic reasoning, we present a case study of semantic search in the domain of computer science courses. In particular, we have defined the domain ontology based on two computer science courses taught by the first author for over 10 years at the University of Massachusetts Dartmouth (UMassD). The two courses are the undergraduate course CIS 361: Models of Computation and the graduate course CIS 560: Theoretical Computer Science. We expect a feasible smart search engine in such a knowledge domain can greatly benefit our computer science students in understanding fundamental concepts and their relations in computer science.

The smart search engine was developed using the Pellet reasoner as a Java-based OWL-DL reasoner, which provides reasoning services for OWL ontologies. The Pellet OWL2 Reasoner for Java works with the Jena framework that provides a programmatic environment for RDF, OWL, SPARQL, and includes a rule-based inference engine. The smart search engine processes a user input as concept(s) rather than keywords, and infers classified and related concepts. A user may select an inferred concept for further semantic reasoning or send it to a traditional search engine for retrieving related information from the Internet.

Once we have defined the domain ontologies for the course contents in CIS 361 and CIS 560, we can use our prototype two-level smart search engine for semantic search. Suppose a computer science student wants to search for some concepts that are related to a pair of concepts (*Stack*, *State*), some partial semantic search results for the paired concepts will be displayed as in Fig. 2.

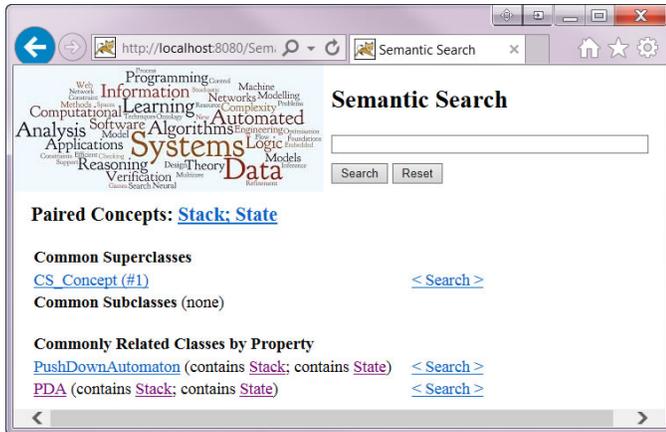


Figure 2. Search results for a paired concepts “Stack; State”

From the search results, we can see that both `Stack` and `State` are subclasses of `CS_Concept`. The commonly related concepts inferred according to the *Property Relation* defined in Section III.C include `PushDownAutomaton` and `PDA` because both contain some `State` and some `Stack`. By clicking on the links of `PushDownAutomaton` and `PDA`, they can be further processed as a single concept. Alternatively, when an associated “Search” link is clicked, the corresponding single concept (e.g., `PushDownAutomaton`) can be searched as a keyword using a traditional search engine.

In another scenario, when a computer science student wants to search for the type of language that can be recognized by a PDA, he/she can simply type in a pair of concepts (`Language`, `PDA`) into the smart search engine, and some partial semantic search results will be displayed as in Fig. 3.

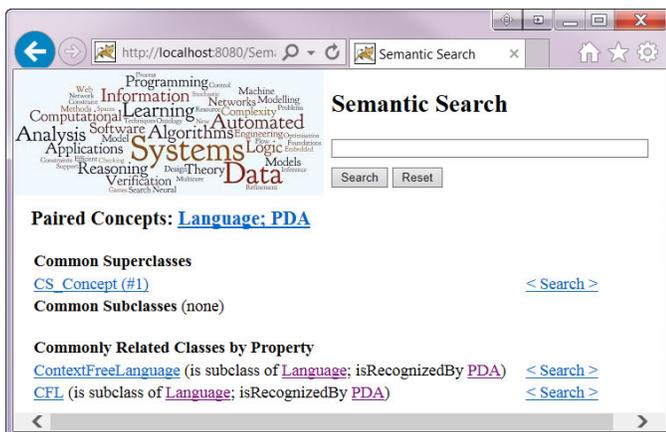


Figure 3. Search results for a paired concepts “Language; PDA”

From the search results, we know that both the concepts `ContextFreeLanguage` and `CFL` are commonly related to the paired concepts (`Language`, `PDA`). The inference of these two new concepts is based on the *Property-Subsumption Relation* defined in Section III.C since both of them are subclasses of `Language` and can be recognized by `PDA`. By clicking on the link of `ContextFreeLanguage` or `CFL`, the inferred concept can be further searched as a single concept. Alternatively, when an associated “Search” link is clicked, the

corresponding concept (e.g., `ContextFreeLanguage`) can be searched as a keyword using a traditional search engine.

## V. CONCLUSIONS AND FUTURE WORK

In this paper, we present an ontology-based methodology for semantic search that can produce more accurate and relevant search results for a concept-based query. The smart search engine consists of two levels, namely the semantic reasoning level and the traditional keyword searching level. The semantic reasoning level can understand the meaning of a concept or multiple concepts entered by a user, and classify and reason about the concepts in the corresponding knowledge domain specified using ontology. Once the semantic reasoning module infers the concepts related to a user query, in the second level, a traditional search engine can be used to search for additional information from the Internet. One major benefit of our approach is that a user does not have to be familiar with the domain knowledge when making queries. Instead, the smart search engine can reason about user queries, and produce those concepts that are closely related to the user inputs. To demonstrate the feasibility of our smart search engine approach, we develop a prototype smart search engine as well as ontologies in the domain of computer science courses. The case studies show that the smart search engine can effectively infer concepts that are semantically related to user queries.

As future work, we plan to develop a set of more complete domain knowledge for the computer science courses taught at UMassD, and show that such smart search engine can greatly help students to understand difficult computer science concepts. In addition to dealing with paired concepts, we will also design algorithms that can directly process a complex query with more than two concepts. Such algorithms should efficiently infer new related concepts from the given multiple concepts by reasoning about their relations with each other.

## REFERENCES

- [1] Y. Lei, V. Uren, and E. Motta, “SemSearch: A Search Engine for the Semantic Web,” *Proceedings of the 15th International Conference on Knowledge Engineering and Knowledge Management (EKAW 2006)*, Lecture Notes in Computer Science, Vol. 4248, Springer, Heidelberg, Oct. 2-6, 2006, Pödebrady, Czech Republic, pp. 238-245.
- [2] C. Mangold, “A Survey and Classification of Semantic Search Approaches,” *International Journal of Metadata, Semantics and Ontologies (IJMSO)*, Vol. 2, No. 1, 2007, pp. 23-34.
- [3] J. Heflin and J. Hendler, “Searching the Web with SHOE,” *Proceedings of the AAAI Workshop on Artificial Intelligence for Web Search*, AAAI Press, Menlo Park, CA, 2000, pp. 35-40.
- [4] E. García and M. Sicilia, “Designing Ontology-Based Interactive Information Retrieval Interfaces,” *Proceedings of the Workshop on Human Computer Interface for Semantic Web and Web applications (HCI-SWWA)*, Lecture Notes in Computer Science 2003, Springer, New York, 2003, pp. 152-165.
- [5] L. Ding, T. Finin, A. Joshi, R. Pan, R. S. Cost, Y. Peng, P. Reddivari, V. Doshi, and J. Sachs, “Swoogle: a Search and Metadata Engine for the Semantic Web,” *Proceedings of the Thirteenth ACM Conference on Information and Knowledge Management (CIKM'04)*, Washington DC, USA, 2004, pp.652-659.
- [6] A. Hogan, A. Harth, J. Umbrich, et. al., “Searching and Browsing Linked Data with SWSE: the Semantic Web Search Engine,” *Journal of Web Semantics*, Vol. 9, No. 4, 2011, pp. 365-401.
- [7] G. Antoniou and F. van Harmelen, *A Semantic Web Primer*, 2<sup>nd</sup> Edition, MIT Press, Massachusetts, 2008.

# User Profile Visualization to facilitate MSLIM-model-based Social Influence Analysis based on Slow Intelligence Approach

Yingze Wang and Shi-Kuo Chang

Department of Computer Science, University of Pittsburgh, Pittsburgh, PA 15260, USA  
{yingzewang,chang}@cs.pitt.edu

**Abstract**—Identifying the hub nodes (the most influential users) in an information diffusion network is a central research topic in social influence analysis. In [2], we developed the Multi-task sparse linear influence model (MSLIM) to automatically select topic-sensitive influential nodes in an implicit network. Our method firstly needs to determine the active node set to model the global volume of the contagions of the entire network nodes. The active node set is chosen by a simple rule in [2]. However different set leads to different model performance. [4] proposed a new framework called Slow Intelligence System, which can continuously learn, search the appropriate method and propagate information according to the environment to improve the performance over time. In this paper, we develop a slow intelligence based active user definition system by utilizing user profile visualization to help interactively determine active node sets in MSLIM model. We apply our system on a set of 2.6 million tweets of 1000 users on twitter. We show that our system can efficiently utilize user’s profile visualization techniques to facilitate determining the appropriate active node sets.

*Keywords*—Slow intelligence system, Data mining, Social influence analysis, Visualization

## I. INTRODUCTION

Modeling the diffusion of information has been one of core research areas for analyzing social network. One important question in modeling information diffusion is how to find influential nodes in the network. Most algorithms [7], [8], [9], [10], [11] rely on a strong assumption that the entire structure of the network is known as prior knowledge and information can only spread over the edges of the underlying diffusion network. However, in many scenarios, the diffusion network is implicit or unknown. Recently, *linear influence model* (LIM) [3] was proposed to analyze the information diffusion with an unknown network structure. LIM models the global influence of each node through an entire implicit network. However it cannot identify the influential users.

We firstly extended the LIM by utilizing the sparse learning framework to develop Sparse Linear Influence Model (SLIM) [1] so that we can identify the most global influential nodes in an implicit information diffusion network. However, both LIM and SLIM use a global influence function for each node, which is based on an underlying assumption that each node has the same influence across all the contagions. Clearly, this assumption could be too restrictive for many applications. For different types of contagions, the set of the most influential nodes could be very different. Some users might be active on some topics but are less influential on the other topics. To achieve

contagion-sensitive node selection in an implicit network, we extended our SLIM model to the multitask sparse learning setting and proposed the Multitask Sparse Linear Influential Model (MSLIM), which can automatically select the hub nodes for different contagions in an implicit network.

In MSLIM model, we firstly need to select a number of users out of total users to construct the so-called active node set for modeling the total volume of contagions. In the experiments, we selected  $N$  twitter users with at least 1,000 tweets during the certain time period. However this active user set may not be optimized. Selection of active user sets is important for our model, since different set can lead to different performance. In this paper, we utilize the slow intelligence system with two visualization techniques (e.g. histogram and choropleth map) to help us define the active node set, which further facilitates the MSLIM model to reduce prediction error and improve contagion-sensitive influential user selection.

Recently, Chang [4] proposed a general intelligent framework called slow intelligence system (SIS). Chang et. al also presented the design of component based SIS [5] and the user interface design to produce and manage the SIS system [6]. In this paper, we propose a SIS-based active node set definition system, which uses visualization techniques for different user profile properties. Our system can choose active user sets according to different properties, e.g. locations, friends counts, etc. It searches the best active user set based on the model prediction error and propagates the learned knowledge over iterations. Furthermore, it can combine different methods into one hybrid system to better define the most suitable active node sets. The proposed system can be applied to many various sources of social media. In this paper, we specifically apply our system on the twitter dataset. The experiments show that our system can efficiently utilize user’s profile visualization techniques to facilitate selecting the proper active node sets. The rest of this paper is organized as follows. We present the background of MSLIM model and Slow Intelligence System in Section II. In Section III, we introduce our SIS-based active node set definition system. In Section IV, we present the experimental results on real twitter dataset. Section V is the conclusion and future work.

## II. BACKGROUND

### A. Multitask Sparse Linear Influence Model (MSLIM)

Following the notations in LIM model [3], we assume there are  $N$  active nodes (corresponds to  $N$  active users) and  $K$  contagions diffused over the network (corresponds

to  $K$  topics). Assuming the entire time intervals are normalized into  $T$  units:  $\{0, 1, 2, \dots, T\}$ .  $M_{u,k}(t)$  is the indicator function to present whether the node  $u$  got infected by the contagion  $k$  between the time interval  $t - 1$  and  $t$ . Furthermore, let  $V_k(t)$  denote the total volume of contagion  $k$  between  $t - 1$  and  $t$ . Multi-task linear influence model can be formulated as:

$$V_k(t+1) = \sum_{u=1}^N \sum_{l=0}^{L-1} M_{u,k}(t-l) I_{uk}(l+1) + \epsilon_k(t+1) \quad (1)$$

Where  $L$  is the maximum lag-length and  $I_{uk}(l)$  is the non-negative influence factor of user  $u$  on contagion  $k$  at the time-lag  $l$  and  $\epsilon_k(t)$  is the i.i.d. zero-mean Gaussian noise. Following [3],  $V_k(t)$  and  $M_{u,k}(t)$  can be organized into the matrix form  $\mathbf{V}_k$  and  $\mathbf{M}_k$ . We further define the contagion-sensitive node influence function  $\mathbf{I}_{u,k} \in R^{L \times 1}$  which is a  $L$ -length vector representing the influence of the node  $u$  on the contagion  $k$ . For each contagion  $k$ , let  $\mathbf{I}^k \in R^{L \cdot N \times 1}$  be the vector obtained by concatenating  $\mathbf{I}_{1k}, \dots, \mathbf{I}_{Nk}$ . For each node  $u$ , we further define the influence matrix for the node  $u$ :  $\mathbf{I}_u = (\mathbf{I}_{u1}, \dots, \mathbf{I}_{uK}) \in R^{L \times K}$ . Given the contagion-sensitive influence function, we assume a linear relationship between  $\mathbf{V}_k$  and  $\mathbf{M}_k$ :

$$\mathbf{V}_k = \mathbf{M}_k \mathbf{I}^k + \epsilon \quad (2)$$

We formulate the problem of contagion-sensitive influential node selection into a convex optimization problem, which jointly minimizes the square loss and a non-smooth sparsity-inducing penalty. In particular, we estimate the non-negative vector  $\{\mathbf{I}_{u,k}\}$  for all active nodes and contagions by multitask sparse linear influence model (MSLIM) [2]:

$$\begin{aligned} \min \quad & \frac{1}{2} \sum_{k=1}^K \|\mathbf{V}_k - \mathbf{M}_k \cdot \mathbf{I}^k\|_2^2 \\ & + \lambda \sum_{u=1}^N \|\mathbf{I}_u\|_F + \gamma \sum_{u=1}^N \sum_{k=1}^K \|\mathbf{I}_{uk}\|_2 \\ \text{s.t.} \quad & \mathbf{I}_{uk} \geq 0, \quad 1 \leq u \leq N, \quad 1 \leq k \leq K, \end{aligned} \quad (3)$$

In particular, for a specific contagion  $k$ , one can identify the most influential nodes  $\mathcal{U}_k$  with respect to this contagion as:

$$\mathcal{U}_k = \{u : \|\hat{\mathbf{I}}_{uk}\|_2 > 0\},$$

where  $\hat{\mathbf{I}}_{uk}$  is the optimal solution of (3).

With the estimated  $\hat{\mathbf{I}}_{uk}$ , one can predict the total volume of the contagion  $k$  at  $T + 1$  by  $\hat{V}_k(T + 1) = \sum_{u=1}^N \sum_{l=0}^{L-1} M_{u,k}(T - l) \hat{I}_{uk}(T + 1)$ . Volume  $V_k(t)$  of a contagion  $k$  can be viewed as a time series in  $t$ . We thus evaluate MSLIM on a time series prediction task

using the prediction mean-squared error (MSE) as the evaluation metric:

$$\text{MSE} := \frac{1}{K} \sum_{k=1}^K \left( \frac{1}{T} \left( \sum_{t=1}^T (\hat{V}_k(t) - V_k(t))^2 \right) \right),$$

Where  $\hat{V}_k(t)$  is the predicted total volume for the contagion  $k$  at the time  $t$  using the data from previous time.

In (1),  $V_k(t)$  models the total volume over the entire nodes (e.g. all Twitter users), while  $\mathcal{N}$  denotes a small subset of nodes of interest. In particular,  $V_k(t)$  is a function of a small active node set. Thus, defining this active set is the first step in applying MSLIM for contagion-sensitive influential nodes detection. In [2], we simply define the active node set as the twitter users with at least 1,000 tweets during the certain time periods. However, this may not be optimal strategy. Twitter offers the API [12] to crawl the profile for each individual twitter user, including the full name, followers count, the location, friends count, the account created time, etc. It is a feasible way to determine active node sets by different user profile properties, e.g. choosing the twitter users located at North American as the active node set.

### B. Slow Intelligence System

A slow intelligence system (SIS) [4] is a general-purpose system characterized by being able to improve performance over time through a process involving enumeration, propagation, adaptation, elimination and concentration. A SIS is a system with multiple decision cycles such that actions of slow decision cycle(s) may override actions of quick decision cycle(s). The main purpose of SIS system is to test multiple/single approaches in each cycle and choose algorithms with better performance and discard the worse ones. Algorithms are classified into different cycles. Algorithms within one cycle do the same job for the same input testing data. Cycle  $i$  and Cycle  $i + 1$  can do the same job or different jobs. If two consecutive cycles do the same job, cycle  $i + 1$  should improve the performance with the advanced algorithms or hybrid approach which adopts the advantages of algorithms from previous cycle  $i$ . A SIS system typically includes the following five system components: Enumerator, DataSender, Verifier, Eliminator and TimeController. Enumerator enumerates algorithms on cycle basis. Algorithms could be instantiated sequentially or in parallel. DataSender sends testing data to algorithm instances created by Enumerator. Verifier evaluates algorithm instance outputs and sends evaluation result to Eliminator. Eliminator eliminates algorithm instances by their performances on cycle basis. TimeController determines which cycle(s) to execute.

A component-based SIS architecture and operational characteristics are presented in [5]. We also developed a visual specification approach using dual visual representations, and designed the user interface to produce and manage this dual visual representations for component-based SIS system [6].

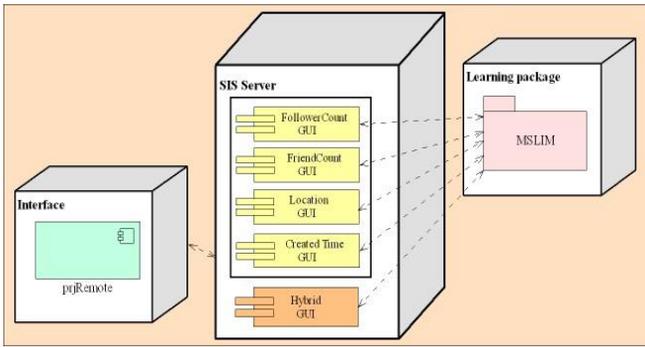


Figure 1. SIS-Based Active Node Sets Definition System

### III. SIS-BASED ACTIVE NODE SETS DEFINATION SYSTEM

#### A. System Architecture

The SIS-based active node sets definition system is illustrated by the UML deployment diagram shown in Fig. 1. The SIS Server controls slow intelligence system components such as Enumerator, Eliminator, Concentrator and Time Controller. The SIS server provides different GUIs with the visualization techniques to help define active node sets. There are two cycles in SIS system. The first cycle contains four GUI interfaces while the second cycle has one hybrid GUI interface. Each GUI communicates with a learning software package which computes the indicator function matrix  $M_k$  of the active nodes set and implements the MSLIM model. A universal interface prjRemote is provided for debugging purpose.

#### B. GUI interfaces for determining active node sets

In cycle one, there are four individual GUI interfaces enumerated by SIS system. Each GUI utilizes one property of twitter user's profile to help define active node sets, e.g. follower count, friend count, location, created time. We apply proper visualization techniques in each GUI to facilitate the task. Three GUIs, follower counts, friend counts and created time use histogram to visualize the distribution of defined active node sets while location GUI uses the choropleth map [13], as shown in Fig. 2. Each GUI has following functions: 1) enable designer to define the range of corresponding property and filter the data set. For example, designer can select the active users with a certain range of followers count from 5000 to 15000, as shown in Fig. 2(a). In GUI, the left panel lists the filtered twitter user list and the right panel shows the active users that designer selected. 2) For each twitter user, GUI shows the short biography on the bottom which helps designer decide whether to choose it as the active node. 3) "Build" button is to create a text file, which contains the user names of active node set in right panel. This is the output of our system in the first step. 4) Designer can simultaneously visualize the distribution according to each property when the active node sets is updated, as shown in Fig. 2 right column. 5) After the active node set is determined, designer can click on "Result" button to execute our MSLIM algorithm. The GUI can show the final predicted MSE to help designer evaluate his

strategy and further improve the way to choose more appropriate active user set.

In cycle one, we only use each individual property of twitter user's profile to facilitate designer defining active node sets. However this is too limited. Considering four attributes together, we can state the logical conditions to filter the twitter users. For example, if designer hope to define active user sets that have less than 10000 followers count and created from 2011 to 2012 and located in USA. The designer can firstly formulate three conditions: C1: followers count < 10000; C2: Created time  $\in$  [Sat Jan 01 00:00:01 2011, Sat Dec 31 23:59:59 2012]; C3: location = "USA" and then use boolean operators "AND" and "OR" to define the above logic as C1 AND C2 AND C3. Thus in cycle two, we develop a hybrid GUI interface incorporating four properties together and allow designer to define active node sets by any logic conditions, as shown in Fig 3. The conditions can be combined randomly or by user defined logic rules.

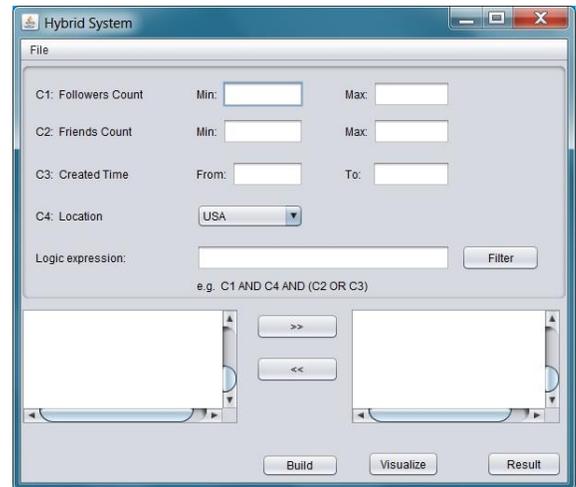
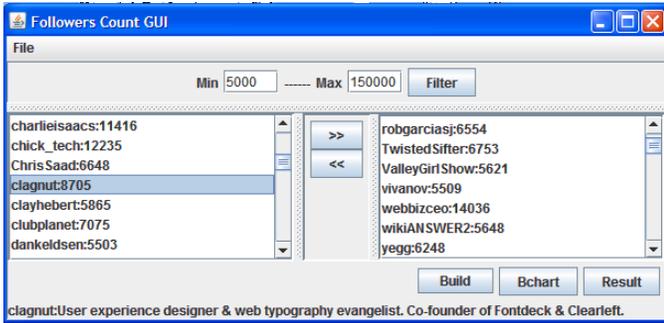


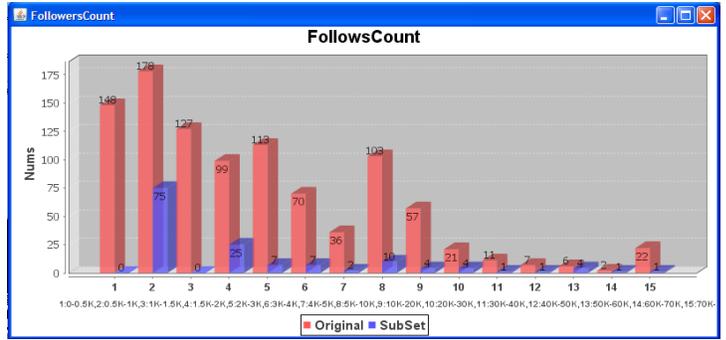
Figure 3. Hybrid GUI Interface

#### C. Scenario of SIS-based active node set defination system

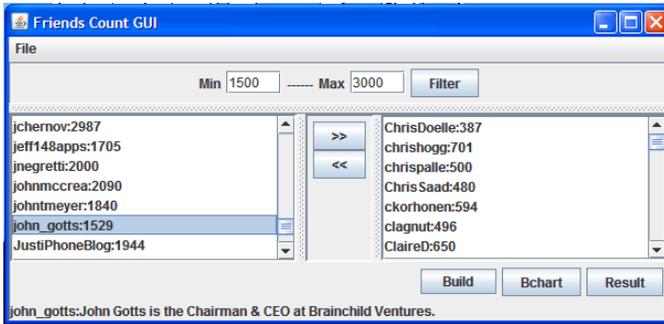
We describe the operation process of the whole system which includes two cycles. SIS server accepts the entire twitter users list and enumerates four GUI interfaces according to four properties of twitter user profile. In cycle one, designer can design the arbitrary conditions for each property in individual GUI and choose the active user sets to run MSLIM model. For each GUI, designer can try different strategies to define active user set until the predicted MSE satisfies the designer's expectation (e.g. a certain threshold). Meanwhile, designer can use histogram or colorpleth map to analyze the distribution of selected active user set and fine tune his strategy to meet the desired distribution for certain profile property. In the end of cycle one, each GUI has one final proper condition rule  $C_i$ ; to choose active node sets and the SIS server selects the optimal condition rule based on predicted MSE.



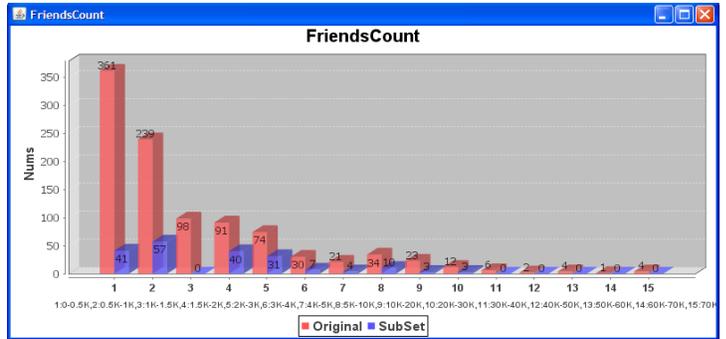
(a) Followers Count GUI



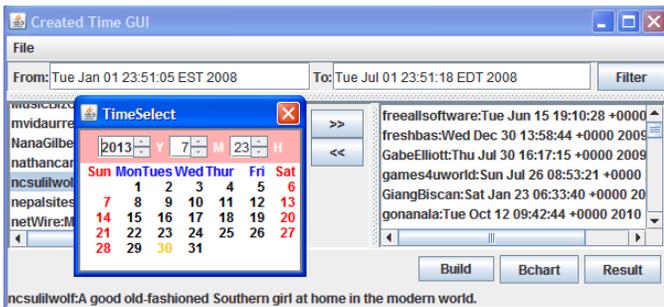
(b) Histogram Visualization for Follows Count



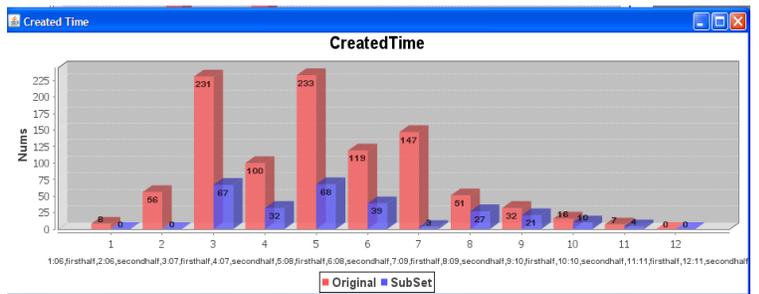
(c) Friends Count GUI



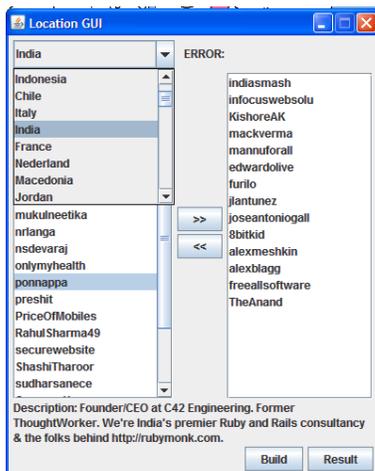
(d) Histogram Visualization for Friends Count



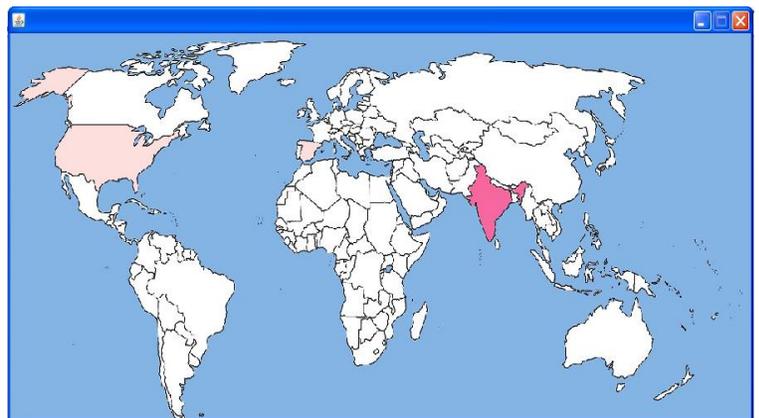
(e) Created Time GUI



(f) Histogram Visualization for Created Time



(g) Location GUI



(h) Colorpleth world map for location

Figure 2. GUI Interfaces with visualization methods for determining active node sets.

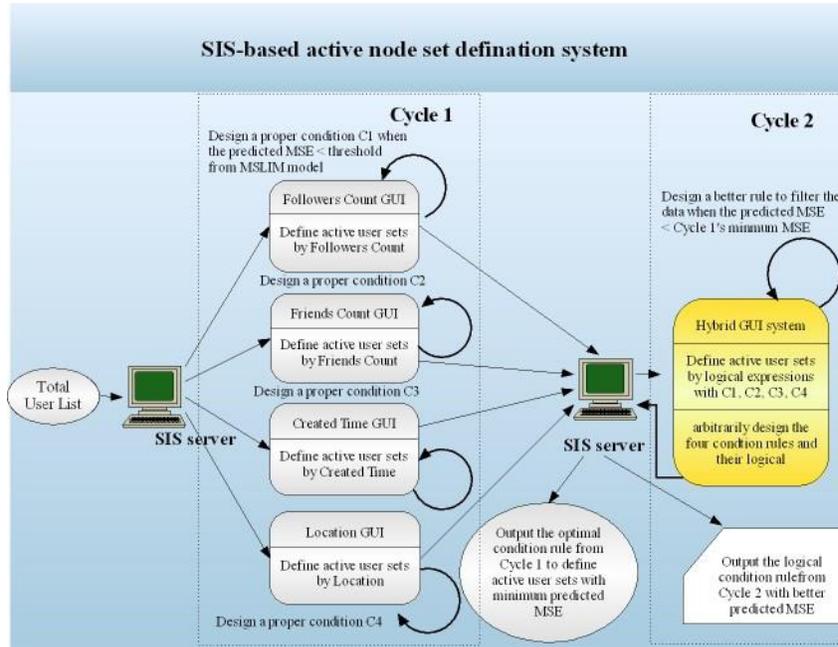


Figure 4. Scenario of SIS-based active node set definition system

In cycle two, the hybrid GUI interface enables designer define any rules by considering four properties together. There are two scenarios:

- Scenario 1: By utilizing the conditions (strategies) C1, C2, C3, C4 developed in cycle one, designer can design any logic expressions and filter the user list to select the active user sets.
- Scenario 2: designer can freely design the four conditions based on four attributes (Followers Count, Friend Count, Created time, Location) and apply logic operators (AND and OR) on them to filter the user list.

Cycle two can be terminated until the predicted MSE from active user sets is smaller than the minimum MSE achieved in cycle one or the improvement of MSE satisfies the designer's requirement.

#### IV. EXPERIMENT

In this section, we evaluate the performance of our system on the twitter data set. Twitter is an online social network used by millions of people around the world to stay connected to their friends, family members and co-workers through their computers and mobile phones.

##### A. Dataset Collection

We follow our work [2] to crawl and collect the Twitter data. In particular, we extracted a list of 1000 Twitter usernames from TechCrunch (<http://techcrunch.com/>). Using the Twitter search API [12], we crawled all the tweets of these twitter users between January 2009 and November 2011, which include the full text, the author, and the time-stamp for each tweet. In addition, we collected the profile

for each individual user, including followers count, friends count, the created time, the location, a short biography, etc. We took the LDA [14] to extract interesting topics from tweets and set the number of topic  $K = 50$ .

##### B. Experimental setup

In the experiments, we set one day as the time unit, and set the time lag  $L = 5$  (i.e., influence of a node decays to zero after 5 days). We use our SIS-based active node sets definition system to determine the active user sets. Then we can directly construct the matrix  $M_k$ , where  $M_{u,k}(t)$  is the number of times that the active user  $u$  mentioned the topic  $k$  in  $[t - 1, t]$ . The volume  $V_k(t)$  is defined by the total number of tweets that the topic  $k$  appears in  $[t - 1, t]$  over the entire 1,000 users. We can apply our MSLIM model to compute predicted MSE and simultaneously identify the topic-sensitive influential users. The detailed experimental setting of MSLIM model is explained in our previous work [2]. In this experiment, we focus on the first step of MSLIM, i.e. active user sets definition.

In cycle one, we test the following experimental scenarios as shown in Table I. For each property, we develop the particular ranges as conditions in individual GUI to filter the total user list data. For each range, we select the entire users as the active node sets and run MSLIM to get the predicted MSE. These ranges are designed to ensure the number of users follows uniformly distribution based on each property. In the experiment, designer can design any conditions (data ranges) and select any subset of users in each range. Here, we intend to compare the experimental results when we use our system to select active node sets based on different ranges of entire data. From table I, for each GUI, designer compares different ranges of active node sets and selects the proper conditions indicated in bold when

setting predicted MSE threshold as 15. We call these four appropriate conditions as C1, C2, C3, C4. In the end of cycle one, SIS server can select the optimal strategy that the active node set are located in North America.

TABLE I. PREDICTED MSE OF DIFFERENT CONDITIONS IN CYCLE ONE

<b>Follower Count</b>	(0,1500]	(1500,3000]	(3000,10000]	>10000
	19.741	<b>13.939</b>	15.745	16.477
<b>Friend Count</b>	(0,500]	(500,1000]	(1000,3000]	>3000
	20.275	<b>14.185</b>	15.599	15.096
<b>Created Time</b>	Before 2007	(2007,2008]	(2008,2009]	After 2009
	<b>14.161</b>	16.114	16.298	19.333
<b>Location</b>	North America	Europe	Asia	Other areas
	<b>13.079</b>	18.594	15.372	15.788

In cycle two, we use C1, C2, C3, C4 in previous cycle and construct the logical expressions with these four conditions, as shown in Table II. We design five different rules to filter the user list and construct the active node sets. Note that designer can develop any other rules over these four conditions and also construct new conditions C1', C2', C3', C4' for each property and combine them in logical expressions. The first two rules are designed in the purpose of comparing the union set and intersection of four conditions. Since C4 is the optimal strategy followed by C1 in cycle one, thus we make the third rule to satisfy C4 and any one in [C1, C2, C3] and rule 4 & 5 are to make boolean operations over these two optimal and suboptimal conditions. From table II, the union set of these four conditions achieve the best performance, which is better than the optimal result in previous cycle. The result is also superior to the result in our previous work [2], where we selected the active twitter users with at least 1,000 tweets during the certain time period. The result indicates that this active node set is more suitable for MSLIM model in our twitter dataset.

TABLE II. PREDICTED MSE OF DIFFERENT CONDITIONS IN CYCLE TWO

	Predicted MSE
$C_1 \cap C_2 \cap C_3 \cap C_4$	N/A (Empty set)
$C_1 \cup C_2 \cup C_3 \cup C_4$	<b>12.988</b>
$C_4 \cap (C_1 \cup C_2 \cup C_3)$	14.589
$C_1 \cup C_4$	13.035
$C_1 \cap C_4$	15.896

## V. CONCLUSIONS AND FUTURE WORKS

In this paper, we develop an efficient SIS-based active node sets definition system to facilitate MSLIM model for prediction and influential user selection task in an implicit information diffusion network. By utilizing slow intelligence system, we develop two cycles which enable designer

evolutionarily searching for a proper condition to construct active node sets to improve MSLIM model performance. We also incorporate two visualization techniques histogram and choropleth map in GUIs to help designer analyze the distribution of selected active node sets and further fine-tune his strategy. The result shows that our system can efficiently help determine the active node sets and incrementally improve the predicted mean square error of MSLIM model in two cycles. One future work is to extend our system to other social network datasets. The key challenge is that the user profile properties are different for various datasets. One needs to develop the particular GUI to visualize the different property. But our SIS-based system is very flexible to incorporate the individual GUI as a component in the whole system.

## REFERENCES

- [1] Yingze Wang, Guang Xiang, and Shi-Kuo Chang, "Sparse Linear Influence Model for Hot User Selection on Mining a Social Network", Proc. of 24<sup>th</sup> International Conference on Software Engineering and Knowledge Engineering, San Francisco Bay, California, July 1-3, 2012.
- [2] Yingze Wang, Guang Xiang, and Shi-Kuo Chang, "Sparse Multi-task Learning for Detecting Influential Nodes in an Implicit Diffusion Network", Proc. of 27<sup>th</sup> AAAI Conference on Artificial Intelligence, Bellevue, Washington, July 14-18, 2013.
- [3] Jaewon Yang and Jure Leskovec, "Modeling information diffusion in implicit network", IEEE 10th International Conference on Data Mining (ICDM), pp 599 – 608, 2010.
- [4] Shi-Kuo Chang, "A General Framework for Slow Intelligence Systems", International Journal of Software Engineering and Knowledge Engineering, Vol. 20, No. 1, pp 1-16, February 2010.
- [5] Shi-Kuo Chang, Yao Sun, Yingze Wang, Chia-Chun Shih and Ting-Chun Peng, "Design of Component-based Slow Intelligence Systems and Application to Social Influence Analysis", Proceedings of 2011 International Conference on Software Engineering and Knowledge Engineering, Miami, USA, July 7-9, pp 9-16, 2011.
- [6] Shi-Kuo Chang, Yingze Wang and Yao Sun, "Visual Specification of Component-based Slow Intelligence Systems", Proceedings of 2011 International Conference on Software Engineering and Knowledge Engineering, Miami, USA, July 7-9, pp 1-8, 2011.
- [7] Ilyas, M.U and Radha, H, "Identifying Influential Nodes in Online Social Networks Using Principal Component Centrality", IEEE International Conference on Communications (ICC), pp 1-5, 2011.
- [8] Hao Ma , Haixuan Yang , Michael R. Lyu , Irwin King, "Mining Social Networks Using Heat Diffusion Processes for Marketing Candidates Selection", Proceedings of the 17th ACM conference on Information and knowledge management (CIKM '08), New York, pp 233-242, 2008.
- [9] Masahiro Kimura, Kazumi Saito, Ryohei Nakano, and Hiroshi Motoda, "Extracting influential nodes on a social network for information diffusion", Data Min. Knowl. Discov. Vol. 20, No. 1, pp. 70-97, 2010.
- [10] David Kempe and Jon Kleinberg and Éva Tardos, "Influential Nodes in a Diffusion Model for Social Networks", IN ICALP, pp. 1127-1138, 2005
- [11] J. Weng, E.-p. Lim, J.Jiang, and Q. He, "Twitterrank: finding topic-sensitive influential twitters", Proc. of the third ACM international conference on web search and data mining, 2010.
- [12] Twitter Search API. <http://apiwiki.twitter.com/Twitter-API-Document>.
- [13] T. Slocum, R. McMaster, F. Kessler, H. Howard (2009). Thematic Cartography and Geovisualization, Third Edn, pp 85-86. Pearson Prentice Hall: Upper Saddle River, NJ.
- [14] Blei, D. M.; Ng, A. Y.; and Jordan, M. I. 2003. Latent Dirichlet Allocation. Journal of Machine Learning Research 3:993-1022.

# Method for Measuring Twitter Content Influence

Euijong Lee  
Dept. of Computer and Radio  
Communications Engineering  
Korea University  
Seoul, Republic of Korea  
Email: kongjjagae@korea.ac.kr

Jeong-Dong Kim  
Dept. of Computer and Radio  
Communications Engineering  
Korea University  
Seoul, Republic of Korea  
Email: kjd4u@korea.ac.kr

Doo-Kwon Baik  
Graduate School of Convergence IT  
Korea University  
Seoul, Republic of Korea  
Email: baikdk@korea.ac.kr

**Abstract**—Twitter is a microblogging website with specific characteristics not found in other social network services. This platform contains a good deal of valuable content, and users can access this content using Twitter search. However, Twitter search returns only time-descending ordered content including keywords. Thus, we propose a linear-time method of measuring the influence of Twitter content considering not only time, but also characteristics of each Twitter account. In analyzing these characteristics, we have found that the number of retweets can measure shareability, while the number of followers held by the content author can measure spreadability. We perform experiments using real Twitter data for proving the effectiveness of the proposed method. We demonstrate that this proposed method is effective at finding up-to-date content. Further, in comparing our method with analysis via PageRank, we demonstrate that our method is more effective at accurately measuring influence.

**Keywords**—Twitter; Contents Search; Retweet; Follower, Contents Influence

## I. INTRODUCTION

Twitter is a microblogging website that allows only posts of 140 characters or less (called tweets) to create content. This service has a well-defined markup vocabulary. Unlike other social networking services, however, Twitter supports one-sided relationships between users. If one user wants to view another user's contents in real-time, that user can add the other user to the user's social network list without approval of the other user. This behavior is referred to as 'following' the other user, with the following user called the 'follower'. The main sharing mechanism of content on twitter is what is referred to as 'retweeting'. A retweet keeps the information of the original content while also including the opinion of users who retweeted that content. Those simple mechanisms, though restricted to making various types of contents, but those lead to the development of Twitter. Those are easy to make sharing of information on Twitter and, making it easy to writing content succinctly.

In 2012, about 140 million tweets a day were created on Twitter [1]. Literally, it is very big data, and therefore, of course, users face problems in finding useful information on this social network [5, 8]. Twitter does provide a content search service, through which users can search for content on Twitter by entering relevant keywords and receive the corresponding content in a time-descending order (see figure 1). Twitter also

provides advanced search offering four categories of options: 'words', 'people', 'places', and 'other'. The 'words' category provides options related to specific keywords, the 'people' category provides options related to specific accounts, the 'places' category provides options related to specific locations, and the 'other' category provides options limiting results to those including positive or negative emoticons or to questions, as well as an option to include otherwise-excluded retweets in the results (see figure 2). Still, advanced search also provides content only in a time-descending order, with no attention to relevancy. Up-to-date information is important in Twitter; however, Twitter search needs to consider not only time, but other characteristics as well [3, 5, 8]. We focus on the problem of improving Twitter search, and in this research, we propose a method of measuring the influence of individual Twitter content using characteristics of individual Twitter accounts in content search.

## II. RELATED WORKS

The most representative studies on Twitter analyze characteristics like follower count, followee count, and retweet count [2-5]. There exists considerable research on Twitter search based on these characteristics.

[2] attempted to measure user influence of Twitter accounts, finding three contributory factors: number of followers, number of retweets, and number of mentions. A large number of followers implies a large audience for the user. More specifically, it means a large audience receives a user's content directly. Number of followers, then, measures a user's ability to spread information. Number of retweets demonstrates the ability of the user to make content with pass-along value; a retweet means that there is worth to be shared. Number of mentions demonstrates the ability of the user to relate to others in a conversation; it shows advertisement value.

[3] researched the possibility of Twitter serving as a news media. They found that number of retweets is the chief measure of user influence, and number of followers is the chief measure of user popularity. Further, they found that trends on Twitter match up with new media trends fairly closely (about 85%); as a result, they claimed that the role of Twitter is not only social networking service, but also news media.



Figure 1. Results of Twitter search

[5] researched differences between microblog search and web search by analyzing search log information. They found that users using Twitter search want to find ‘timely information’, ‘social information’ and ‘topical information’. Queries in Twitter searches are shorter than web searches, but words are longer, and Twitter search users use site-specific grammar (for example, using ‘@’ or ‘#’). [8] suggested that Twitter search should reflect characteristics of the users. In surveying users, they found that users of Twitter search most often want to find events, trending topics, or specific people.

In this research, we use characteristics of Twitter extracted in previous research to produce a method for measuring the influence of a single post that can be used via Twitter search and mining.

### III. CHARACTERISTICS OF TWITTER AND A METHOD OF MEASURING CONTENT INFLUENCE

We want to solve the problem that Twitter content search only reflects time of posting. We chose characteristics of Twitter and created a method for measuring the influence of content using these characteristics. In III.A, we describe the specific characteristics of Twitter that are used in our proposed method. In III.B, we explain our proposed method.

#### A. Characteristics of Twitter

Content influence is a value that measures to what degree a piece of content contains meaningful information for users. We choose three factors for measuring influence. The first factor is spreadability of content. In Twitter, content is delivered to the author’s followers; thus, the number of followers shows how many users were delivered that content. Previous research showed that the number of followers measures popularity of

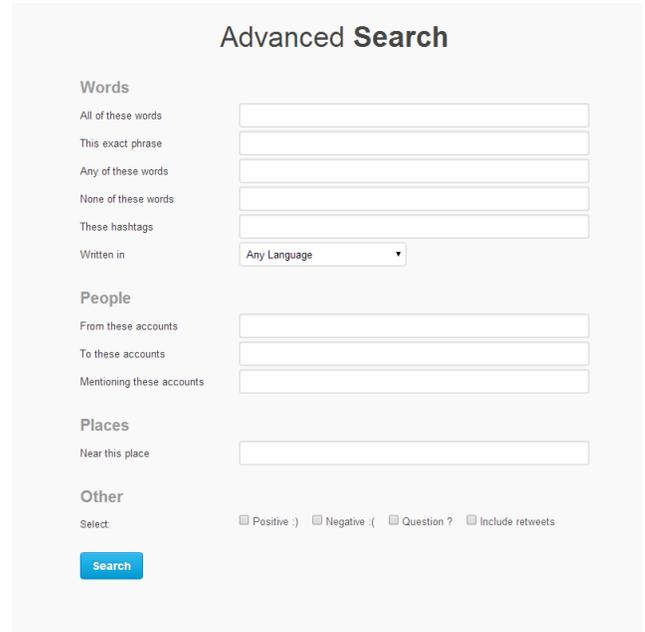


Figure 2. Twitter Advanced Search

author [3, 5]. Therefore, we can say it is similar for the ability to spread information, and therefore, we use the number of the author’s followers as spreadability. The second factor is the value of content information. Value of information can be measured using number of times shared, because if the content is valuable, it will be shared with others. In Twitter, the retweet mechanism is the main method for sharing content, and therefore, we use retweet count to measure this factor. The last factor is the currency of the information. Newer content contains more valuable information than older content. Twitter is sensitive to up-to-date information [3, 8] and Twitter search shows results by time-descending order. This factor is very sensitive to use in our proposed method, because as we pointed out, this is a problem of Twitter search. However, it is a very important factor in news media and we treat this very carefully. We reflect this factor using the time information of the content.

#### B. Method for Measuring Content Influence

We use these three characteristics of Twitter in our proposed method: follower number as spreadability of content, retweet count as the value of content information, and time written as currency of information. The proposed method using these factors is as follows.

$$I(C_i) = \alpha \log(RT_i + 1) + \beta \log(F_i + 1) + \gamma \log \frac{k}{NT - WT_i} \quad (1)$$

$$\alpha + \beta + \gamma = 1$$

Here,  $C_i$  is the  $i$ th piece of content, and  $I$  is the influence of content  $C_i$ . In this equation, ‘ $\alpha \log(RT_i + 1)$ ’ represents shareability of  $C_i$ , where  $RT_i$  is the retweet count of  $C_i$ . We take a logarithm function to normalize, and we add 1 to ensure the output is always defined, because if we do not add 1, and retweet count is 0, then it will be negative infinity.  $F_i$  is the follower number of the author of  $C_i$ , and therefore, ‘ $\beta \log(F_i + 1)$ ’ represents the spreadability of  $C_i$ . It takes a logarithm

function for the same reason as the previous factor.  $\gamma \log \frac{k}{NT-WT_i}$  represents currency influence, where  $NT$  means ‘now time’ and  $WT_i$  means ‘written time of  $C_i$ ’. We calculate interval of time as a real number, where subtraction of hours is an integer value and subtraction of minutes is a decimal place. It also takes a logarithm function for the same reason as the previous two factors. We take the reciprocal to ensure that up-to-date contents have a larger contribution than out-of-date contents. Taking a reciprocal, however, has the disadvantage that the currency factor becomes much smaller than the other factors. To correct this disadvantage, we multiply by a constant ‘ $k$ ’. In the equation ‘ $y = \log \frac{k}{x}$ ’, when  $x$  equals  $k$  then  $y$  is 0, and if  $x$  is smaller than  $k$ , then  $y$  takes a negative value. Therefore, if a user wants to find information about posts in the last  $k$  hours, then they need only fix the value of  $k$  in the equation. Finally,  $\alpha$ ,  $\beta$  and  $\gamma$  are mediators that adjust the power of each factor in the equation. We ensure that the summation of these terms is 1 in order to prevent one factor from being too much more influential than the others.

#### IV. EXPERIMENTS AND EVALUATION

We crawled Korean Twitter contents and set up experiments using that data. We describe the crawled dataset and experiments in IV.A, and then show the result of an experiment for content influence using our method in IV.B.

##### A. Dataset for Experiments

We used a crawled contents of Korean Twitter contents and user relation (followee information) of Korean users from July 1, 2012 to July 31. The size of the content dataset is about 93.1GB, and the size of the user information dataset is 12.3GB. We saved content data using JSON data style, and user data by simple text style. Content data includes various pieces of information about a single piece of content: author, retweet count, time written, etc. (see figure 3) User data consisted of user-followee relationships (see figure 4)

For our experiments, we extracted related content in two different domains: smartphones (Galaxy series and iPhone) and Psy (a Korean pop singer). We chose the first subject because the Galaxy series and the iPhone are presently the most popular models of smartphone, and the Galaxy S3 was issued in July 2013. We chose the second subject because Psy released a new

```

"documentId": "219082875629342720", "content": "그런데서 느낀건...정출 잠시 끌어와주세요", "documentDateTime": "20120701000002",
"documentId": "219082900518342656", "content": "오복 생일 축하합니다~", "documentDateTime": "20120701000008", "registerDateTime":
"documentId": "219082919618004674", "content": "시범 공개", "documentDateTime": "20120701000013", "registerDateTime": "20120701
"documentId": "219082971900079609", "content": "할인 노트 아예 팔면 어췌 수 있냐?", "documentDateTime": "20120701000018", "register
"documentId": "219082978598982894", "content": ".....아직도 록마루 거 길다는게 황경", "documentDateTime": "20120701000022",
"documentId": "219082923050399649", "content": "12시", "documentDateTime": "20120701000019", "registerDateTime": "201207010000
"documentId": "21908292120762944", "content": "어머정말이...얼두시야 ㅠㅠ", "documentDateTime": "20120701000029", "registerDateTi
"documentId": "219082943128272896", "content": "내일 필요함", "documentDateTime": "20120701000018", "registerDateTime": "201207
"documentId": "219082864904196881", "content": "8Hmsenkanngis 우리 같이 레시피연구실공들으려가요 !!!!!", "documentDateTime":
"documentId": "21908286503443106", "content": "#special104 생일 축하합니다!!!!사랑합니다 ㅎㅎ #100Day", "documentDateTime
"documentId": "2190828650468984613", "content": "#special104 생일 축하합니다!!!!오복 오복에게 좋은 날이되기를기쁘고 있습니다

```

Figure 3. Example of contents files

```

14055644 90420314, 40511475, 50374439, 30973, 58528137, 11348282
14055664 30973, 759251, 18073211, 13
14055704 17093617, 3108351, 18479513, 15907720
14055744 14749606
14055804 90420314, 6449282, 15907720
14055824 16228398, 92367751, 33995409, 20, 20221159, 7017692, 39288259, 110827653, 12, 13, 5988062
14055904 13876182, 15962096, 58166411
14055964 18676177

```

Figure 4. Example of relationship files

album, ‘Gangnam Style’, which has become a popular topic worldwide. We collected posts about smartphones (especially the Samsung Galaxy series and iPhone) and Psy using related Korean and English words: 12 for smartphones, and 5 for Psy (see table 1).

TABLE 1. RELATED WORDS IN THE TEST DATA SET

Topic	Related Korean and English words
Galaxy Series	갤럭시, 갤럭시, 겔스, 겔스, 겔노트, 겔노트, 겔넷, 갤럭시, galaxy (not case sensitive)
iPhone	아이폰, iphone, I-phone (not case sensitive)
Psy	싸이, 강남 스타일, 강남스타일, Psy, Gangnam Style (not case sensitive)

The number of collected smartphone-related posts is 207,022, and the number of collected Psy-related posts is 118,521. We extracted 10 times more retweeted content, and deleted content for which we could not get a user profile for effective experimentation. Finally, we selected 558 posts about smartphones and 421 about Psy.

After resizing, we classified posts by the nature of the user’s identity into one of four categories: public user, personal user, bot, and unclassified data. A public user is a spokesperson of a specific community or group. For example, ‘samsung’, ‘SBS8news’ and ‘YTN24’: ‘samsung’ is the ID of the SAMSUNG Corporation, and ‘SBS8news’ and ‘YTN24’ are broadcasting companies in the Republic of Korea. A personal user is a personal user’s ID. A bot is a program used to produce automated content, generally called a Twitter bot. Finally, unclassified refers to any IDs whose category could not be determined. Contents as classified by user identity are presented in table 2.

TABLE 2. CLASSIFIED DATA SET BY USER’S IDENTITY

Topic	Public	Personal	Bot	Unclassified
Smart phone	205	198	18	37
Psy	115	172	36	98

We also classified posts according to their relation to the keywords, sorting into one of four categories: direct, indirect, private, and unrelated. A direct post is directly related to the topic. An indirect post does not contain direct information about the topic, but does have relevant information such as a related event, music video or so on. A private post contains personal information. Finally, an unrelated post contains information unrelated to the topic. Classification of relation to keywords is presented in table 3 with examples of content type in table 4.

TABLE 3. CLASSIFIED DATA SET BY INFORMATION OF CONTENTS

Topic	Direct	Indirect	Private	Unrelated
Smart phone	166	157	130	105
Psy	205	298	18	37

TABLE 4. EXAMPLE OF CONTENTS DIVIDED BY INFORMATION

Type of relationship	Contents	
Directed	Korean	갤럭시 S3' 발화 사건, 단순 해프닝으로 종결
	English	Galaxy S3 bun-in incident is ended just happening
Indirect	Korean	[대한민국 올림픽 선수단 응원 이벤트] 응원 메시지 보내고, 금메달 개수를 맞추세요. 갤럭시 S3의 주인공이 되실 수 있습니다.
	English	[Korean Olympic team cheers up event] Send a cheer up message and guess the number of Olympic gold medals. You can be an owner of Galaxy S3.
Private	Korean	RT))여수엑스포에서 갤럭시노트 화이트 잃어 버렸어요 엑스포에 분실 신고 했으니깐 거기에 맡겨주세요 제발제발 부탁드립니다.
	English	RT))I lost my white galaxy note in Yeosu Expo. If you find my phone, please leave it lost-and-found center.
Unrelated	Korean	갤럭시 익스프레스의 미국투어 다큐 <만드시 크게 들을 것>
	English	The America tour documentary of Galaxy Express <Listen loudly>

B. Experiments

Before our experiments on the applicability of our proposed method, we first had to demonstrate that there is a difference between retweet in-degree and follower in-degree, as if those two in-degrees are linearly correlated; we do not need to use both factors. We thus first performed an experiment on the independence of retweet and follower in-degree, and then perform our experiment on the effectiveness of the proposed method.

1) Independence of retweet count and follower number : First, we used Pearson correlation coefficient as a measure of correlation between two data sets (See equation (2))

$$r = \frac{\sum_{i=1}^n (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^n (X_i - \bar{X})^2} \sqrt{\sum_{i=1}^n (Y_i - \bar{Y})^2}} \quad (2)$$

The Pearson correlation coefficient has a value between -1 and 1. If the absolute value of r is close to 1, there is a significant linear relationship, but if the absolute value of r is close to 0, there is no linear relationship between data sets. After performing the correlation test, we found that the coefficient of follower and retweet in-degree was close to 0, and therefore, follower and retweet in-degree had no linear relationship. (See table 5)

TABLE 5. RESULT OF PEARSON CORRELATION COEFFICIENT

Topic	r
Smart-phone	-0.0348392
Psy	0.17547095

2) Influence with retweet and followers: Next, we performed experiments to measure the level of shareability and spreadability, and interrelation of both. Before our experiments,

we created a modified PageRank based on follower-followee relationship for comparison with the proposed method [10], substituting followee for out-degree and follower for in-degree. The initial value of the modified PageRank is then the number of followers, and d-value is 0.85 as previous research (see equation 3) [10]. The number of smartphone domain authors is 312 (312 nodes, 12376 edges), and the number of Psy domain authors is 306 (306 nodes, 3004 edges). We calculated content PageRank using this equation about three times recursively.

$$PR(i) = \frac{(1-d)}{TotalNum\ of\ User} + d \left( \sum_{j=1}^{followerNum\ of\ i} \frac{PR(j)}{followerNum\ of\ j} \right) \quad (3)$$

For these experiments, we assumed that users want to find direct and indirect information [8]. In other words, users do not want to find private and unrelated information. Based on this assumption, we calculate F-measure with results as shown in figures 5 and 6 (in this experiment we do not consider time influence; therefore, γ is zero)

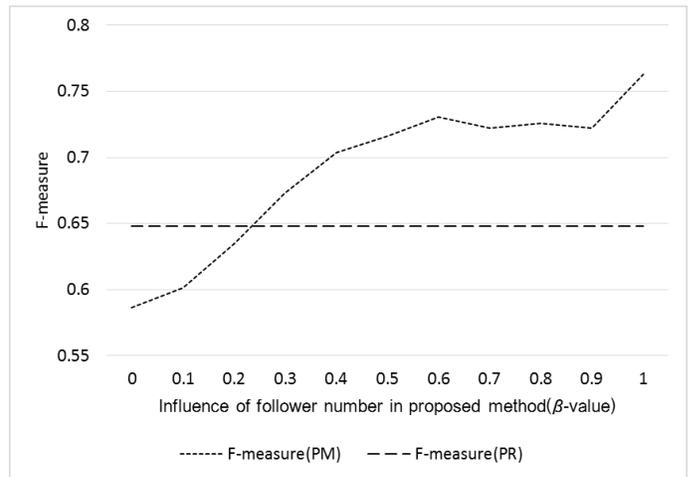


Figure 5. Result of Smart-phone Domain

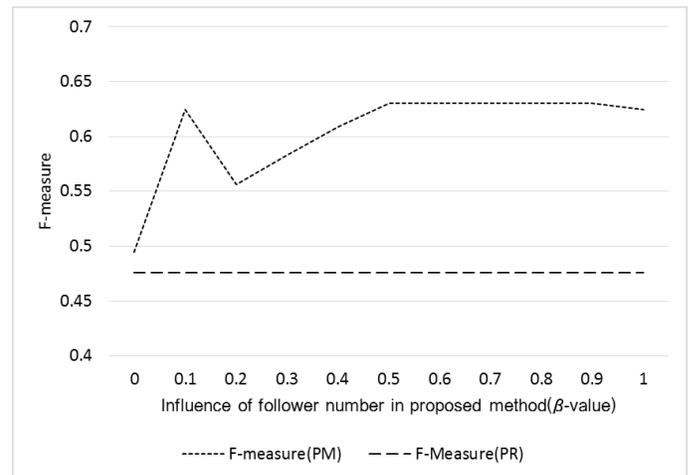


Figure 6. Result of The Psy Domain

In figures 5 and 6, the x-axis measures the value of the mediator term  $\beta$ , and the y-axis measures the value of the F-measure. Both results show that our proposed method is more effective than modified PageRank. In the case of the smartphone domain, the greatest F-measure value is occurred when  $\alpha$  (shareability) is zero, indicating that  $\beta$  (spreadability) may be more important than  $\alpha$ . However, as you can see from other values in the case of the smartphone domain and of the Psy domain (see figure 5 and 6), the shareability impact on measuring the influence of contents is greatest when  $\alpha$  is between 0.5 and 0.6. Therefore, we can find that retweet count and author follower number are both useful in measuring the influence of content.

As a note, the result of the Psy domain is worse than smart-phone domain because there is considerable garbage data in that dataset. The reason for garbage data is that there is a social network service, ‘Cyworld’, whose pronunciation ‘Cy’ is same as ‘Psy’ in Korean language, and whose Korean character is written the same, giving many false positives in the Psy domain. Nevertheless, despite the garbage data in the experiment set, the proposed method is still more effective than the comparison function.

Through this experiment, we have found that retweet count and follower number are both useful for measuring the influence of Twitter content, and without the recursive calculation needed in a method such as PageRank. Instead, it needs only linear time if the search system already knows some information about the content.

3) *Influence with time*: The influence of Twitter should reflect not only the influence of content but also how up-to-date the information is [5]. In this section, we want to discover time influence in Twitter content. More specifically, we want to discover time influence of Twitter content in specific k-hour (detailed meaning of ‘k’ is in the III.B). To experiment, we focused on a burn-in incident that occurred in the middle of July 2012. Burn-in is a failure of a screen caused by displaying the same image on the screen for an extended period, creating an after image that remains on the screen afterwards. The Galaxy S3 had a burn-in problem, but the manual of it contained a note stating that the company is not responsible for burn-in failure. However, the company corrected that sentence and announced a new policy that they will provide service for burn-in failure. We assume that some user wanted to find information about the burn-in problem and Galaxy S3 at July 11, 18:00 and that the user wanted to find up-to-date information over the last 5 hours from that time. We extracted contents that have Korean keywords–‘Galaxy’ and ‘image-remain’–and calculated each score using our proposed method both with time ( $\alpha=0.16, \beta=0.24, \gamma=0.6, k=5$ ) and without time ( $\alpha=0.4, \beta=0.6, \gamma=0, k=0$ ). The result of this experiment is shown below (see table 6).

The results show that if a user has many followers and high retweet counts, then time influence is not important (see C1-C5, C7-C8). However, if the user has a small number of followers and their retweet counts are very small, then their influence score is very small even if they have up-to-date

information about the topic (see C6). If content influence includes time, however, then this result is changed. C6 has the second smallest in the result without time influence even though it has up-to-date information. However, if the measurement of content influence considers time, C6 gets a higher score than older contents (C1-C4) even though they have more followers and a higher retweet count. This experiment shows us that our proposed method reflects time effectively with shareability and spreadability in k-hour.

TABLE 6. RESULT OF EXPERIMENT WITH INFLUENCE OF TIME

Content#	Content	Written Time	Re-tweet	Follower	Score with time	Score without time
C1	No AS for burn-in	7.10 12:36:38	27	872	0.475762	2.343471
C2		7.11 07:33:25	34	106320	1.261351	3.633598
C3		7.11 08:05:23	21	120437	1.255733	3.585427
C4		7.11 09:01:40	13	12733	1.015891	2.921430
C5	Offer AS for burn-in	7.11 16:19:09	64	122936	1.795269	3.778974
C6		7.11 16:35:05	12	1622	1.277328	2.371768
C7		7.11 17:42:14	34	135214	2.211606	3.696242
C8		7.11 17:51:59	19	142715	2.358965	3.613095

## V. CONCLUSION AND FUTUREWORKS

In this research, we hoped to aid Twitter content search that presently returns results in only time-descending order. To solve this problem, we analyzed characteristics of Twitter content, and proposed a method for measuring influence of each post. We used crawled data about Korean Twitter contents and user relation data from July 1, 2012 to July 31, extracting content on two subjects: smartphones and Korean pop singer Psy. We refined this extracted content, classifying by author’s identity (public, personal, bot, and unclassified) and type of information (direct, indirect, private, and unrelated). We then performed experiments using our proposed method, comparing it with PageRank to demonstrate that our approach is more effective and more computationally efficient in measuring influence of Twitter content, running in linear time on search results given associated information. We also performed experiments with our proposed method incorporating time influence, showing that our proposed method is effective when a post comes from a user with a smaller number of followers and a lower retweet count than other users, but still has more up-to-date information.

In the future, we would like to incorporate other information about the Twitter account, such as friendship of users, the feeling expressed in a post, and the closeness between users. Through this, we hope to create a more

advanced search and mining method for not only Twitter, but other social network services as well.

#### ACKNOWLEDGMENT

This research was supported by the basic Science Research Program through the National Research Foundation of Korea (NRF), funded by the Ministry of Education, Science and Technology (2011-0025588), and Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (2012M3C4A7033346). The corresponding authors is Doo-Kwon Baik

#### REFERENCES

- [1] Twitter, #numbers [Online]. Available: <https://blog.twitter.com/2011/numbers>, 2011
- [2] Cha, M., Haddadi, H., Benevenuto, F., & Gummadi, P. K., "Measuring User Influence in Twitter: The Million Follower Fallacy", ICWSM, pp10-17, 2010.
- [3] Kwak, H., Lee, C., Park, H., & Moon, S., "What is Twitter, a social network or a news media?", In Proceedings of the 19th international conference on World wide web, pp. 591-600, 2010.
- [4] Weng, J., Lim, E. P., Jiang, J., & He, Q., "Twitterrank: finding topic-sensitive influential twitterers", In Proceedings of the third ACM international conference on Web search and data mining, pp. 261-270 2010.
- [5] Teevan, Jaime, Daniel Ramage, Merredith Ringel Morris. "# TwitterSearch: a comparison of microblog search and web search.", Proceedings of the fourth ACM international conference on Web search and data mining. pp. 25-44, 2011.
- [6] Horowitz, Damon, Sepandar D. Kamvar. "The anatomy of a large-scale social search engine.", Proceedings of the 19th international conference on World wide web. pp. 431-440, 2010.
- [7] Carmel, D., Zwerdling, N., Guy, I., Ofek-Koifman, S., Har'El, N., Ronen, I., Chernov S., "Personalized social search based on the user's social network.", Proceedings of the 18th ACM conference on Information and knowledge management. pp. 1227 – 1236, 2009.
- [8] Golovchinsky, Gene, and Miles Efron. "Making sense of twitter search.", 2010.
- [9] M. Oussalah, F. Bhat, K. Challis, T. Schnier, "A software architecture for Twitter collection, search and geolocation services", Knowledge-Based Systems, Vol 37, pp. 105-120, 2012
- [10] Brin, Sergey, Lawrence Page. "The anatomy of a large-scale hypertextual Web search engine." Computer networks and ISDN systems pp. 107-117, 1998.
- [11] Carmel, D., Zwerdling, N., Guy, I., Ofek-Koifman, S., Har'El, N., Ronen, I., Chernov, S., "Personalized social search based on the user's social network", 18th ACM conference on Information and knowledge management, pp. 1227-1236, 2009.
- [12] A. Java, X. Song, T. Finin and B. Tseng, "Why We Twitter : Understanding Microblogging Usage and Communities", In Proc of 9th WebKDD and 1st SNA-KDD 2007 workshop on Web mining and social network analysis, 2007.
- [13] Kristina Lerman and Rumi Ghosh, "Information Contagion: n Empirical Study of the Spread of News on Digg and Twitter Social Networks", ICWSM, 2010. Web mining and social network analysis, 2007.
- [14] N. J. Belkin, "Some(what) grand challenges for information retrieval", SIGIR Forum, 42(1):p47–54, 2008.
- [15] M. J. Carman, M. Baillie, and F. Crestani, "Tag data and personalized information retrieval", In Procof the CIKM workshop on Search in social media, pp27–34. ACM, 2008.
- [16] D. Carmel, N Zwerdling, I. Guy, S. Ofek-Koifman, N. Har'el, I. Ronen, E. Uziel, S. Yogev and S. Chernov, "Personalized Social Search based on the User's Social Network", In Proc of CIKM '09, 1227-1236

# An Exploratory Search for Presentation Contents based on Slide Semantic Structure

Yuanyuan Wang  
Kyoto Sangyo University, Japan  
circl.wang@gmail.com

Yukiko Kawai  
Kyoto Sangyo University, Japan  
kawai@cc.kyoto-su.ac.jp

Kazutoshi Sumiya  
University of Hyogo, Japan  
sumiya@shse.u-hyogo.ac.jp

**Abstract**—MOOC is a crucial platform for improving education; students are able to browse various presentation contents through the Web. Any single presentation content can only cover a small fraction of knowledge in a specific domain by a given query, and thus offers a limited depth of information. Students then have to go through series of presentation contents, but this would be time-consuming and difficult to explore relevant information from various presentation contents. Therefore, we aim to build a novel exploratory search tool based on a meaningfully structured presentation, called “iPoster.” The system places elements such as text and graphics of slides in a structural layout with a zooming user interface (ZUI) by semantically analyzing the slide structure. Through this, iPoster can support students interactively browsing slides, for retrieving and navigating information from other presentation contents by considering the students’ browsing behavior. In this paper, we discuss two types of exploratory search, (1) focused searching based on well-matched browsing behavior that enables users obtain details of specific topics; and (2) exploratory browsing based on partially-matched browsing behavior that enables the users find various relevant information on topics of interest.

**Keywords**-exploratory search; presentation contents; iPoster;

## I. INTRODUCTION

Slide-based presentation tools, such as Microsoft PowerPoint or Apple Keynote is now one of the most frequently used tools for educational purposes. A huge amount of slide-based educational materials for MOOC, are freely shared on Web sites such as Coursera<sup>1</sup> and SlideShare<sup>2</sup>. Thus, not only students who missed a lecture or presentation, but also anyone interested in a topic can study the presentation on their own. Therefore, techniques are in demand that will efficiently find appropriate information worth learning from the vast numbers of presentations available. Although many techniques for searching and recommending presentation slides have been proposed, some problems remain from the viewpoint of exploratory search. One problem is current slideshow mode of presentations does not allow users operate freely on the presentations for stimulating the users’ interests. Recently, Prezi<sup>3</sup> utilizes a ZUI as an alternative

<sup>1</sup><https://www.coursera.org/>

<sup>2</sup><http://www.slideshare.net/>

<sup>3</sup><http://prezi.com/>

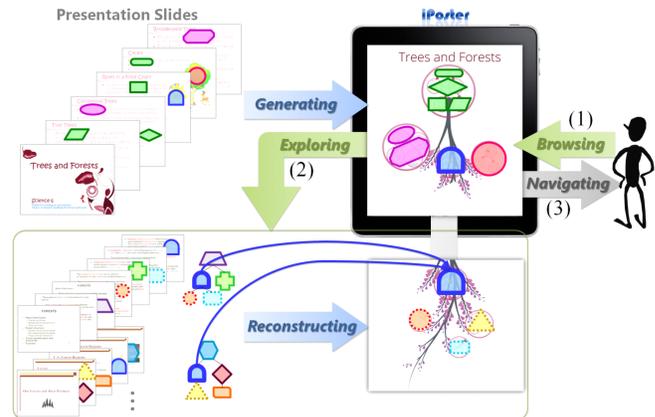


Figure 1. Conceptual diagram of an exploratory search tool by iPoster

to the traditional slides, allowing users easily operate the presentations. Another problem is any single educational material only cover a small fraction of knowledge in a specific domain by a given search query, and thus offers a limited depth of information. The users then have to go through various presentation contents, but this will be time-consuming and difficult to find relevant information from multiple presentation contents. Therefore, users will be required to browse them in structural layouts with ZUIs, and easily obtain information meets the users’ specific needs by considering user browsing behavior.

As depicted in Figure 1, we present an exploratory search tool that generates a meaningfully structured presentation by using the slides, which is called an iPoster. With our exploratory search tool, (1) users can interactively browse an iPoster, therefore, (2) the iPoster can explore information from other presentations by considering user browsing behavior; and (3) represent and navigate information meets the users’ specific needs. To achieve our goal, the iPoster can be implemented by 1) extracting textual and graphic elements in slides and semantic relationships between them; and 2) organizing elements in structural layouts with zooming and panning transitions based on a idea of Prezi. In semantic structure analysis, we first extract elements by examining the presentation context of the particular elements in the

slides. The semantic relationships between these elements are determined using implicit hyperlinks in slides, based on slide structures. Specifically, we derive the slide structure by focusing on the itemized sentences in the slide text. For providing an overview of the content, we utilize a hierarchical structure, combined with a stacked Venn. Finally, our iPoster is generated in a structural layout based on semantic relationships, using a ZUI, which can enable users to explore the presentations easily and efficiently.

The next section reviews related work. Section 3 describes our semantic structure analysis model. Section 4 explains iPoster generation. Our exploratory search on presentations and conclusions are given in Sections 5 and 6, respectively.

## II. RELATED WORK

A variety of applications address the weaknesses of the current slideware tools in the presentation and authoring domains. Our approach in an iPoster builds on the strength of exploratory search. Lanir et al. [1] proposed a MultiPresenter application that leverages spatial reasoning capabilities to relate content through dual-screen projection. Although the iPoster does not adopt the dual-audience-display paradigm, it addresses the need to navigate through elements dynamically during the presentation. NextSlidePlease [2] creates and delivers presentations in a nonlinear fashion. The iPoster is similar to this work, as we utilize a structural layout with the ZUI, rather than one or more slide lists, to allow users interactively browsing and automatically navigating.

Exploratory is constantly being changed and shaped by a range of related research. White et al. [3] suggests that browsing is a cognitive and behavioral expression of exploratory behavior and she claims that it has four elements: (1) glimpse a scene; (2) target an element of a scene visually and/or physically; (3) examine items of interest; and (4) physically or conceptually acquire or abandon examined items. Therefore, our method according to this, offer an overview (glimpses), the ability to operate the content through various presentations (exploratory browsing). Detlor et al. [4] developed a model of information seeking that combines both browsing and searching. It suggests that much of Ellis's model [5] is already implemented by components currently available in Web browsers. We then applied this model to search presentations by considering user browsing behavior.

## III. SEMANTIC STRUCTURE ANALYSIS

### A. Element Extraction

There are two elements, i.e., textual elements and graphic elements, from presentation slides based on itemized sentences based on the XML files of slides. We define the slide title is the 1st level, the first item of text within the slide body is the 2nd level, and the depth of the sub-items increases with indentation level (3rd level, 4th level, etc.).

1) *Textual Elements*: We define textual elements as topics that focus on noun phrases in slides. Based on the presentation context, a topic that frequently appears at the higher levels (i.e., slide title) in neighboring slides. The topics that appear in the title of a slide and the body of other slides can be considered to indicate its context in a presentation. Then, we extract topics by locating the same noun phrases in different slides, at varied levels. If a noun phrase  $k$  appears at different levels in slides  $s_i$  and  $s_j$ , then  $k$  is a candidate for being one of the topics  $T$  in the presentation.

$$T = \{(k, s_i, s_j) | l_{max}(k, s_i) \neq l_{max}(k, s_j)\} \quad (1)$$

where,  $T$  is a bag of noun phrases that can be considered as candidates for topics.  $l_{max}(k, s_i)$  returns the highest level of  $k$  in  $s_i$ . For instance, when the highest level is the title, i.e., the 1st level, then  $l_{max}(k, s_i)$  is 1; and when the highest level is the 3rd level, then  $l_{max}(k, s_i)$  is 3. When  $k$  appears at different levels,  $k$  is determined as a candidate for topics by Eq. (1). Then, the weight of  $k$  in  $T$  is defined using the levels of  $k$ , and the distance between  $s_i$  and  $s_j$ , as follows:

$$I(k) = \frac{1}{l_{max}(k, s_i)} + \sum_{k, s_i, s_j \in T} \left( \frac{1}{l_{max}(k, s_j)} \cdot \frac{1}{dist(s_i, s_j)} \right) \quad (2)$$

where  $dist(s_i, s_j)$  corresponds to the strength of the association between  $s_i$  and  $s_j$ , and it denotes the distance between  $s_i$  and  $s_j$ . If  $k$  appears at a high level in  $s_i$  and  $s_j$ , and the distance between  $s_i$  and  $s_j$  is short,  $I(k)$  of  $k$  is high.

2) *Graphic Elements*: When compared to pure textual elements, images are more attractive, appealing and informative from a psychological standpoint. Therefore, we define graphic elements as images corresponding to the topic candidates in slides, given that the surrounding text of the images are similar to the topic candidates. We considered that the images used to describe the content in slides, and a slide title can be a subject of the content. When the similarity exceeds a predefined threshold by calculating the Simpson similarity coefficient, the images are recognized as the corresponding images of the topic candidates.

### B. Determination of Semantic Relationships

Semantic relationships between elements are determined from a document tree of a presentation to enable users obtain relevant information between the key elements. Preliminary ideas are given in an algebraic query model [6] as well.

1) *Basic Definitions and Algebra*: The presentation content shown in Figure 2 is represented as a rooted ordered tree  $D = (N, E)$  with a set of nodes  $N$  and a set of edges  $E \subseteq N \times N$ . There exists a distinguished root node from which the rest of the nodes can be reached by traversing the edges in  $E$ . Each node  $n$ , except the root, has a unique parent node, it of the document tree is associated with a logical component, i.e.,  $\langle title \rangle$  or  $\langle sections \rangle$ , based on an XML file in the given presentation. Function  $words(n)$



of  $x$  is included in the concept of  $y$ . When  $x$  and  $y$  fail to match these determinations of semantic relationships,  $x$  and  $y$  are independent. Therefore, a numbers of semantic relationships between  $x$  and  $y$  are formed from a set of fragments produced by taking the pairwise fragment join; a semantic relationship is determined by majority.

We conduct multiple presentations based on this semantic structure analysis, the semantic relationships follow a transitivity law, e.g., iff  $x$  shows  $y$  in presentation  $A$ ,  $y$  shows  $z$  in presentation  $B$ , then it is assumed that  $x$  shows  $z$ .

#### IV. IPOSTER GENERATION

We generate an iPoster possessing two features: (1) an overview of elements from the slides, retaining this feature of traditional posters; and (2) a ZUI, promoting user interaction and reflecting the semantics of the elements.

##### A. Determination of Element Layouts

When hierarchical relationships between two elements, reveal a hierarchy applied as to a hierarchical structure. *Show* or *describe* maps a parent-child relationship, if  $x$  shows  $y$  ( $y$  describes  $x$ ), then we mark  $x$  in a parent area and  $y$  in a child area, suggesting that the layer of  $x$  is higher than the layer of  $y$ . Moreover, *likewise* maps a sibling relationship, if  $x$  likewise  $y$ , then we locate  $x$  and  $y$  in the same layer. Inclusion relationships between two elements, reveal a logical relationship of inclusion and exclusion applied as to a stacked Venn. If  $x$  has-a  $y$  ( $y$  part-of  $x$ ), we conceive an area of  $y$  that is included in an area of  $x$ , and that the area of  $x$  is larger than the area of  $y$ .

##### B. Determination of Element Transitions

To utilize a ZUI, (1) users can browse the iPosters with their operations, such as zoom-in, zoom-out, and pan; (2) users can browse the iPosters without their operations by automatically navigations with transitions between elements. The transitions discussed here explain the kinds of visual effects that are applied to the semantic relationship types.

When *show* (*describe*) between two elements. Then, firstly the view must be zoomed-out from the focused element to an area of both, following which; it must be zoomed-in to the target element. Therefore, the transitions include passing through the area of both, which helps users to easily grasp the super-sub relation existing between them.

When *likewise* between two elements, the transitions between them include zooming-out from the focused element to an area enclosing both the elements and their parent element, and then zooming-in to the target element. Then, the transitions provide their parent element helps users to easily know they are subservient to the same concept.

When *has-a* (*part-of*) exists between two elements, the transition between the two elements pans from the focused element to the target element. Therefore, this simple and direct transition between the two elements helps users to

easily understand that they are dependent on each other, and that there exists an inclusion relationship between them.

The transitions between two independent elements include zooming-out from the focused element to all elements, and then zooming-in to the target element. These transitions help the user to easily know that they are irrelevant.

As depicted in Figure 3, we generated iPosters using actual Lecture #1<sup>4</sup> for Database at Portland State University by Prof. Laura Bright. We can easily find that this lecture emphasized the content of **Relational Database**.

#### V. EXPLORATORY SEARCH ON PRESENTATIONS

We build an exploratory search tool that aids users to search multiple presentations in search results by a given query: (1) focused searching and (2) exploratory browsing. Then, we measure dependence of the structure of the iPoster based on user browsing behavior, as follows:

$$D(H) = \frac{1}{|H| - 1} \sum_{n=1}^{|H|-1} \frac{1}{\text{dist}(e_n, e_{n+1})}, \quad e_n \in H \quad (4)$$

Here,  $H$  is a browsing history based on user browsing behavior.  $e_n$  is a browsed element in  $H$ . We define the browsed element, focusing on zoom-in operations of elements by the users, that the elements can be considered as the users are interested in. Then, we calculate a degree of  $D(H)$  by using average of relevance between the browsed elements. Function  $|H|$  returns the number of the browsed elements,  $|H| - 1$  then denotes the number of edges between them.  $\text{dist}(e_n, e_{n+1})$  is a shortest distance between  $e_n$  and  $e_{n+1}$  in an order, which is calculated by the number of edges between the browsed elements on the structure of the iPoster, then,  $\text{dist}(e_n, e_{n+1}) \geq 1$ . When  $\text{dist}(e_n, e_{n+1})$  returns 1, the relevance between  $e_n$  and  $e_{n+1}$  is closest. In this case, we set the threshold value  $\gamma$  at  $|H|/|E_p|$ ,  $|E_p|$  denotes the number of nodes in a partial tree included all browsed elements of the structure of the iPoster. If  $D(H) \geq \gamma$ , the browsing behavior can be considered well-matched on the structure of the iPoster; contrarily, if  $D(H) < \gamma$ , the browsing behavior can be considered partially-matched on the structure of the iPoster.

##### A. Focused Searching of Presentation Contents

When a user browses along the structure of the iPoster focused on a topic and its subtopics with zoom-in operations, we assume that it is focused searching based on well-matched browsing behavior, he wants to get details of the focused topics. Algorithm 1 describes a procedure for focused searching in a sub-structural layout  $FS = (E_d, R, P)$ .  $E_d$  is a set of elements related to  $x$  by users' operations as an input.  $r$  is a type of semantic relationships  $R$  defined in Table I.  $P$  is a set of presentations in search results by a given query. This procedure represents  $e'$  related to

<sup>4</sup>[http://web.cecs.pdx.edu/~howe/cs410/lectures/Relational\\_Intro\\_1.ppt](http://web.cecs.pdx.edu/~howe/cs410/lectures/Relational_Intro_1.ppt)

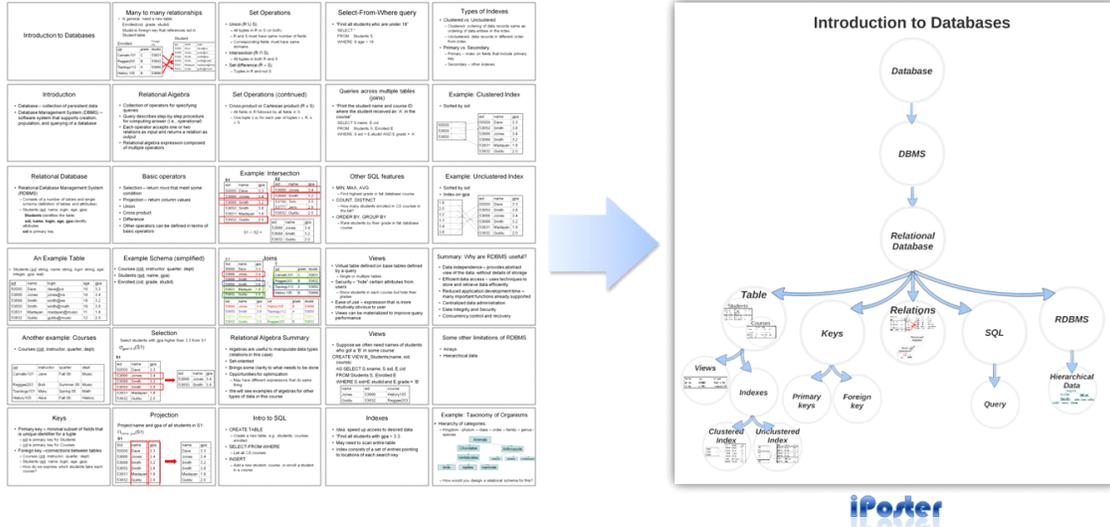


Figure 3. An Example of a generated iPoster based on our proposed method

---

**Algorithm 1** Explore  $FS = (E_d, R, P)$

---

**Require:**  $x$  is an element in a given presentation  $p$ , last browsed by a user with a zoom-in operation.

**Ensure:**  $R = \{(e, e', r) | e, e' \in E, e, e' \in p'\}$

$R \Leftarrow \phi$

**for** all presentation  $p'$  in a given domain **do**  
**if**  $r$  is *show* relationship **then**

$e \Leftarrow x$

$R \Leftarrow (x, e', r)$

$P \Leftarrow p'$

**end if**

**end for**

---

$x$  according to  $(x, e', r)$ , in which  $r$  is *show* for finding details of  $x$ .

**B. Exploratory Browsing of Presentation Contents**

When a user browses topics in apart on the iPoster, we assume that it is exploratory browsing based on partially-matched browsing behavior, he wants to get much relevant information of the browsed topics. Algorithm 2 describes a procedure for exploratory browsing in a sub-structural layout  $EB = (E_w, R, P)$ .  $E_w$  is a set of elements related to  $x$  and  $y$  by users' operations as an input. This procedure represents  $e'$  related to  $x$  and  $y$  according to  $(x, y, r)$ , in which  $r$  is *likewise* for finding relevant information of  $x$  and  $y$ .

Figure 4 illustrates an example of exploratory browsing, a user firstly zooms-in to the area of 'Forest Ecosystem,' after that zooms-out it and zooms-in to the area of 'Food Chain,' and zooms-out it and zooms-in to the area of 'Products.' We considered that he wants to get a lot of information about 'Food Chain' and 'Products' along the content related to

---

**Algorithm 2** Explore  $EB = (E_w, R, P)$

---

**Require:**  $x, y$  are elements in a given presentation  $p$ , browsed by a user with a zoom-in operation.

**Ensure:**  $R = \{(e, e', r) | e, e' \in E, e, e' \in p'\}$

$R \Leftarrow \phi$

**for**  $x, y$  such that  $(x, y, r) \in R$  in  $p$  **do**  
**for** all presentations  $p'$  in a given domain **do**  
**if**  $r$  is *likewise* relationship **then**

$e \Leftarrow x, y$

$r \Leftarrow describe$

$R \Leftarrow (x, e', r) = (y, e', r)$

$P \Leftarrow p'$

**if**  $r$  is *describe* relationship **then**

$e' \Leftarrow z$

$R \Leftarrow (e, z, r)$

$P \Leftarrow p'$

**end if**

**end if**

**end for**

**end for**

---

'Forests and Humans.' Due to 'Products' *likewise* 'Food Chain,' and they *describe* 'Forests and Humans,' 'Forests and Humans' has its details (i.e., 'Products' and 'Food Chain') only in  $P_A^5$ . In this work, we can extract 'Nitrogen Cycle' and 'Rainforest Animals', which *describe* 'Forests and Humans' in  $P_B^6$ , and represent a whole of 'Forests and Humans' with its details (i.e., 'Products,' 'Food Chain,' 'Nitrogen Cycle,' and 'Rainforest Animals'). In addition,

<sup>5</sup><http://teacherweb.com/AB/GilbertPatersonMiddleSchool/MsDavid/Tree-Types-2b-Posting-version.ppt>

<sup>6</sup><http://www.marinepolicy.net/cparsons/Ecology/12-Forests.PPT>

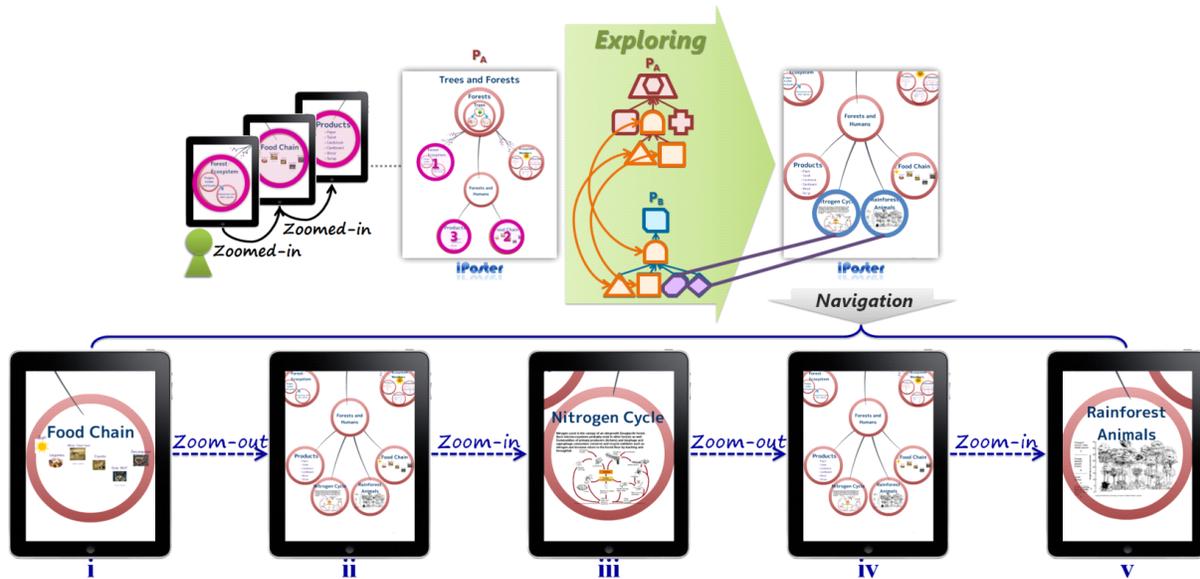


Figure 4. An example of exploratory browsing based on partially-matched browsing behavior

the iPoster can automatically navigate the whole of ‘Forests and Humans’ to ‘Nitrogen Cycle’ and ‘Rainforest Animals.’ The iPoster firstly zooms-out from the area of ‘Food Chain’ (i) to the whole area of ‘Forests and Humans’ (ii) that shows an overview of ‘Forests and Humans’ with ‘Nitrogen Cycle’ and ‘Rainforest Animals.’ Next, the iPoster zooms-in to ‘Nitrogen Cycle’ (iii) and ‘Rainforest Animals’ (v), respectively. It helps the user to understand details ‘Nitrogen Cycle’ and ‘Rainforest Animals’ of ‘Forests and Humans.’

In this way, when many elements are extracted, we need to consider how to select candidate elements from presentation contents such as weights of the elements.

## VI. CONCLUSIONS AND FUTURE WORK

In this paper, we built an exploratory search tool for presentation contents based on iPoster generation, which represents textual and graphic elements in a structural layout with ZUIs, to promote user interaction. In order to generate an iPoster, we introduced a semantic structure analysis model for extracting elements and determining semantic relationships between them from slides. The iPoster enables users to browse and explore easily and efficiently through various presentations.

In the future, we plan to consider a collaborative exploratory searching tool, which will provide a way to summarize already encountered information. The tool could tailor these summaries to the respective skill levels of collaborators.

## ACKNOWLEDGMENTS

This research was supported in part by Strategic Information and Communications R&D Promotion Programme

(SCOPE), the Ministry of Internal Affairs and Communications of Japan, and a Grant-in-Aid for Scientific Research (B)(2) 26280042 from the Ministry of Education, Culture, Sports, Science, and Technology of Japan.

## REFERENCES

- [1] J. Lanir, K. S. Booth, and S. Wolfman, “An Observational Study of Dual Display Usage in University Classroom Lectures,” in *Journal of Human-Computer Interaction*, vol. 28, no. 4, 2013, pp. 335–377.
- [2] R. Spicer, Y. Lin, A. Kelliher, and H. Sundaram, “NextSlidePlease: Authoring and Delivering Agile Multimedia Presentations,” in *ACM Transactions on Multimedia Computing, Communication and Applications*, vol. 8, no. 4, 2012, pp. 53:1–53:20.
- [3] R. W. White and R. Roth, “Exploratory Search: Beyond the Query Response Paradigm,” in *Synthesis Lectures on Information Concepts, Retrieval, and Services*, vol. 1, no. 1, 2009, pp. 1–98.
- [4] B. Detlor, C. W. Choo, M. MacKenzie, and D. Tunbull, “Information Seeking and Use in Diverse Organizational Contexts,” in *Proc. of the American Society for Information Science and Technology*, vol. 46, no. 1, 2009, pp. 1–4.
- [5] D. Ellis and M. Haugan, “Modelling the Information Seeking Patterns of Engineers and Research Scientists in an Industrial Environment,” in *Journal of Documentation*, vol. 53, no. 4, 1997, pp. 384–403.
- [6] S. Pradhan, “An algebraic query model for effective and efficient retrieval of xml fragments,” in *Proc. of the 32nd international conference on Very large data bases (VLDB 2006)*, 2006, pp. 295–306.

# From Intentions to Decisions: Understanding Stakeholders' Objectives in Software Product Line Configuration

Mahdi Noorian<sup>1</sup>, Ebrahim Bagheri<sup>2</sup>, Weichang Du<sup>1</sup>

University of New Brunswick, Fredericton, Canada<sup>1</sup>

Ryerson University, Toronto, Canada<sup>2</sup>

m.noorian@unb.ca, bagheri@ryerson.ca, wdu@unb.ca

**Abstract**—Software Product Line (SPL) engineering promotes the systematic and large-scale reuse of design and implementation artifacts. Feature models are one of the main artefact of SPLE approach which essentially characterize the similar and variant functional and operational specifications of the product family. Given the complexity of the variabilities represented by feature models, it is often hard for the stakeholders to analyze a feature model and identify the features that are most important for their purpose. So, given large-scale software product families, one of the important questions is how and what features should be selected for the target software product from the product family. To address this problem, we adopt concepts from the domain of goal-oriented requirement engineering and base feature selection decisions on software stakeholders' intentions and expectations. In this work, we propose a framework to automatically map stakeholders' objectives, which can be captured in the form of goal models, on feature models through the application of semantic analysis methods. Our proposed approach not only provides the means to systematically interrelate feature models and goal models but also helps software practitioners in moving from the stakeholders' goals and expectations towards domain model feature selection decisions in such a way that a more desirable final product for the stakeholders is developed.

## I. INTRODUCTION

A software product family provides the means for capturing the commonalities of all possible products of a given domain and also addresses variability by covering a comprehensive set of dissimilarities between the products [16]. There are basically two lifecycles in software product lines, namely domain engineering and application engineering. While, domain engineering is concerned with the process of understanding the target domain and developing a comprehensive formal representation of the concepts in that domain, application engineering takes the products of the domain engineering process and develops an appropriate application instance by carefully choosing and instantiating the right elements of the formal representation of the domain model. The process of selecting the right set of features and development of a new product from a software product line during application engineering is referred to as the configuration process [8].

Software product line configuration is an important step in application engineering, which can be seen as a constraint satisfaction process throughout which a product is being developed that needs to possess the highest number of desired features and properties and the least number of excessive and

undesirable ones from the perspective of stakeholders [4]. The software product line research community has developed effective methods for configuring product line models such as feature models. The basic assumption of these methods and tools is that the set of initial desirable features is already known to the stakeholders or at least to the software product designers [5]. However in reality regardless of the configuration process itself, the selection of the initial set of desirable features is both important and very difficult to do for the stakeholders and product designers. The selection of these features depends on the restrictions placed by and objectives of the stakeholders and the requirements of the target deployment environment. Therefore, an important research challenge is to develop methods or processes that can help identify the set of desirable features to fulfill the stakeholders' needs.

In reality, stakeholders are more comfortable in explaining their expectations from a software product in terms of their objectives, intentions, and goals rather than very formally explaining the functionality that they expect [1], [9]. Therefore, in the realm of software product line engineering where software products are defined by their features, the gap between the stakeholders' intention space and the product family feature space needs to be bridged. Kang [11] has mentioned that feature elements represent system functionality in a domain and the authors in [20] indicated that task elements in goal models represent the operation or function that can be defined for satisfying parent goals. Therefore, it is possible to integrate feature and goal models by identifying conceptually related pairs of task and feature elements.

In this work, we are interested in analyzing the early stages of the software product application engineering process, i.e., we investigate how the most suitable set of software product line features can be selected for the product configuration process by examining the stakeholders' needs and requirements gathered through a standard requirement engineering process. We propose a (semi) automated framework, which systematically maps the stakeholders' objective and goals (which is in the form of a goal model) to a domain feature model through shared domain Semantic Web ontologies. To this end, we first annotate feature and goal models' elements with ontological concepts through an annotation process and then integrate these two models by identifying and connecting conceptually related

elements e.g., pair of feature and task elements through a mapping process. Note that, the information in feature and goal models come from domain documents such as interview transcripts, functional requirements documents, strategic planning documents, among others [12]. In the feature and goal modeling process, domain analysts use these documents as a source to derive the elements to design the feature and goal models. Based on this assumption, we posit the existence of either explicit or implicit relation between the domain documents and the elements of feature and goal models. In order to have precise annotations, we analyze such domain documents and associate feature and goal model elements with their relevant supporting texts (the part of the domain documents based on which an element is contextualized). In the annotation process, we benefit from text analysis methods to extract important domain concepts from the associated texts and automatically annotate the feature and goal model elements with relevant ontological domain concepts. Afterwards, in the mapping process, we utilize an explicit semantic similarity function to measure the semantic relatedness of elements in the feature and goal models and connect the conceptually related elements. Finally, through the established mapping links and using diagrammatic reasoning techniques it is possible to trace between the intention and feature spaces and therefore find the most relevant features of the target software product family in the context of the stakeholders' goals and requirements.

## II. BACKGROUND

### A. Feature Models

Features are important distinguishing aspects or characteristics of a family of systems [16]. In a feature model, features are hierarchically organized by *Structural Constraints* which can be typically classified as: 1) *Mandatory*: a feature must be included in the product if the parent feature is included; 2) *Optional*: a feature may or may not be included in product if the parent feature is included; 3) *Alternative feature group*: one and only one of features from the feature group can be selected if the parent node is included; 4) *Or feature group*: one or more features from a feature group can be included in the product if their parent feature is included. In some case, the cross tree mutual interdependencies among the features exist; thus, additional constraints are often added to feature models and are referred to as *Integrity Constraints*. The two most widely used integrity constraints are: *Includes* - the presence of a given feature (set of features) requires the inclusion of another feature (set of features); and *Excludes* - the presence of a given feature (set of features) requires the elimination of another feature (set of features). Figure 1 depicts a small feature model for the tablet domain.

### B. Goal Models

Goal models have been introduced and widely used, as a controlled approach for organizing and structuring stakeholders' intentions in a graph-like representation [9]. In essence, goal models are fundamentally built over three important

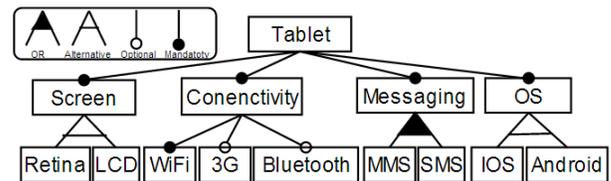


Fig. 1. A sample feature model representing a tablet domain.

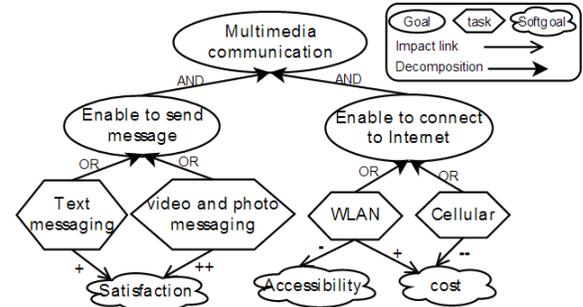


Fig. 2. A goal model representing stakeholders' expectations from the tablet domain.

concepts, namely goals, softgoals, and tasks. Goals are objectives related to the functional aspects of the system. In contrast, softgoals refer to non-functional or quality attributes of the system. Furthermore, tasks are ways to operationalize stakeholders' goals.

Goal models are often refined such that high-level goals are expressed through finer grained goals. This is achieved using decomposition links, i.e., each parent node is broken down into smaller child nodes whose disjunction or conjunction will satisfy the parent. In addition, since goals are operationalized through tasks, they are interrelated through contribution links. Tasks can also be refined using decomposition links. Moreover, impact links are used to show to what extent the developed tasks contribute to the satisfaction of a softgoal. Figure 2 shows a goal model representing some of the stakeholders' expectations from the tablet domain.

## III. A FRAMEWORK FOR FEATURE MODEL PRE-CONFIGURATION

The overview of our proposed feature model pre-configuration process is shown in Figure 3. Generally, the domain level processes are involved to create an overarching representation of the domain using feature models. Later, the application level process takes the developed formal models,

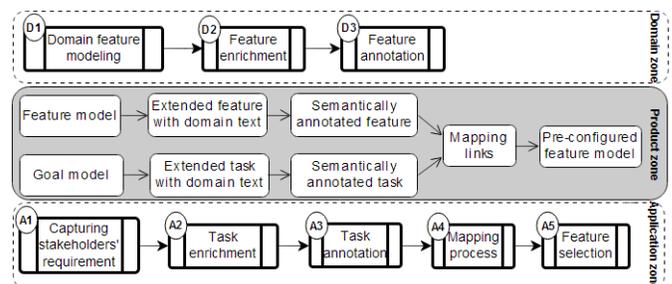


Fig. 3. A framework for feature model pre-configuration.

and also the needs, requirements and expectations of the stakeholders, which are represented in a goal model, into account in order to select a desirable set of features for a final product.

As seen, the domain level process consists of three steps:

- (D1) Using an available standard domain analysis process the set of all possible features and their dependencies (structural and integrity constraints) will be identified. As a result of this process, a formal representation of the features of the target domain are developed, which is referred to as the feature space.
- (D2) During the feature modeling process, domain documents are used as information sources for extracting feature model elements. Based on this, our second step will create the extended version of the feature model in which the elements in the model are linked with the sections of the domain documents from which the elements are derived from. In other words, each model element will be accompanied by textual snippets from domain documents that justify the existence of the model element.
- (D3) Once the feature elements are labeled with their related domain texts, each element will be semantically annotated with domain ontological concepts. We employ an automated semantic text analysis method in order to analyze the elements' associated texts and cross-reference the elements with a set of concepts that are identified from the texts.

The main focus of the devised processes in domain engineering is to develop a feature model whose elements are extended with semantic concepts. Furthermore, in application engineering the annotated feature model will be used as a basis of the mapping process. In the context of feature model configuration, the application engineering process is as follows:

- (A1) Communicates and understands the stakeholders' needs and requirements by identifying and modeling their intentions and goals through an available standard requirement engineering process (the intention space).
- (A2) During the goal modeling process, application engineers use domain related documents to derive goal, softgoal, and task elements. Similar to step (D2), in this step, the derived elements will be extended with their related supporting texts for more clarification.
- (A3) Similar to step D3, here we benefit from semantic text analysis methods in order to extract ontological concepts from the associated supporting text and annotate task elements in the goal model accordingly.
- (A4) In this step, the main concern is to identify the possible correspondences between the task elements of the intention space with feature elements of the feature space and link them through appropriate mapping links. In the D3 and A3 steps, both feature elements and task elements have already been semantically annotated with concepts from a shared domain ontology. Hence, through the mapping process the pair of semantically related task and feature elements can be identified. Here,

pairwise comparisons will be performed on the elements' representative concepts in order to measure the conceptual relatedness of each pair of elements. This process highlights a set of possible mappings between the models.

- (A5) Finally, diagrammatic reasoning techniques [10], [17] are employed to trace between the intention and feature spaces and therefore find the most relevant features of the target software product family in the context of the stakeholders' goals and requirements. These set of selected features can be used as appropriate input for feature model configuration techniques that given a feature model and a set of input features are able to create a fully configured software product model.

In the following sections, the details behind each of these steps are introduced.

#### A. Feature and Task Element Enrichment

In real world practice, domain engineers refer to a set of textual assets such as interview transcripts, functional requirements documents, strategic planning documents, as their information source in order to design and develop a feature and/or goal model. These documents are used as a source of information to extract the model elements like features and define relations among the derived elements [12].

During the design process, the rationale for extracting elements is documented by building traceability links between the derived elements and their supporting texts in the domain documents. These explicit links help to keep track of the reasons behind selecting the developed model elements, which augment the maintainability of the model. Specifically such augmented models will be more understandable and maintainable when they need to be updated with new identified requirements. Based on this, we consider that there is a traceability link between the identified elements, e.g. features or tasks, and the sources of information that the element is derived from.

In the current literature, the typical approach for mapping tasks and features has been to look at the syntactical similarity or synonymity between feature and task element names. Our approach takes this one step further by trying to find correspondences between feature and task elements by looking at their textual sources. In fact, we benefit from the traceability link between task/features and their textual sources, developed in the element enrichment step.

The result of the feature/task element enrichment is an extended version of feature and goal models in which each individual feature and task elements in the models is connected to its appropriate supporting text. For instance, the "IOS" feature can be extended with "IOS is an OS developed and distributed by Apple Inc." supporting text. In the following section, we will explain how the associated supporting text can be used in a semantic annotation process for identifying the relevant ontological concepts of each element.

#### B. Feature and Task Element Annotation

The main task in this step is to annotate feature and goal model elements with appropriate semantic concepts. For

this purpose, we use each element’s supporting texts that are added in feature/task element enrichment step (D2 and A2). We employ a text semantic analysis method to automatically extract semantic concepts from the texts and annotate the feature/task elements with ontological concepts.

Various types of semantic annotation systems exist, which can be used in our text semantic analysis process. The underlying techniques of such systems rely on a combined use of Natural Language Processing (NLP) and the large scale knowledge bases like DBpedia [6]. In our work, we benefit from Denote [7], a semantic annotation system, which employs each element’s supporting text as input. For a given element’s supporting text, the system automatically detects and identifies the relevant phrases (spots) that are available in the supporting text and relates each of them to appropriate ontological concepts. Accordingly, the element can be annotated with the identified semantic concepts.

### C. Mapping Process

The mapping process provides the opportunity to connect feature space with goal space through identifying a set of feature and task elements that are represented by related ontological concepts. The underlying challenge of mapping can be simply viewed as the problem of selecting the feature and the task that are annotated with a set of most similar concepts. The mapping process can automatically identify the mappings in two stages: first, it calculates the relatedness between all feasible combinations of tasks and features. Second, using Integer Linear Programming (ILP) [13], the best mappings are identified.

#### Stage 1: Calculating semantic relatedness

Lets assume that feature  $f \in F$  is annotated with a set of concepts  $A_f = \{\langle f, c_1 \rangle, \langle f, c_2 \rangle \dots \langle f, c_n \rangle\}$  and, task  $t \in T$  is annotated with  $A_t = \{\langle t, c'_1 \rangle, \langle t, c'_2 \rangle \dots \langle t, c'_n \rangle\}$ . In order to be able to calculate the semantic relatedness between two elements, we will need to measure and quantify the semantic relatedness between their concept sets.

$$Relatedness(f, t) = Relatedness(A_f, A_t). \quad (1)$$

Equation (2) calculates the semantic relatedness between two elements  $f$  and  $t$  based on their associated concept sets. The similarity between each pair of concepts  $c$  and  $c'$  is calculated through  $SIM(c, c')$  function.

$$Relatedness(A_f, A_t) = \sum_{i=1}^{|A_f|} (\max_{j=1}^{|A_t|} SIM(c_i, c'_j)). \quad (2)$$

As indicated by Strube et al. in [19], the measure for computing semantic similarity between two ontological concepts ( $c$  and  $c'$ ) can be categorized as: 1) path based measures, 2) information content based measures, and 3) text overlap based measures. To explain, path based measures compute the relatedness of two concepts by calculating the number of edges along the path between two concepts within the taxonomy hierarchy. Information content based measures compute the relatedness based on the extent in which two

concepts share information. Text overlap based measures calculate relatedness based on the overlap exist between the texts associated to each concept. Here,  $SIM$  function calculates and quantifies the conceptual similarity between two ontological concepts based on the combined measures that are introduced in [19]. Interested readers are encouraged to see this paper for details of the  $SIM$  function.

#### Stage 2: Find the mapping elements

We adopt the Integer Linear Programming (ILP) [13] method to find the most relevant feature element for each task element. We assume that, at most there is only one relevant feature for each task. This problem can be formulated as one variation of the classic assignment problem [13]. For a given feature set  $F = \{f_1, f_2 \dots f_n\}$  and task set  $T = \{t_1, t_2 \dots t_m\}$ , the mathematical model is as follows:

$$Maximize \sum_{i=1}^n \sum_{j=1}^m Relatedness(i, j) * x_{ij} \quad (3)$$

$$\sum_{i=1}^n x_{ij} \leq 1 \quad j = 1, 2 \dots m \quad (4)$$

$$\sum_{j=1}^m x_{ij} \leq 1 \quad i = 1, 2 \dots n \quad (5)$$

Equation (3) represents the objective function, which is the summation of  $x_{ij}$  as the decision variable where,  $x_{ij} = 1$  if feature  $i$  can be mapped on task  $j$  in the optimal solution and 0 otherwise. The  $Relatedness(i, j)$  represents the similarity degree of feature  $i$  in compare to task  $j$  which can be calculated based on Equation (1). Equation (4) shows the first set of constraints, which ensure that every feature is mapped on only one task and Equation (5) as a second set of constraints ensure that every task is mapped on a feature. As a result of the mapping process, each task  $t \in T$  will be mapped onto a semantically related feature element  $f \in F$ .

#### D. Feature Selection

In the mapping process step, the stakeholders’ objectives (intention space) is connected to the feature model (decision space) through a set of mapping links. In fact, considering the intention space can be helpful in selecting the most appropriate set of features for the specific purpose of the stakeholders. Here, the decision related to the selection of the best features can be translated into which one of the available features are related to the important goals of the stakeholders. Simply stated, the features that are connected through some path to the highly desirable goals of the stakeholders are considered to be more attractive to be included the final product.

This approach to feature selection can be formally defined through two types of diagrammatic reasoning approaches performed over goal models [15]. The first approach is called the backward label propagation algorithm proposed by Sebastiani et al [17]. This algorithm accepts as input the desired degrees of (dis)satisfaction of a set of high level goals, and propagates

these degrees throughout the goal model over the lower level goals and tasks. So, using the backward propagation algorithm, the stakeholders can select a set of high level goals as highly desirable, which will then be propagated through the goal diagram until the utility of all of the related lower level goals and tasks have been calculated. Now, since features are accessible for the goal model through the task-feature mappings, the desirability of each feature can also be calculated based on its relation to the goal model's goals and tasks. In this way, we are able to calculate the degree of utility and desirability of the feature model elements based on the stakeholders' intentions.

The second approach is referred to as the forward label propagation algorithm [10], which is introduced by Giorgini et al. In contrast to the latter approach, this algorithm starts from lower level goals and works its way to the top goals; therefore, it is able to estimate to what extent high level goals are satisfied given the satisfiability of the lower level goals and tasks. The interpretation of this algorithm for our context would be to select a specific feature and see how it relates to the stakeholder goals and to what extent it is able to satisfy the most important concerns of the stakeholders.

By benefiting from the backward and forward label propagation algorithms on goal models and the mapping between goal model and feature model elements, we are able to move from the intentions of the stakeholders to actual decisions with regards to the selection of the appropriate product features. Hence, we are able to find the features that significantly contribute to the satisfaction of stakeholders' intentions and objectives, and choose them as the most desirable ones.

#### IV. A DESCRIPTIVE CASE STUDY

Here, we intended to show how stakeholders' objectives can be translated into desirable software product line features using our proposed framework. We benefit from the tablet feature model (represented in Figure 1) and a related goal model (represented in Figure 2) as a case study. We first explain the processes in domain engineering and then discuss the processes that need to be conducted in application engineering.

**D1. Domain feature modeling:** In this step, using a standard domain analysis method a feature model is developed. Given we already have the tablet feature model, we skip this process.

**D2. Feature enrichment:** Here, the leaf features in the feature model will be extended with the related textual snippets which come from the sections of domain documents that the elements are derived from. The supporting texts that are selected for each element support and describe the context where the element is defined. In fact, during the feature model development these textual snippets should already be linked to each model element. Table I represents the supporting texts that are associated to each elements.

**D3. Feature annotation:** Once the feature elements are extended with their appropriate supporting texts, each feature element will be annotated with domain ontological concepts. We benefit from a semantic annotation system named Denote for extracting semantic concepts. Denote gets

TABLE I  
SAMPLE SUPPORTING TEXT FOR FEATURE ELEMENTS.

Feature Name	Supporting Text
Retina	Retina is liquid crystal display with high pixel density.
LCD	LCD is a flat panel display that uses the light modulating properties of liquid crystals.
WiFi	WiFi technology allows tablet connect to other devices through wireless communication.
3G	3G is mobile telecommunications technology, which provides Internet access via cellular network.
Bluetooth	Bluetooth is a wireless technology, which enables the device exchange file over short distances.
MMS	MMS is a standard way to send messages that include video and photo.
SMS	SMS is a messaging service component for sending text in mobile communication systems.
IOS	IOS is an OS developed and distributed by Apple Inc.
Android	Android is an OS that is developed by Google.

elements' associated supporting text as input and identifies the meaningful spots from the text. Furthermore, using DBpedia, each spot will be linked to a proper concept URI. Accordingly, each element will be cross-referenced with a set of concept URI that are identified from the supporting text. Table II represents the spots and the related concept URIs that are associated to "WiFi" and "3G" features. For other feature elements we can perform a similar process in order to extract concept URIs from their supporting texts. Due to space limitation, we only present the concepts extracted for "WiFi" and "3G" features.

TABLE II  
CONCEPT SET ASSOCIATED TO THE FEATURE ELEMENT.

Feature Name	Spot	Concept URI
WiFi	WiFi	<a href="http://dbpedia.org/page/Wi-Fi">http://dbpedia.org/page/Wi-Fi</a>
	Technology	<a href="http://dbpedia.org/page/Technology">http://dbpedia.org/page/Technology</a>
	Tablet	<a href="http://dbpedia.org/page/Tablet_computer">http://dbpedia.org/page/Tablet_computer</a>
	Device	<a href="http://dbpedia.org/page/Electronics">http://dbpedia.org/page/Electronics</a>
	Wireless	<a href="http://dbpedia.org/page/Wireless">http://dbpedia.org/page/Wireless</a>
3G	3G	<a href="http://dbpedia.org/page/3G">http://dbpedia.org/page/3G</a>
	Mobile	<a href="http://dbpedia.org/page/Mobile_telephony">http://dbpedia.org/page/Mobile_telephony</a>
	Internet	<a href="http://dbpedia.org/page/Internet">http://dbpedia.org/page/Internet</a>
	Access	<a href="http://dbpedia.org/page/Internet_access">http://dbpedia.org/page/Internet_access</a>
	Cellular Network	<a href="http://dbpedia.org/page/Cellular_network">http://dbpedia.org/page/Cellular_network</a>

Once the tablet feature model is annotated with ontological concepts, in the application engineering phase, this feature model can be used as a basis for the mapping and feature selection process. The application level processes can be conducted as follows:

**A1. Capturing stakeholders' requirement:** Here, we will benefit from the goal model that is represented in Figure 2. We assume that the stakeholders' objectives are captured and modeled based on a standard requirement engineering process.

**A2. Task enrichment:** Similar to step D2, in this step each task element in the goal model will be extended with supporting textual snippets. Table III represents the supporting textual snippets that are associated to the task elements.

TABLE III  
SAMPLE SUPPORTING TEXT FOR TASK ELEMENTS.

Task Name	Supporting Text
Text messaging	Text messaging is the act of sending electronic message between mobile phones.
Video and photo messaging	The video and photo content can be sent from mobile system.
WLAN	Using WLAN two or more tablet can be linked using wireless interfaces.
Cellular	Tablets can connect to cellular network in order to transmit voice and data.

TABLE IV  
CONCEPT SET ASSOCIATED TO THE TASK ELEMENT.

Task Name	Spot	Concept URI
WLAN	WLAN	<a href="http://dbpedia.org/page/Wireless_LAN">http://dbpedia.org/page/Wireless_LAN</a>
	Wireless	<a href="http://dbpedia.org/page/Wireless">http://dbpedia.org/page/Wireless</a>
Cellular	Tablet	<a href="http://dbpedia.org/page/Tablet_computer">http://dbpedia.org/page/Tablet_computer</a>
	Cellular	<a href="http://dbpedia.org/page/Cellular_network">http://dbpedia.org/page/Cellular_network</a>
	Voice	<a href="http://dbpedia.org/page/Data">http://dbpedia.org/page/Data</a>
	Data	<a href="http://dbpedia.org/page/Data">http://dbpedia.org/page/Data</a>

**A3. Task annotation:** Similar to step D3, here we benefit from Denote in order to identify the ontological concepts from the task elements' supporting text. As represented in Table IV, the "WLAN" and "Cellular" tasks are annotated with related concept URIs. Due to space limitation, we only present the identified concept URIs for "WLAN" and "cellular" task elements.

**A4. Mapping process:** The main concern in the mapping process is to connect the goal space to the feature space. To do this, the mapping process automatically identifies pairs of task and feature elements that are semantically related. The mapping process is performed in two stages: 1) calculating semantic relatedness and 2) finding the mapping elements. First, the semantic relatedness between all possible combination feature and task pairs are calculated. For each feature and task pair, semantic relatedness is computed. Table V shows the calculated semantic relatedness for each feature and task pair. For instance, the computed semantic relatedness score between "Text messaging" task ( $T_1$ ) and "Retina" feature ( $F_1$ ) is 48%.

TABLE V  
SIMILARITY MATRIX (%)

		F1	F2	F3	F4	F5	F6	F7	F8	F9
		Retina	LCD	WiFi	3G	Bluetooth	MMS	SMS	IOS	Android
T1	Text messaging	48	45	53	64	60	66	80	49	52
T2	Video and photo messaging	51	55	54	58	55	75	60	47	48
T3	WLAN	47	48	70	59	63	46	56	48	53
T4	Cellular	57	54	70	77	72	62	66	55	61

In the second stage, we benefit from ILP and calculate the mappings based on the available semantic relatedness scores. The integer linear programming formulation for this problem is represented in Table VI. The decision variables are developed according to combination of  $F_i$  and  $T_j$  letters, e.g.,  $F_1T_1$ ,  $F_1T_2$ , and etc. The objective function  $P$  in this problem is the summation of  $F_iT_j$  multiplied by the calculated semantic related score between  $F_i$  and  $T_j$ , e.g.,  $(0.48)F_1T_1$ . In addition, based on Equations 4 and 5, constraints are developed to ensure that a feature can only be mapped onto one task and also each task can only be mapped onto one feature.

TABLE VI  
ILP FORMULATION

Objective function	$P = (0.48)F_1T_1 + (0.51)F_1T_2 + (0.47)F_1T_3 + (0.57)F_1T_4 + (0.45)F_2T_1 + (0.55)F_2T_2 + (0.48)F_2T_3 + (0.54)F_2T_4 + (0.53)F_3T_1 + (0.54)F_3T_2 + (0.70)F_3T_3 + (0.70)F_3T_4 + (0.64)F_4T_1 + (0.58)F_4T_2 + (0.59)F_4T_3 + (0.77)F_4T_4 + (0.60)F_5T_1 + (0.55)F_5T_2 + (0.63)F_5T_3 + (0.72)F_5T_4 + (0.66)F_6T_1 + (0.75)F_6T_2 + (0.46)F_6T_3 + (0.62)F_6T_4 + (0.80)F_7T_1 + (0.60)F_7T_2 + (0.56)F_7T_3 + (0.66)F_7T_4 + (0.49)F_8T_1 + (0.47)F_8T_2 + (0.48)F_8T_3 + (0.55)F_8T_4 + (0.52)F_9T_1 + (0.48)F_9T_2 + (0.53)F_9T_3 + (0.61)F_9T_4$
Constraints	$F_1T_1 + F_1T_2 + F_1T_3 + F_1T_4 \leq 1$ , $F_2T_1 + F_2T_2 + F_2T_3 + F_2T_4 \leq 1$ , $F_3T_1 + F_3T_2 + F_3T_3 + F_3T_4 \leq 1$ , $F_4T_1 + F_4T_2 + F_4T_3 + F_4T_4 \leq 1$ , $F_5T_1 + F_5T_2 + F_5T_3 + F_5T_4 \leq 1$ , $F_6T_1 + F_6T_2 + F_6T_3 + F_6T_4 \leq 1$ , $F_7T_1 + F_7T_2 + F_7T_3 + F_7T_4 \leq 1$ , $F_8T_1 + F_8T_2 + F_8T_3 + F_8T_4 \leq 1$ , $F_9T_1 + F_9T_2 + F_9T_3 + F_9T_4 \leq 1$ , $F_1T_1 + F_2T_1 + F_3T_1 + F_4T_1 + F_5T_1 + F_6T_1 + F_7T_1 + F_8T_1 + F_9T_1 \leq 1$ , $F_1T_2 + F_2T_2 + F_3T_2 + F_4T_2 + F_5T_2 + F_6T_2 + F_7T_2 + F_8T_2 + F_9T_2 \leq 1$ , $F_1T_3 + F_2T_3 + F_3T_3 + F_4T_3 + F_5T_3 + F_6T_3 + F_7T_3 + F_8T_3 + F_9T_3 \leq 1$ , $F_1T_4 + F_2T_4 + F_3T_4 + F_4T_4 + F_5T_4 + F_6T_4 + F_7T_4 + F_8T_4 + F_9T_4 \leq 1$

The result for each decision variable  $F_iT_j$  would be 1 or 0 ( the value 1 indicates the  $F_i$  potentially can be mapped on  $T_j$  and value 0 indicates otherwise). For better representation, we show the obtained results in Table VII. The cells with value 1 (gray cells) indicate the optimal mapping between the corresponding feature  $F_i$  and task  $T_j$  elements. For instance, the value for  $F_1T_1$  is 0 which indicate that the corresponding

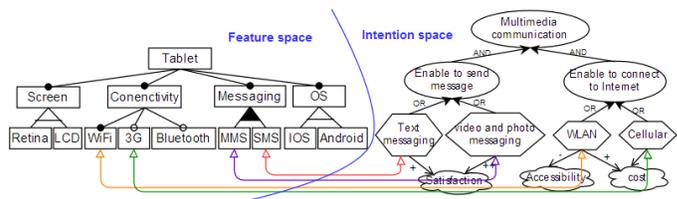


Fig. 4. Mapping the goal and feature spaces in the tablet product line.

"Text messaging" task ( $T_1$ ) cannot be mapped onto "Retina" feature ( $F_1$ ). While the value assigned to  $F_4T_4$  variable is 1, which indicates that the "Cellular" task ( $T_4$ ) can be mapped onto the "3G" feature ( $F_4$ ).

TABLE VII  
MAPPING BETWEEN TASK AND FEATURE ELEMENTS.

		F1	F2	F3	F4	F5	F6	F7	F8	F9
		Retina	LCD	WiFi	3G	Bluetooth	MMS	SMS	IOS	Android
T1	Text messaging	0	0	0	0	0	0	1	0	0
T2	Video and photo messaging	0	0	0	0	0	1	0	0	0
T3	WLAN	0	0	1	0	0	0	0	0	0
T4	Cellular	0	0	0	1	0	0	0	0	0

**A5. Feature selection:** In order to identify the features that are relevant to the stakeholders' goals, the links between the goal space and feature space can be employed. These links allow us to trace from goals and their operational tasks through to actually implementable features. Figure 4 shows how goals can be decision rationale for choosing features. As an example, "enable to connect to Internet" goal constitutes the "WLAN" and "Cellular" tasks. If we follow the linkage from this goal and its corresponding task to the feature space, we will find the matching features, which are the "WiFi" and "3G" features. Therefore, the justification can be made with regards to the decision to choose the "WiFi" or "3G" feature, which is because they are required to implement the "WLAN" and "Cellular" tasks that are needed to satisfy the "enable to connect to Internet" goal of the stakeholders. The process described here simply starts from stakeholders' high level goals and works its way through the defined tasks and finds the corresponding features from the feature space; therefore, this approach is a form of backward label propagation algorithm.

## V. RELATED WORK

Due to the importance of formally capturing and modeling requirements in SPL lifecycles, the idea of employing goal-oriented approaches has gained more attention in the product line research community. Therefore, some authors have tried to benefit from goal models as a early requirement engineering products and use them as a foundation to develop a variability model like the feature model [2], [18], [21], [22].

In [22], the authors describe how one can gradually enrich a standard goal model with a set of notations and then using the provided heuristic rules derive high variability design models such as feature models, statecharts, and component models. Furthermore, the authors in [21] present a tool that benefits from the transformation rules described in [22] in order derive feature models from goal models. Also, in [2], Antonio et al., propose an approach for constructing feature models using the goal model represented in  $i^*$  goal models. They define a set

of high level processes for transforming  $i^*$  elements and their relationships to a feature model. Siva et al [18] also followed a similar approach to Antonio et al in which the  $i^*$  model is extended with cardinality information and through a set of heuristics rules the extended  $i^*$  model is transformed to a feature model.

Moreover, there has been interest in trying to employ goal models in software system customization and configuration process [14], [15], [23]. In [15], Liaskos et al. take into account the user goals in the configuration process. They, first develop a goal model on top of the software configuration options and use them as a mediator between user and configuration options. Then, through the qualitative reasoning on the goal model, the appropriate configuration options can be selected that satisfy the user's high level goals. Similar to the work presented in [15], Yu et al. in [23] propose a requirement-driven configuration process which consist of two steps: 1) establish a goal model for software system and connect it to system's configuration items; and 2) configure a personalized software by collecting user's goal and expectations. Furthermore, in [14] Lapouchnian et al. show how goal models can be used as a basis for designing automatic software, which are able to self-configure based on the changing operating environment.

In the light of the aforementioned significant achievements, we propose a semi-automated framework in the context of software product line for mapping goal models onto a domain feature model. In our problem, we consider the stakeholders' intentions and goals as a key factors in the feature selection process in order to ensure that 1) a complete and comprehensive set of initial features from the set of available features is selected (that can be passed into automated feature model configuration processes), which is due to the fact that we can make sure that all stakeholders' intentions, objectives and concerns are covered and addressed by the selected features; 2) irrelevant superfluous features are not included in the selected features, since features that do not correspond with at least one of the stakeholders' goals will not be considered; and 3) the rationale behind the feature selection process is clear for the stakeholders.

## VI. CONCLUDING REMARKS

In this work, we promote the consideration of early requirements information in the form of stakeholders' goals and objectives in the process of selecting the right features for a particular software product. The introduced approach can serve as a best practice guideline that provides a convenient way for capturing stakeholders' intentions, mapping them as concrete requirements into software features of the whole product line family, and feeding the process of the software product line configuration with a more accurate set of stakeholders' desired features. To support our approach, we have proposed an automated framework to help application engineers identify the most suitable set of features of a software product feature model to be included in the final software product. We employ semantic analysis methods in order to map stakeholders' goals onto feature models and then by diagrammatic reasoning tech-

niques one can trace between the intention and feature spaces for making actual product functionality selection decisions. For future work, we are interested in enhancing our framework by considering softgoals as the decision criteria in the feature selection process.

## REFERENCES

- [1] A.I. Anton. Goal-based requirements analysis. In *Requirements Engineering, 1996., Proceedings of the Second International Conference on*, pages 136–144. IEEE, 1996.
- [2] S. António, J. Araújo, and C. Silva. Adapting the  $i^*$  framework for software product lines. In *Advances in Conceptual Modeling-Challenging Perspectives*, pages 286–295. Springer, 2009.
- [3] E. Bagheri, M. Asadi, D. Gasevic, and S. Soltani. Stratified analytic hierarchy process: Prioritization and selection of software features. In *Software Product Lines: Going Beyond*, pages 300–315. Springer, 2010.
- [4] D. Batory. Feature models, grammars, and propositional formulas. *Software Product Lines*, pages 7–20, 2005.
- [5] D. Batory, D. Benavides, and A. Ruiz-Cortes. Automated analysis of feature models: challenges ahead. *Communications of the ACM*, 49(12):45–47, 2006.
- [6] M. Cornolti, P. Ferragina, and M. Ciaramita. A framework for benchmarking entity-annotation systems. In *Proceedings of the 22nd international conference on World Wide Web*, pages 249–260, 2013.
- [7] J. Cuzzola, Z. Jeremic, E. Bagheri, D. Gasevic, J. Jovanovic, and R. Bashash. Semantic tagging with linked open data. In *Canadian Semantic Web Symposium*, 2013.
- [8] K. Czarniecki, S. Helsen, and U. Eisencker. Staged configuration through specialization and multilevel configuration of feature models. *Software Process: Improvement and Practice*, 10(2):143–169, 2005.
- [9] SK Eric and J. Mylopoulos. From er to armodelling strategic actor relationships for business process reengineering. In *Entity-Relationship Approach ER'94 Business Modelling and Re-Engineering*, pages 548–565. Springer, 1994.
- [10] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani. Reasoning with goal models. *Conceptual Modeling?ER 2002*, pages 167–181, 2003.
- [11] 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, 1990.
- [12] 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*, 5(1):143–168, 1998.
- [13] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.
- [14] A. Lapouchnian, Y. Yu, S. Liaskos, and J. Mylopoulos. Requirements-driven design of autonomic application software. In *Proceedings of the 2006 conference of the Center for Advanced Studies on Collaborative research*, page 7. IBM Corp., 2006.
- [15] S. Liaskos, A. Lapouchnian, Y. Wang, Y. Yu, and S. Easterbrook. Configuring common personal software: a requirements-driven approach. In *Requirements Engineering, 2005*, pages 9–18, 2005.
- [16] K. Pohl, G. Böckle, and F. Linden. *Software product line engineering: Foundations, principles, and techniques*. 2005.
- [17] R. Sebastiani, P. Giorgini, and J. Mylopoulos. Simple and minimum-cost satisfiability for goal models. In *Advanced Information Systems Engineering*, pages 675–693. Springer, 2004.
- [18] C. TLL Silva, C. Borba, and J. Castro. A goal oriented approach to identify and configure feature models for software product lines. In *WER*, 2011.
- [19] M. Strube and S. P. Ponzetto. Wikirelate! computing semantic relatedness using wikipedia. In *AAAI*, volume 6, pages 1419–1424, 2006.
- [20] E.S.K. Yu. Towards modelling and reasoning support for early-phase requirements engineering. In *Requirements Engineering*, pages 226–235, 1997.
- [21] Y. Yu, J.C.S. do Prado Leite, A. Lapouchnian, and J. Mylopoulos. Configuring features with stakeholder goals. In *Proceedings of the 2008 ACM symposium on Applied computing*, pages 645–649. ACM, 2008.
- [22] Y. Yu, A. Lapouchnian, S. Liaskos, J. Mylopoulos, and J. Leite. From goals to high-variability software design. In *Foundations of Intelligent Systems*, pages 1–16. Springer, 2008.
- [23] Y. Yu, A. Liaskos, S. Lapouchnian, and J. Mylopoulos. Requirements-driven configuration of software systems.

# Towards the Establishment of a Software Product Line for Mobile Learning Applications

Venilton Falvo Júnior  
ICMC/USP, Brazil  
venilton@icmc.usp.br

Nemésio F. Duarte Filho  
ICMC/USP, Brazil  
nemesio@icmc.usp.br

Edson Oliveira Jr  
UEM/PR, Brazil  
edson@din.uem.br

Ellen Francine Barbosa  
ICMC/USP, Brazil  
francine@icmc.usp.br

**Abstract**—The enormous popularity of mobile devices in the society has motivated the development of mobile learning applications. In spite of the benefits with regard to teaching and training, the existing learning applications still have to address issues and challenges related to the development, reuse and architectural standardization. On the other hand, researches have been carried out to employ the Software Product Lines (SPL) approach through the development of mobile learning applications. A SPL focuses on software reuse and has been successfully applied for specific domains. In this context, this paper proposes M-SPLearning, one SPL particularly established to the mobile learning applications domain. The main goal of M-SPLearning is to provide benefits with regard to the overall quality, domain comprehension, and reduction of the time spent in the development and maintenance of m-learning applications.

**Keywords**- *m-learning; software product line; mobile learning applications; proactive adoption model*

## INTRODUCTION

In recent years, learning environments have shown an increasing importance, playing a fundamental role in teaching and training activities, both in academic and industrial settings. Together with the advent of ubiquitous and mobile computing, learning environments have also contributed for a new modality of education – the m-learning (mobile learning) [1, 2]. As an attempt to analyze the main motivations for m-learning, O’Malley et al. [3] emphasize the impact of technological advances, such as intelligent interfaces, contextual modeling applications and the recent progresses in the wireless communications area, which altogether have provided several new and innovative perspectives for technology users.

M-learning introduces flexibility to the learning process, since the creation, exchange and access to information occur naturally due to the ubiquity of mobile devices. In this sense, apprentices are able to decide when, how and where they feel more comfortable to learn [3]. On the other hand, one of the main problems of m-learning is the lack of a well-defined standard with respect to the means of information access. Due to the large number of mobile devices available in the market, the production of content for these devices becomes strongly dependent of issues such as manufacturer and operating system, for instance [2].

In a different but related perspective, the introduction of object-oriented (OO) concepts, component-based development and service-oriented development have attracted the interest of the software community to the opportunities and benefits of code reuse. The success of such initiatives have stimulated the

reuse in several stages of the software development process, including in artifacts such as documents, specifications and models, further increasing the perspective of cost reduction and return on investment (ROI) [4].

The evolution of these ideas has led to the concept of Software Product Line (SPL), which represents a paradigm change in the regard of the traditional software development. Instead of developing software “project-to-project”, organizations should now focus their efforts on creating and maintaining a core asset, which would be the basis for the construction of specific products for a given domain [5].

Motivated by this scenario, in this paper we propose a SPL for the development of m-learning applications, named M-SPLearning. The main goal is to investigate the benefits of systematic reuse of an SPL in the context of m-learning. At the very end, the idea is to promote the overall quality, domain comprehension, and reduction of the time spent in the development/maintenance of m-learning applications.

This paper is organized as follows. In Section II, the background for our work is summarized. In Section III, we describe the main aspects of M-SPLearning through the process used for its creation. In Section IV, we discuss a preliminary evaluation of the proposed SPL. In Section V, we briefly discuss the threats to validity. Finally, in Section VI, we summarize our conclusions and perspectives for future work.

## BACKGROUND

SPL represents a new paradigm in Software Engineering, providing expressive results in terms of cost, schedule and quality [6]. Linden et al. [5] describe a SPL as a set of software systems sharing common features that satisfy a specific need for a particular market segment, developed from core assets in a systematic way, usually formed by a software architecture and its components.

The concept of SPL is suitable to domains in which there is a demand for products that have common features but which also contain a well-defined set of variabilities. Despite its relevance, the required activities to adopt the SPL concept are not trivial, demanding considerable time and effort. Krueger [7, 8] presents three different adoption models to support the establishment of a SPL: (i) proactive; (ii) reactive; and (iii) extractive. The proactive model is fully supported by the scope of the required systems, being appropriate when the requirements for the set of products to be developed are stable and can be previously defined. Therefore, each activity has to be finished before the next one be started.

In the reactive model, the SPL incrementally evolves whenever there is a demand for new products or new requirements are specified for the existing products. This approach is suitable when it is not possible to predict the requirements for each specific product.

The extractive model reuses one or more existing software to the initial base of the SPL. To be an effective choice, this model should not require complex technologies for the development of the SPL and must allow the reuse of existing software without the need of a high level of reengineering.

Other relevant concept refers to m-learning. This new type of electronic learning takes place when the interaction among the actors of the learning process is performed through mobile devices (tablets, smartphones, PDAs, etc). As a new and emerging paradigm, there are several attempts for defining it. According to O'Malley et al. [3], m-learning refers to any kind of learning that occurs when the apprentice is not in a fixed place, or when one takes advantage of learning opportunities provided by mobile devices, thereby relating technological and mobility concepts. Ozdamli and Cavus [9] describe m-learning as an activity that allows individuals to be more productive when they consume, create or interact with information, supported by mobile devices.

No matter the definition adopted, the interaction with learning applications through mobile devices provides benefits that go beyond accessibility, convenience and communication [10]. However, in spite of the advantages offered and even with the increasing demand for m-learning applications, there are few works addressing development issues in this new learning setting. One of the researching initiatives in terms of the development of m-learning applications can be found in Duarte Filho and Barbosa's work [11]. In short, the authors proposed a requirements catalog for m-learning environments. The catalog was established from the results of a systematic review conducted in this domain. To facilitate the understanding and the maintenance of the catalog, a three-level hierarchical structure (criteria, requirements and description) was adopted. Additionally, based on the knowledge of domain specialists, the requirements were prioritized in order to reflect the main experiences and needs in the m-learning setting.

Duarte Filho and Barbosa [11] also suggest that the requirements defined in the catalog may serve as a basis for: (1) the specification of a quality model for m-learning environments; and (2) the establishment of a reference architecture for m-learning environments. Actually, quality standards can support the definition of requirements, thereby contributing to the establishment of a high quality architecture.

The SPL architecture (PLA) plays a central role to successfully generate specific products taking into account the development and evolution of a SPL. It represents, in an abstract level, the architecture of all potential products for a specific domain. The PLA addresses the SPL design decisions by means of similarities and variabilities [12]. Thus, the PLA evaluation can be seen as one of the most important activities throughout a SPL life cycle [5]. In this sense, the requirements catalog proposed by Duarte Filho and Barbosa [11] and the adoption models proposed by Krueger [7, 8] have been investigated as the basis for the M-SPLearning construction.

Before starting to develop M-SPLearning, we looked for some related works dealing with the construction of SPLs (Table I). Although considering different domains, all the works analyzed provide relevant information regarding the techniques and approaches used in the construction and adoption of SPLs. For the sake of space, these works will not be detailed herein.

TABLE I. SUMMARY OF THIS AND RELATED WORK

Work	Domain	Adoption Model	Architectural Base
M-SPLearning	M-learning	Proactive	Component
Dalmon et al. [13]	Interactive modules	Extractive (5 applications)	Component
Marinho et al. [14, 15]	M-guides	Extractive (57 applications)	SOA
Pascual et al. [16]	Pervasive systems	Proactive	Component and Aspect

Linden et al. [5] claim that one of the most critical activities in creating a SPL refers to the scoping of the target domain. To define the initial scope of M-SPLearning, we mainly considered the requirements catalog proposed by Duarte Filho and Barbosa [11]. According to the authors, the catalog intends to reflect, in a high level basis, the experience gained from developers and researchers in this new modality of learning. Furthermore, the catalog is generic and embracing, benefiting its adoption for different purposes in the m-learning domain.

Still with regard to scoping issues, since mobile applications can be built using a native development approach, the m-learning domain encompasses several different operating systems. Thus, to define an acceptable domain in terms of scope, we had to select a single operating system. Our choice, Android OS, was based on the amount of devices that each operating system controls. According to Llamas et al. [17], Android OS owns 68.8% of the world's smartphones market, which represents nearly 500 million devices.

Having delimited the scope, the adoption models proposed by Krueger [7, 8] were analyzed to identify the most appropriate to the construction of M-SPLearning. The specificities and features of the mobile domain, along with the existence of a requirements catalog for m-learning environments [11], have motivated the choice of the proactive model. Such a proactive approach comprises four main phases, described next.

#### A. Domain Analysis

According to Krueger [7, 8], at this phase the domain is analyzed in order to identify the variation in the specific products from a SPL. In this sense, requirements catalog proposed by Duarte Filho and Barbosa [11] was analyzed with respect to the ISO/IEC 25010 – International Standard for Software Product Quality (successor of the ISO/IEC 9126 standard) [18]. The aim was to identify missing or disconnected requirements.

From the analysis conducted, we noticed that most of the requirements were equivalent to the features/sub-features established by the ISO/IEC 25010 standard. However, some requirements had to be rearranged and/or renamed and, in a few cases, added and/or removed.

The following step consisted of identifying the variabilities incorporated by the specific products of the SPL. Variability is one of the most important issues in designing a SPL, reflecting the diversity and commonality of its artifacts [19]. Therefore, the precise and explicit representation of variabilities makes it possible the generation of specific products in a SPL.

Variabilities may be identified and represented by the concept of features [20]. A feature is defined as a system characteristic that is relevant and visible to the end user [21]. Features are usually represented by a feature model, i.e., a hierarchical representation that captures the structural relationships among the features of a specific domain.

Feature models are visually represented by means of feature diagrams. Figure 1 illustrates the feature diagram representing the requirements catalog for M-SPLearning. The interpretation of the feature diagram is straightforward and its construction is in agreement with the concepts and guidelines proposed by Kang et al. [21]. Shortly, each requirement in the catalog was evaluated considering its relevance in the SPL domain and, if appropriate, mapped as feature. The primary features are summarized as follows:

- *Pedagogical*: incorporates educational and pedagogical requirements in order to facilitate and support teaching and training activities. This feature and its sub-features are represented as mandatory due to their relevance in the mobile learning domain. The only optional sub-feature is *interactivity*, which allows communication with social networks;
- *Usability*: addresses relevant issues with respect to the visual interface of a product, being crucial to the software market acceptance. This feature assures that all products generated by the SPL follow an usability standard, adding quality to the final product. Since usability is a global feature of the product and not a specific feature, it is defined as mandatory and abstract.
- *Compatibility*: encompasses coexistence and the ability of a product to exchange information with other systems in the same operating environment. As the feature and its sub-features are crucial for mobile applications, they are defined as mandatory (and the *coexistence* feature is abstract);
- *Security*: fundamental feature for any educational application. Due to the relevance of the features regarding *integrity* and *confidentiality*, they are defined as mandatory. The *authentication* sub-feature is optional;
- *Communication*: supports the exchange of information among users enabling, for instance, message exchange, delivery of test results and even synchronization of activities performed in other mobile devices. It is an optional feature; and
- *Support*: provides some supporting alternatives to the user such as help and internationalization. It is classified as optional, as well as its sub-features.

### B. Validation of the Domain Analysis

Aiming at validating the requirements previously elicited, this phase suggests the application of a checklist to evaluate the main technical issues related to M-SPLearning. An online checklist, composed of twelve multiple-choice questions, was prepared and applied to domain experts. The first two questions

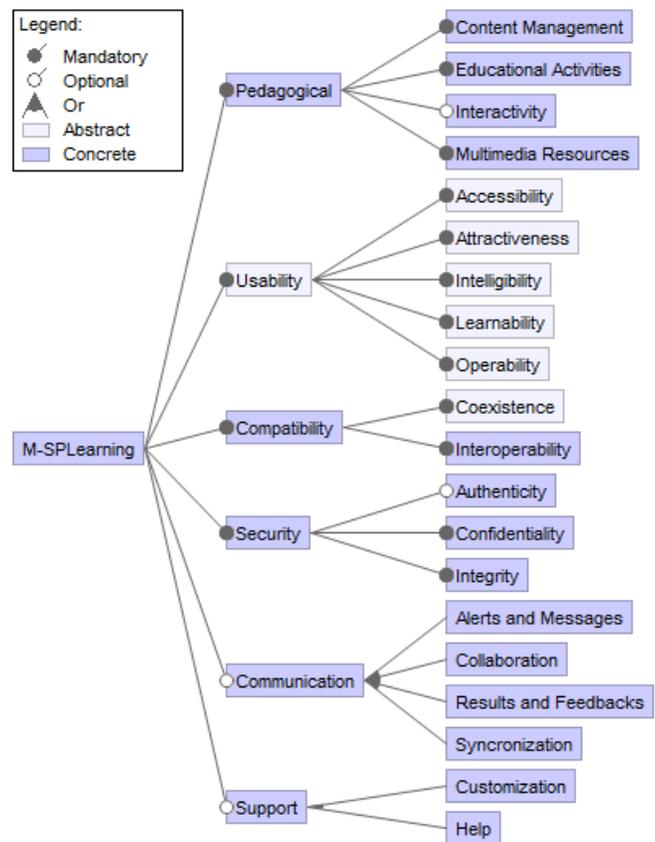


Figure 1. The M-SPLearning Feature Diagram

were related to the participants' experience in SPL and m-learning. The remaining questions were related to the requirements catalog resultant from the domain analysis.

From the obtained results, taking into account the participants' point of view, the requirements catalog was considered adequate (60.90%) or at least regular (39.10%) for all items evaluated. None of the questions related to the catalog was answered as unsatisfactory.

Despite the positive results achieved, it is important to highlight that the requirements validation conducted is still preliminary. In this scenario, an empirical validation through experiments with qualified practitioners from industry and academia would be extremely relevant. These experiments have been planned and should be performed in short term.

### C. Architecture Definition

The third phase involves the definition of a PLA for M-SPLearning. Ultimately, such architecture can serve as a basis for the derivation of all products defined in the scope of the SPL proposed.

In short, we defined a component-based architecture aiming at bringing together the benefits of reuse and modularization to the systematic characteristics of SPLs. To do so, we take into account SMarty [22], a variability management approach for UML-based SPL. More specifically, we applied the UML profile that the approach provides, the SmartProfile, according to a set of guidelines from the SMartyProcess.

According to Oliveira Jr et al. [22], the SMartyProfile contains a set of stereotypes and tagged values to represent variability in SPL models. This profile uses a standard object-oriented notation and its profiling mechanism in order to provide an extension of UML and to allow graphical representation of variability concepts. Figure 2 illustrates the M-SPLearning architecture according to SMarty.

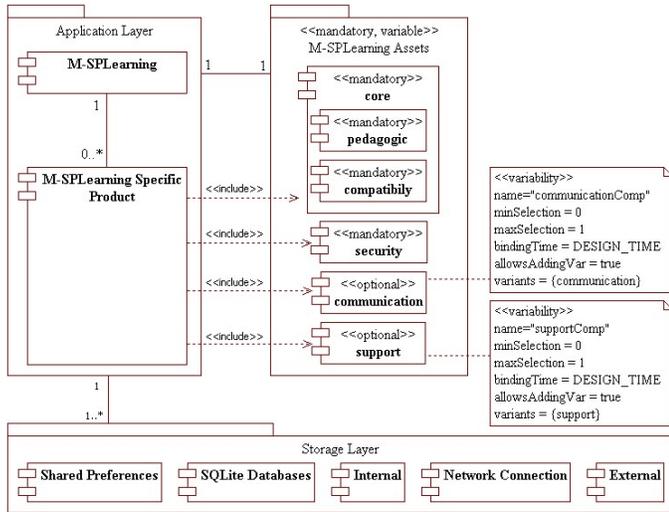


Figure 2. The M-SPLearning Architecture

From the architectural diagram, it is possible to identify the basic structure used in the construction of M-SPLearning. Notice that the assets package contains the components that correspond to the requirements of M-SPLearning. These components provide the concrete features, defined in the feature diagram (Figure 1).

The components that represent the essential features were grouped as the core component. Thus, the idea is to unify all the fundamental modules to any product generated by M-SPLearning. Furthermore, the other components have a dependence relationship with the core, since they need the core to provide their functionalities.

The application layer contains the component that represents M-SPLearning, showing their association with the assets package and making explicit the possibility of creating multiple products. Each generated product uses the components available in the M-SPLearning assets, thereby allowing that different configurations can be used according to the settings defined in application layer.

The products communicate with the storage layer to perform the operations provided by SPL that require data access or data writing. Since the products generated can be manipulated by developers, they can also communicate with the storage layer in an external mode. This alternative can be useful, for instance, if it is necessary to implement a variability that is not supported by M-SPLearning.

Next we summarize the design of the components that implement the concrete features of M-SPLearning. These components define the similarities and variabilities supported by M-SPLearning.

#### D. Components Design

The design phase completes the process of creating a SPL. According to Krueger [7, 8], variabilities and similarities among specific products of a SPL must be identified. Therefore, the features of the components stored in the repository should be designed and visually represented through a new component diagram, as shown in Figure 3.

From the component diagram, it is possible to explore all the details of the concrete features covered by the SPL, detailing the variation points and identifying each component with its respective SMartyProfile stereotype. These stereotypes make it easier the understanding of the possible configurations of the m-learning applications resulting from M-SPLearning.

The dependence relationship among components becomes explicit as well. As previously said, the core component fully unifies the mandatory features. This is particularly necessary for *security*, *communication* and *support* components.

Notice that the *communication* component is not directly dependent to the *core*. Actually, this dependence is intermediated by the *security* component, guaranteeing that all products have a secure communication. This ensures the architectural obligation that all components depend on the *core* component, which is responsible for providing the fundamental features of M-SPLearning.

#### M-SPLEARNING: IMPLEMENTATION

In order to preliminary evaluate the application of M-SPLearning, some of its features were implemented according to the SOA architectural pattern, which is frequently associated with the concept of SPL [14, 15, 23].

Basically, M-SPLearning presents a logical view of a SPL defined for the configuration and creation of Android applications. These applications were generated through the consumption of services, and mainly by the collaboration among the implemented modules. Three modules are particularly important, since their interactions and responsibilities characterize our SPL. Figure 4 highlights the adopted architecture, relating it to the initial architecture proposed by M-SPLearning.

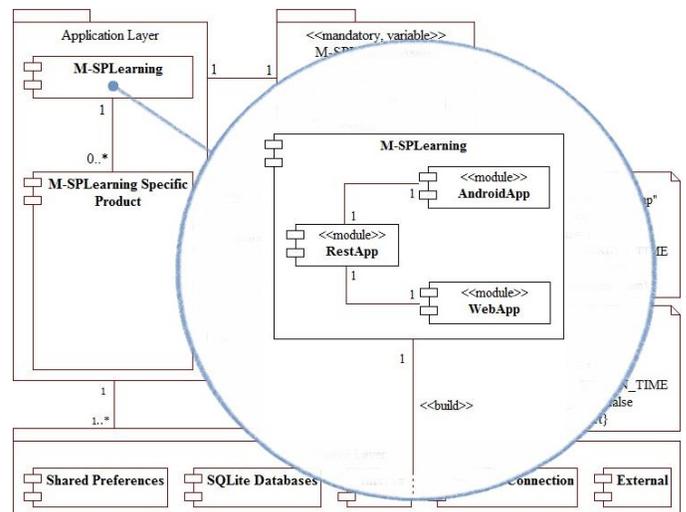


Figure 4. Main modules implemented in the architecture of M-SPLearning

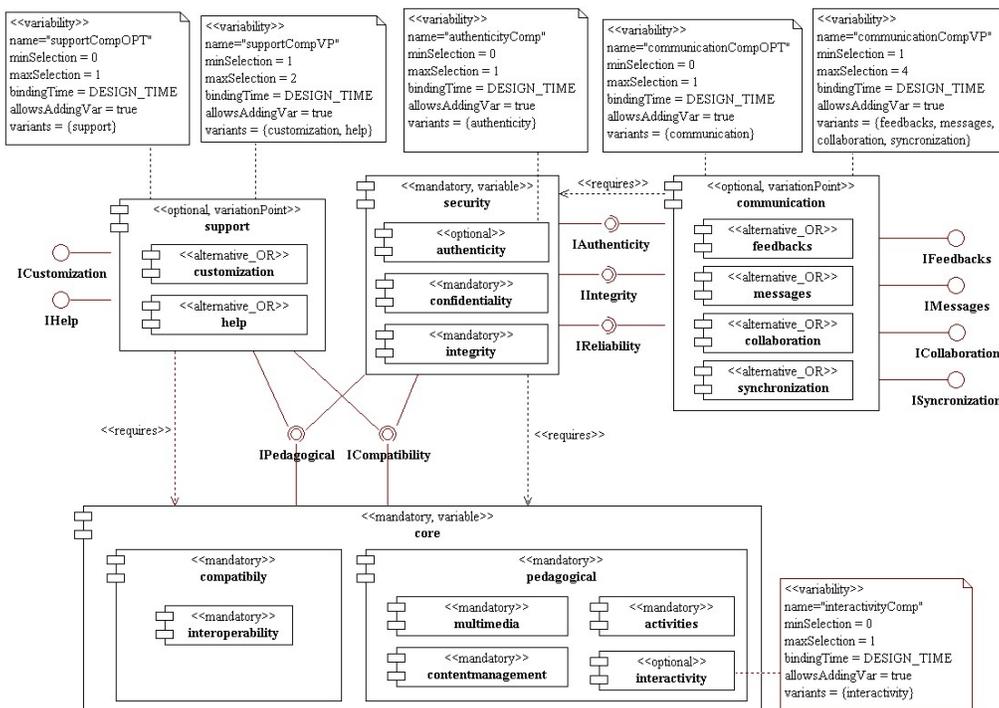


Figure 3. The M-SPLearning Component Diagram

The *RestApp* module is based on the *Representational State Transfer* (REST), an architectural style that is characterized by the use of Web technologies and protocols for the creation and delivery of services [24]. This application accesses a remote database in which all the information of this case study was stored in addition to the features (similarities and variabilities) specified in M-SPLearning and used for the generation of customized products.

A visual interface for creating the products is provided by the *WebApp* application. In this interface, it is possible to configure the variabilities and request the generation of a customized product. This module was implemented only using client-side technologies. The idea was to demonstrate interoperability in a SOA architecture, since *RestApp* module also communicates with an Android module (described next).

Finally, the *AndroidApp* module was developed on the Android platform using the Java programming language. This application allows a service, available in *RestApp* module, executes a custom “build” according to the variabilities configured in *WebApp* application. So, it is possible to generate customized products, which can be installed on Android devices.

The generated product is adapted, in run time, to the variabilities configured at time of its creation. Actually, each application carries its own similarities and variabilities.

In our preliminary implementation, the *compatibility* feature and its sub-feature *interoperability* were built aiming to consumption of REST services. The *security* feature and all its sub-features were also available, allowing secure and reliable authentications. The *interactivity* feature, in turn, allowed the integration with social networks (Facebook and Twitter).

Figure 5 illustrates two products generated by the implementation of M-SPLearning. The first product was

generated with only the optional *authentication* feature; the second product was configured in a similar way, but including the *interactivity* feature.

Although not all the features defined by M-SPLearning were implemented yet, it was possible to apply a set of concrete software architectures that altogether characterize a functional SPL, with some of the features presented in this paper.

#### THREATS TO VALIDITY

Regarding the preliminary validation of M-SPLearning, some threats have been identified:

- *Missing relevant requirements.* The requirements catalog adopted [11] may have omitted important requirements to the m-learning applications domain. This threat was identified since there is no guarantee that the derivation of the original catalog [11] from the ISO [18] included all the features of the target domain.

- *Validation of domain analysis.* In this phase a checklist was proposed. In spite of being answered by experts, this checklist may be not efficient in the case of the proposed questions be not really effective for this validation.

- *Similarities and variabilities.* M-SPLearning is a proposal of SPL, i.e., it has not been fully implemented yet. This can represent a threat, since similarities and variabilities were still not applied in real products, making difficult to evaluate the proposed SPL in practice.

Despite the threats observed, it is important to highlight that the derivation of a PLA from the requirements catalog and the feature diagram, both of them in agreement with the ISO quality standard [18], can significantly increase the quality of M-SPLearning. Controlled and systematic experiments should be conducted in the near future in order to formally validate M-SPLearning and its main aspects.

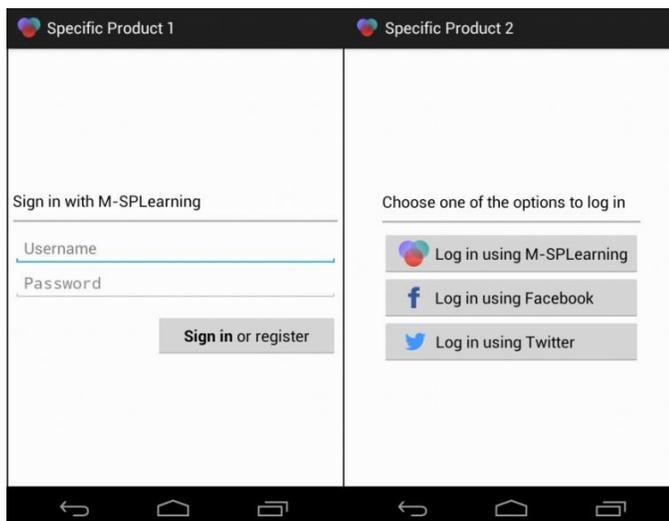


Figure 5. Products generated by the implementation of M-SPLearning

### CONCLUSIONS AND FUTURE WORK

In this paper we describe M-SPLearning – a SPL for m-learning applications. M-SPLearning was developed through a process based on concepts and relevant practices of software engineering. Its main contribution relies on providing benefits with regard to the overall quality, domain comprehension, and reduction of the time spent in the development and maintenance of m-learning applications.

As future work, M-SPLearning must be fully implemented and the generated products should be evaluated with regard to quality and compliance. In this sense, we point out the need of conducting a complete evaluation of M-SPLearning. This evaluation has been planned and will require efforts to develop a considerable set of m-learning applications. Quantitative studies involving detailed experiments to measure the effort required to use M-SPLearning should be conducted as well.

Finally, we also intend to investigate other adoption models. For instance, the extractive model can be applied in similar products to those generated by M-SPLearning aiming at increasing the validity of similarities and variabilities specified. The reactive model can be investigated as an alternative to the evolution of the proposed SPL as well.

### ACKNOWLEDGMENT

The authors would like to thank the Brazilian funding agencies (FAPESP, CNPq and CAPES) and CEPID-CeMEAI (FAPESP 2013/07375-0) for their financial support. We also thank Cast Informática S.A. for supporting this research.

### REFERENCES

- [1] D. Keegan. The incorporation of mobile learning into mainstream education and training. *Proceedings of m-Learning 2005 - 4th World Conference on m-learning*, 2005.
- [2] S. Wexler, J. Brown, D. Metcalf, D. Rogers and E. Wagner. *Mobile learning: What it is, why it matters, and how to incorporate it into your learning strategy*. Guild Research, 2008.
- [3] C. O'Malley, G. Vavoula, J. P. Glew, J. Taylor, M. Sharples and P. Lefrere. *Guidelines for learning/teaching/tutoring in a mobile environment*. Technical Report, MOBIlearn/UoN/UoB/OU, 2003.
- [4] R. C. Durscki, M. M. Spinola, R. C. Burnett and S. S. Reinehr. Software product lines: risks and benefits of your implanting. *VI International Symposium on Software Process Improvement*, 2004.
- [5] F. J. Linden, K. Schmid and E. Rommes. *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.
- [6] P. Clements. Being Proactive Pays Off. *IEEE Software - Point / Counter Point*, p. 28–31, 2002.
- [7] C. Krueger. Easing the Transition to Software Mass Customization. *4th International Workshop on Software Product-Family Engineering*, Springer-Verlag, London, UK, p. 282–293, 2002.
- [8] C. Krueger. Eliminating the Adoption Barrier. *IEEE Software - Point / Counter Point*, p. 28–31, 2002.
- [9] F. Ozdamli and N. Cavus. Basic elements and characteristics of mobile learning. *World Conference on Educational Technology Researches*, Volume 28, p. 937–94, 2011.
- [10] A. Schepman, P. Rodway, C. Beattie and J. Lambert. An observational study of undergraduate students' adoption of (mobile) note-taking software. *Computers in Human Behavior*, 28(2), 308-317, 2012.
- [11] N. F. Duarte Filho and E. F. Barbosa. A Requirements Catalog for Mobile Learning Environments. *SAC'13 Proceedings of the 28th Annual ACM Symposium on Applied Computing*, 2013.
- [12] N. Taylor, N. Medvidovic and E. M. Dashofy. *Software Architecture: Foundations, Theory, and Practice*. John Wiley & Sons, USA, 2009.
- [13] D. L. Dalmon, L. O. Brandão, A. F. Anarosa and S. Isotani. A Domain Engineering for Interactive Learning Modules. *Journal of Research and Practice in Information Technology*. v. 44, p. 309–330, 2012.
- [14] F. G. Marinho, R. M. C. Andrade, C. Werner, W. Viana, M. E. F. Maia, L. S. Rocha, E. Teixeira, J. B. F. Filho, V. L. L. Dantas, F. Lima and S. Aguiar. MobiLine: A Nested Software Product Line for the domain of mobile and context-aware applications, *Science of Computer Programming*, v. 78, p. 2381–2398, 2013.
- [15] F. G. Marinho, A. L. Costa, F. F. P. Lima, J. B. B. Neto, J. B. F. Filho, L. Rocha, V. L. L. Dantas, R. M. C. Andrade, E. Teixeira and C. Werner. An architecture proposal for nested software product lines in the domain of mobile and context-aware applications. *Proceedings - 4th Brazilian Symposium on Software Components, Architectures and Reuse, SBCARS 2010*, p. 51–60, 2010.
- [16] G. G. Pascual, M. Pinto and L. Fuentes. Component and aspect-based service product line for pervasive systems. *CBSE'12 - Proceedings of the 15th ACM SIGSOFT Symposium on Component Based Software Engineering*, p. 115–124, 2012.
- [17] R. Llamas, R. Reith and M. Shirer. Android and iOSCombine for 91.1% of the Worldwide Smartphone OS Market in 4Q12 and 87.6% for the Year. IDC - Press Release, 2013. Available in: <http://goo.gl/WsBRBh>.
- [18] ISO/IEC JTC 1/SC 7. *ISO/IEC 25010: Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models*, 2011. Available in: <http://goo.gl/tpkpPa>.
- [19] J. van Gurp, J. Bosch and M. Svahnberg. On the notion of Variability in Software Product Lines. *Working IEEE/IFIP Conference on Software Architecture (WICSA 2001)*, 2001.
- [20] J. Bosch. *Software Product Lines: Organizational Alternatives*. International Conference on Software Engineering (ICSE'01), 2001.
- [21] 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 University Software Engineering Institute, Pittsburgh, Pennsylvania, 1990.
- [22] E. A. OliveiraJr, I. M. S.Gimenes and J. C. Maldonado. Systematic Management of Variability in UML-based Software Product Lines. In: *J. UCS*, vol. 16, num. 17, pp. 2374–2393, 2010.
- [23] A. S. Nascimento, C. M. F. Rubira and J. Lee. An SPL approach for adaptive fault tolerance in SOA. In *Proceedings of the 15th International Software Product Line Conference (SPLC '11)*, vol.2, art. 15, pp. 1–8, New York, 2011.
- [24] R. T. Fielding. *Architectural Styles and the Design of Network-Based Software Architectures*. Ph.D. Dissertation. University of California, Irvine, 2000.

# Automated transformation of Business Rules into Business Processes

From SBVR to BPMN

Olfa Chourabi TANTAN

(Institut MinesTélécom/Télécom Ecole de Management)  
Evry, France  
Olfa.chourabi@telecom-em.eu

Jacky AKOKA

(CEDRIC-CNAM & Institut MinesTélécom/Télécom  
Ecole de Management)  
Paris, France

**Abstract**—this paper presents a novel approach for transforming Business Rules expressed with Semantic of Business Vocabulary and Rules (SBVR) into (BPMN) Business Process models. This transformation provides several benefits to Information System project stakeholders, such as: enhancing requirement validation and refinement, improving Business Processes documentation, and reducing their overall modeling effort. To bridge the gap between SBVR and BPMN, we propose novel transformation rules that cover structural and behavioral SBVR rules. We illustrate and discuss our transformations with the EU-Rent case study, provided in the OMG specification.

**Keywords**- *Business Rules; Semantic of Business Vocabulary and Rules (SBVR); Business Process Model and Notation (BPMN); Model Driven Architecture (MDA).*

## I. INTRODUCTION

BPMN [1] is a widely used Business Process (BP) modeling language in Information System (IS) Engineering projects [3]. On the basis of Bunge-Wand-Weber (BWW) representation theory, Recker et al. [4] reveals that BPMN models cover a significant number of real word concepts.

However, the current version BPMN 2.0<sup>1</sup> provides a sophisticated notation that could be quite complex for business experts. Feuto et al. [5] stress that business experts usually use Business Rules (BRs) to express their business needs. BRs specify the semantics of the business objects involved in business activities, the constraints (precondition/post condition) on these activities, the reactions to events, as well as the actor's rights and responsibilities in a BP [2].

To facilitate communication between business and IT experts, the Object Management Group have proposed Semantic of Business Vocabulary and Rules (SBVR) specification [2]. This standard offers a meta model for documenting the semantics of business domain concepts and business rules, in an unambiguous and understandable way to humans as well as computer systems (as SBVR is grounded on first order and modal logic [2]).

Since SBVR and BPMN are fully integrated in the MDA approach and behave as Computer Independent Model (CIM),

we advocate that model to model transformation [3] could offer an interesting solution to bridge the gap between business and IT experts. Thus, instead of using BPs models as a requirement validation mean, we could use BRs natural language specification to communicate with business expert and then run transformations to generate the corresponding BP models.

The main objective of this paper is to introduce a transformation methodology from SBVR meta model to BPMN Meta model, in order to ease the communication and the requirement validation step in BPM projects.

Different strategies were used in the literature to address the relationships between Business Rules and Business Processes [7][8][9][10]. However, few works have focused on the automated transformation from SBVR to BPMN, without significant alteration of their original concepts.

A. Raj et al. [11], Z. Wu et al. [13], Roover et al. [14], and B.Steen et al.[12] focus on behavioral aspects of BPs and neglect organizational and informational aspects [15]. Therefore, in this paper we address a more comprehensive coverage of SBVR Business Rules (i.e. structural and operative rules [2]) in order to generate more accurate BPs models. To this end, we have refined the above mentioned transformations and we have formalized basic ideas discussed in [2] [16] and [17] in our transformation methodology.

The remainder of this paper is structured as follows. Section II summarizes SBVR and BPMN meta models. Section III presents and illustrates the new transformation rules from SBVR to BPMN with excerpts from the EU-RENT case study [2]. Section IV is devoted to the conclusion and presents future research directions.

## II. BACKGROUND: SBVR AND BPMN METAMODELS

This section summarizes the main SBVR and BPMN meta model elements that were used in our transformation approach.

### A. *Semantic of Business Vocabulary and Rules (SBVR)*[2]

The SBVR meta model is organized into two main elements:

<sup>1</sup> [http://www.bpmn.de/images/BPMN2\\_0\\_Poster\\_EN.pdf](http://www.bpmn.de/images/BPMN2_0_Poster_EN.pdf)

1) *Business vocabulary*: composed of noun concepts and verb concepts (fact types) used by a community or an organization.

2) *Business rules*: Define the structure and the constraints on the business vocabulary. They are specialized into structural and operatives rules. The formers are definitional rules and express the claims of necessity (alethic modality). The later are related to business behavior (deontic modality).

The specification also offers an English vocabulary for describing vocabularies and stating rules called SBVR Structured English. It is based on the use of font styles and there are of four formatting styles as follows: An object type is represented in underlined green, an individual concept is represented double underlined green, a fact type is represented in italics blue and keywords (cf. 2.semantic formulation) are represented in red. Example: behavior rule: obligation: **Each order must be processed within one business day.**

SBVR specification offers a way to express the logical composition rules [2] called Semantic Formulation. Semantic formulations are specialized into logical formulation and projection. In this paper, we use logical formulation to structure business rules; the projections were used in the literature to express questions, to formalize SPARQL queries [18].

We focus on the five logical formulations that may structure BPs models, namely:

- *Atomic formulation* which refers to propositions that are based on exactly one fact type. (example : EU-Rent purchases from General Motors Company).

Example (mixing Logical operation: implications, and negation): **If the country of the pick-up branch of a rental is not the country of registration of the rented car then it is obligatory that the country of the return branch of the rental is the country of registration of the rented car.**)

- *Modal formulation* which is a Logical formulation that formulates the meaning of another logical formulation. Each modal formulation embeds exactly one logical formulation. Modal formulations are further specialized into necessity, obligation, possibility and permissibility. The SBVR specification provides a set of keywords referring to these modalities. (example: It is prohibited that a rental is open if an estimated rental charge is not provisionally charged for the rental).
- *Quantification*: introduces and constrains a variable in logical formulations, such as: existential quantification
- *Instantiation formulation* which refers to proposition that is based on exactly one fact type and that embeds individual concepts. (example: EU-Rent is a car rental company)
  - *Logical operation* which refers to proposition that are based on one or many fact types. These fact types may formulate negation, conjunctions, disjunctions, implication, nor, nand and wether or not formulations.

Figure. 1 shows an excerpt of the SBVR meta model [2].

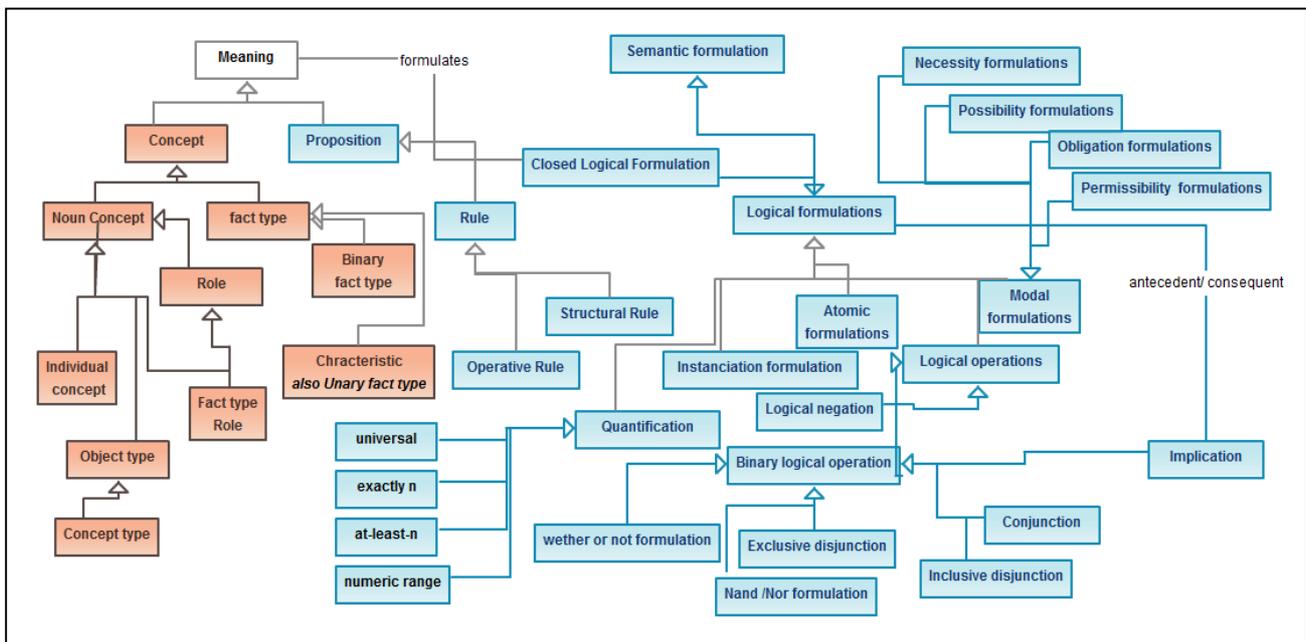


Figure 1. SBVR Meta model excerpt

### B. Business Process Model and Notation (BPMN)

We use the current version BPMN2.0 [1] as a target meta model of our transformation approach. The BPMN meta model is depicted in figure 2. A BPMN process model is a graph consisting of four types of elements.

- *Flow objects*: consisting of activities, gateway and events
- *Connecting objects*: Allow flow objects connections and are further specialized into: sequence flows, default flows, conditional flows, message flows and associations.

- **Swimlanes:** Allow flow object assignment to BPs participants. They are divided into pools and lanes (partition of pools).
- **Artifacts:** Dataobject and textAnnotation are artifacts that represent piece of information processed in a BPs and comments on BPs respectively

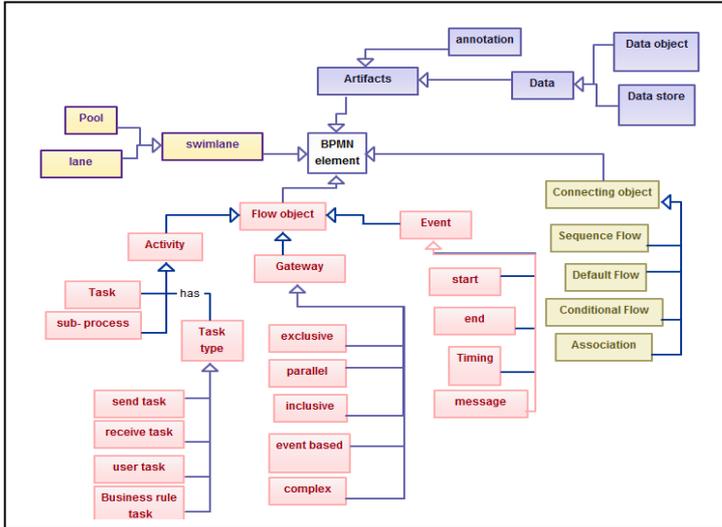


Figure 2. BPMN meta model excerpt

### III. THE TRANSFORMATION APPROACH

#### A. Overview of our approach

Our ultimate aim is to define automated transformations from natural language to a BPMN model. But, in this paper, we focus on model to model transformation from a Structured English SBVR specification to a BPMN model. Text to model transformations proposed in [6] could be used as a starting point to our approach. We make the assumption that the input SBVR SE specification is consistent and complete.

Our approach is divided into two steps:

- Business vocabulary supporting fact types to BRs is transformed to BPMN core BP constructs, namely: BP participants, activities, pre/post conditions, and data objects.(cf. subsection B)
- Business Rules transformations are then applied to define the control flow of the identified core constructs, and to potentially refine them. (cf. subsection C)

#### B. Mapping Business Vocabulary to BPMN primitives

##### 1) SBVR Fact Type transformation

###### a) Binary fact type[11] [12]

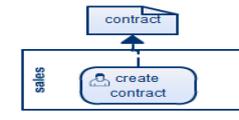
- T1:** Each *binary fact type* is mapped to a **BPMN task**
- T2:** Each *fact type role 1/individual concept role 1* of a binary fact type is mapped to a **BPMN lane**
- T3:** Each *fact type role2/individual concept role2* of a binary fact type complements the **task name** of the transformation T1

#### b) Binary fact mapping refinement

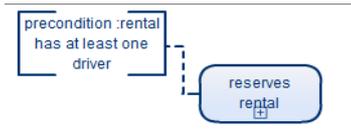
**T4:** Each *binary fact type*, synonym of sends /receives terms is mapped to a **BPMN send/ receive task**. If the *task type* is *sends*, then we add a **message flow** named as the fact type role or the individual concept represented in **role2**  
**SBVR example:** *Booking clerk sends rejection*  
**BPMN output:**



**T5:** Each *binary fact type*, synonym of create/read/update/delete terms (CRUD) is mapped to a **BPMN user task attached to data object**  
**SBVR example:** *Sales creates contract*  
**BPMN output:**



**T6:** Each *is-property-of fact type* is mapped to a **BPMN annotation attached to a task to express a precondition/post condition**  
**SBVR example:** *Rental has at least one driver*  
**BPMN output:**



**T7:** each *specialization fact type related to the identified lanes of (T2)* is mapped to a **BPMN Pool**  
**SBVR example:** *Sales specializes EU rent Enterprise*  
**BPMN output:**



#### c) Binary fact mapping refinement

**T8:** Each unary fact type (*characteristic*) is mapped to:  
 - Message initiating event if it belongs to the first business rule of the SBVR specification  
 -Else it is mapped to an Exclusive gateway

#### C. Mapping Business Rules to BPMN elements

##### 1) Mapping Structural Business Rules

###### a) Mapping integrity rules

**T9:** Each structural rule expressing integrity constraint is mapped to an annotation attached to a task to express **precondition/postcondition**. The logical formulation type is further specialized into atomic formulation, instantiation formulation, modal formulation and logical formulation except implications formulations.  
**SBVR example:** *A customer of the car rental company EU-Rent must be at least 25 years old*  
**BPMN output:**

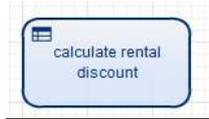


### b) Mapping derivation rules

**T10:** Each derivation rule is mapped to a **BPMN business rule task**<sup>2</sup> (BPMN 2.0 extension); the stereotype business rule is added to a previously identified task with **T1** (based on underlying binary fact type of the derivation rules).

**SBVR example:** It is necessary that If customer is an enterprise, the renter discount is calculated as (20% of the rental price) Else the renter discount is calculated as (10% of the rental price).

**BPMN output:**



NB: The enforcement of this rule simplifies the BPMN model as Business rule task replaces complex paths modeled with gateways

## 2) Transforming operative Business Rules

### a) Transforming Reaction rules and production rules

Except the event concept, reaction and production rule patterns are similar.

- Reaction rule: <event><condition><action>
- Production rule : <condition><action>

This transformation addresses the behavior view of BPMN models. It links *implication formulations*<sup>3</sup> to six basic control flow patterns of BPs models. T11 is an example of sequence pattern identification

**T11:** Each implication formulation expressed as : **If <atomic formulation 1> then < atomic formulation 2>** is transformed to a sequence pattern.

**Constraint :** <atomic formulation 1> doesn't belong to the <antecedent> of another rule

**SBVR example:** if the system display the welcome-screen then the user insert the card



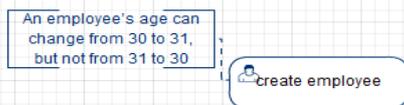
**BPMN output:**

### b) Transforming transforming rules

**T12:** Each operative rule expressing integrity constraint is mapped to an annotation attached to a task to express *precondition/postcondition*

**SBVR example:** An employee's age can change from 30 to 31, but not from 31 to 30

**BPMN output:**



## IV.CONCLUSION

In this paper, an automated approach to transform SBVR meta model to BPMN meta model is proposed. The main objective of this transformation is to assist business experts in the requirement validation phase, as BRs are expressed in natural language. Literature review has revealed that SBVR to BPMN approaches address only the behavioral view of a BP. We tried to offer a more unified view of a BP, by

<sup>2</sup> A Business Rule Task provides a mechanism for the Process to provide input to a Business Rules Engine and to get the output of calculations that the Business Rules Engine might provide." [1]

<sup>3</sup> <http://www.workflowpatterns.com/>

covering the main categories of BRs in our transformations. Future work will address validation issues in real industrial setting, prototype implementation and transformations refinement to BPMN2.0 *choreographies* and *conversation* diagrams.

## REFERENCES

- [1] Business Process Model and Notation (BPMN), v.2.0, OMG Document Number: formal/2011-01-03. <http://www.omg.org>
- [2] OMG, Semantics of business vocabulary and business rules (SBVR), OMG Standard, v. 1.0, January 2008
- [3] MDA, <http://www.omg.org/mda>
- [4] J. Recker, "Opportunities and constraints: the current struggle with BPMN". Business Process Management Journal, vol. 16, no. 1, pp.181–201,2010.
- [5] J.Recker, M.Indulska, M.Rosemann and P.Green, "Do Process Modelling Techniques Get Better? A Comparative Ontological Analysis of BPMN". In 16th Australasian Conference on Information Systems, Sydney, Australia, 2005.
- [6] P.Brillant Feuto, W. El Abed, From Natural Language Business Requirements to Executable Models via SBVR. 2012 International Conference on Systems and Informatics (ICSAI 2012).
- [7] G. Governatori and A. Rotolo, "A conceptually rich model of business process compliance," in 7th Asia-Pacific Conf. on Conceptual Modelling (APCCM). Australian Computer Society, pp. 3–12. 2010.
- [8] A. Charfi and M. Mezini, "Hybrid web service composition: business processes meet business rules," in 2nd Int. Conf. on Service-Oriented Computing (ICSOC). ACM, 2004.
- [9] R. Lu and S. Sadiq, "A survey of comparative business process modeling approaches," in 10th Int. Conf. on Business Information Systems (BIS), ser. LNCS, vol. 4439. Springer, pp. 82–94, 2007.
- [10] L. Nemuraite, T. Skersys, A. Sukys, E. Sinkevicius, L.Blonskis, "Vetis tool for editing and transforming SBVR business vocabularies and business rules into UML&OCL models". In 16th Int. Conf. on Information and Software Technologies, pp. 377–384, Lithuania 2010.
- [11] A. Raj , T. V. Prabhakar and S.Hendryx, "Transformation of SBVR business design to UML models", Proceedings of the 1st India software engineering conference, February 19-22, 2008.
- [12] B.Steen, L.Ferreira Pires and M. Iacob. "Automatic generation of optimal business processes from business rules". 4th IEEE International Enterprise Distributed Object Computing Conference Workshops, 2010.
- [13] Z. Wu, S. Yao, G. He and G. Zue, "Rules Oriented Business Process Modeling", in Proc. of the International conference on Internet Technology and Applications (iTAP), pp. 1–4, 2011.
- [14] W. De Roover and J.Vanthenien. "A Transformation from SBVR Business Rules into Event Coordinated Rules by Means of SBVR Patterns". In Towards a Service-Based Internet. ServiceWave 2010 Workshops, Lecture Notes in Computer Science Volume 6569, pp 172-179, 2011.
- [15] B. List and B. Korherr, "An evaluation of conceptual business process modelling languages," in Proceedings of the ACM Symposium on Applied Computing, vol. 2 , pp 277–302, Springer, New York, NY, USA,2006.
- [16] J. Koehler, "The Process-Rule Continuum— Can BPMN & SBVR Cope with the Challenge?" in IEEE 13th Conference on Commerce and Enterprise Computing(CEC),pp. 302 - 309,2011.
- [17] J.Koehler, "The Role of BPMN in a Modeling Methodology for Dynamic Process Solutions". Business Process Modeling Notation Lecture Notes in Business Information Processing Volume 67, pp 46-62, 2010.
- [18] A.Sukys, L.Nemuraite, B.Paradauskas, "Representing and Transforming SBVR Question Patterns into SPARQL". In Information and Software Technologies, Communications in Computer and Information Science Volume 319, pp 436-451, 2012.

# Practical Human Resource Allocation in Software Projects Using Genetic Algorithm

Jihun Park, Dongwon Seo, Gwangui Hong, Donghwan Shin, Jimin Hwa, Doo-Hwan Bae  
Department of Computer Science  
Korea Advanced Institute of Science and Technology  
{jhpark, dwseo, gwangui.hong, donghwan, jmhwa, bae}@se.kaist.ac.kr

## Abstract

*Software planning is becoming more complicated as the size of software project grows, making the planning process more important. Many approaches have been proposed to help software project managers by providing optimal human resource allocations in terms of minimizing the cost. Since previous approaches only concentrated on minimizing the cost, there has not been a study that considers the practical issues affecting project schedule in practice.*

*We elicited the practical considerations on the human resource allocation problem by communicating with a group of software project experts. In this paper, we propose an approach for solving the human resource allocation problem using a genetic algorithm (GA) reflecting the practical considerations. Our experiment shows that our algorithm considers the practical considerations well, in terms of continuous allocation on relevant tasks, minimization of developer multitasking time, and balance of allocation.*

## 1 Introduction

As the size of software project increases, software planning process becomes more complicated and important. In addition, an inappropriate software plan often results in the failure of a project [15]. For these reasons, software project managers can significantly benefit from the human resource allocation technique. The human resource allocation technique automatically allocates each employee to the tasks to make an optimal plan of the project in terms of time and money.

Many researches have been proposed to deal with the human resource allocation problem. For example, Alba et al. [1] and Chang et al. [6] suggested a genetic algorithm (GA) approach for minimizing the project duration and project cost. Chang et al. [7] suggested a fine-grained representation of a human resource allocation result and

used GA to find a schedule minimizing payment and delay penalty.

While these approaches only considered minimizing the cost in terms of time or money, they did not consider practical issues which can affect the actual development process when the plan is applied in a real world. We discussed with a group of software experts to elicit practical issues that affect the project schedule in practice. We then suggest a GA approach to minimize the inefficient assignments which can delay the schedule.

The rest of this paper is organized as follows. Section 2 explains the practical considerations for the human resource allocation problem and Section 3 describes our genetic algorithm. Section 4 presents a case study, Section 5 discusses threats to validity, Section 6 introduces related work, and Section 7 summarizes our findings.

## 2 Practical Considerations

By consulting with an expert group, we elicited practical issues which can affect the project schedule. The expert group consisted of managers and developers from a software development and consulting company, military software experts from a research institute, and a professor and graduate students. The derived practical considerations are described below.

**C1. Short project plan** The basic objective of human resource allocation problem is to generate a project plan that can be finished within the minimum time span, in order to reduce the entire project cost in terms of time and money.

**C2. Minimization of multitasking time** In practice, a developer can work on more than one task at the same time, while many researches assumed that a developer can work on only one task at a time [3, 9, 18]. Thus, we allow assigning developers to work on multiple tasks at the same time. If a developer is involved in too many tasks at a certain moment, however, productivity will decrease since the developer cannot concentrate on one task due to the fre-

quently switching tasks. In addition, a developer involved in multiple tasks at the same time is more likely to introduce bugs. [11].

**C3. Assignment on relevant tasks** Since the task precedence relationship indicates closely related tasks, assigning a developer to both of the pre-task and the post-task is efficient in terms of minimizing the context-switching cost. If a developer should work on a series of tasks that are not related to each other, the developer must learn the new context for every assigned task, e.g., requirements or design of an unfamiliar module.

**C4. Balance of allocation** Task size should be considered in the human resource allocation problem. On the one hand, if a few developers are assigned to a huge task, developers will be overwhelmed by the heavy workload. On the other hand, if too many developers are assigned to a small task, the high communication overhead causes inefficiency. The staff level (e.g., director/manager/engineer) of developers also should be considered. Developers with high staff level have more experiences than lower level developers, so they will manage a task rather than concentrating on the implementation, which is the main work of a low staff level developer. To manage each task efficiently, developers having different staff levels should be assigned together.

Previous project scheduling algorithms do not reflect the inefficiencies caused by these practical issues, but in practice, project managers consider them very important. Our GA approach takes into account of these practical considerations by representing them as a part of the fitness function.

### 3 Human Resource Allocation with GA

#### 3.1 Problem Description

Our problem is described by the tasks and developers. A software project consists of a set of tasks  $T = \{t_1, t_2, \dots, t_n\}$ . We use the task precedence graph (TPG) to represent the precedence relationship between tasks. Figure 1 shows an example of a TPG. In Figure 1, task  $t_1$  must be completed before  $t_2, t_3$  and  $t_4$  begin. Each task  $t_i (i = 1, 2, \dots, n)$  is defined by the following attributes.

- $type_i$ : The type of the task. In our research, four task types which are the basic phases for software process models are used: analysis, design, implementation, and testing.
- $effort_i$ : The effort estimated by the project manager, which is assumed to be available as an input. Project manager can use existing effort estimation techniques, such as COCOMO models [4, 5], or analogy-based software effort estimation [16]. The unit of effort is man-hours.

- $PT_i$ : A set of preceding tasks for the task. A task cannot begin if any of the preceding task is not completed. TPG is constructed based on the precedence relationship.

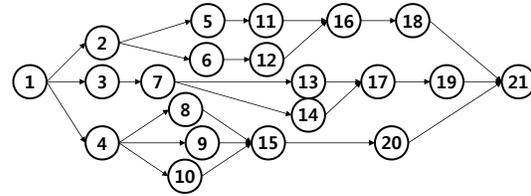


Figure 1. A Task Precedence Graph (TPG)

A set of developers  $D = \{d_1, d_2, \dots, d_m\}$  is allocated to tasks. We assume that the developer information is managed by a database, and the project manager can access it for planning a project. Developer  $d_j (j = 1, 2, \dots, m)$  is defined by the following attributes

- $sl_j$ : The staff level of the developer. In our problem, we assume that there are three hierarchical staff levels—director, manager, and engineer.
- $ability_j^k$ : The ability of the developer for task type  $k$ . Developers have different levels of proficiency on each task type. If a developer has 0.7 as the ability value on the *design* task type, he decreases 0.7 remaining man-hour per a time unit when he is only working on the task.

A single allocation for a task is represented by  $\{t_i, \{d_k, \dots, d_l\}\}$ , and the *assignment result* consists of the allocations for each task.

#### 3.2 Genetic Algorithm

A genetic algorithm (GA) is an evolutionary search-based heuristic algorithm introduced by Holland [12]. Many researches tried to utilize the GA to find an optimal solution for the human resource allocation problem, which has a very large search space [1, 2, 6, 7]. We develop a GA approach reflecting the practical considerations to minimize inefficient allocations that can delay the schedule in practice. Figure 2 shows the pseudocode of our GA. Each step of the algorithm is explained in this section.

**Representation** We define a chromosome to represent an assignment result as a fixed length of genes. The chromosome has  $|T|$  number of genes, where each gene records the assigned developers for each task. For example, the first gene contains a set of developers who are assigned to the task  $t_1$ . Figure 3 shows an example of a chromosome.

**Initial population** The first step of our GA algorithm is to generate an initial population of chromosomes. The initial

```

//Initial population
Generate initial population  $P_0$ 
Initialize generation counter  $g \leftarrow 0$ 

while( $g <$  the maximum number of generation counter){
  Generate empty population  $P_{g+1}$ 
  //Assessment
  Evaluate each chromosome in population  $P_g$ 

  //Elitism Selection
  Copy a chromosome  $c_{best}$  which has the highest fitness
  value from  $P_g$ 

  Copy  $c_{best}$  into  $P_{g+1}$ 
  while(the number of chromosome in  $P_{g+1} \neq$  the number
  of chromosome in  $P_g$  ){

  //Tournament Selection
  Select  $c_1$  and  $c_2$  from  $P_0$  using tournament selection

  //Crossover
  Generate  $c_3$  from Crossover  $c_1, c_2$ 

  //Mutation
  for ( $gene_i$  in  $c_3$ )
    if ( $rand(0,1) <$   $mutation\ rate$ ) Mutate  $gene_i$ 

  Copy  $c_3$  into  $P_{g+1}$ 
  }
  Evaluate each chromosome in population  $P_g$ 
  Copy  $c_{best}$  which has the highest fitness value from  $P_g$ 

return  $c_{best}$ 

```

Figure 2. Genetic algorithm

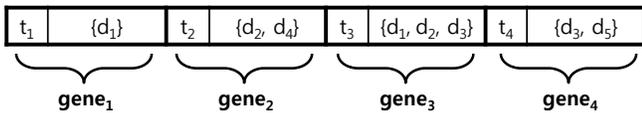


Figure 3. An example of a chromosome

population comprises the *population.Size* number of chromosomes. We randomly assign a developer to a task, until every task has at least one assigned developer.

**Assessment** We evaluate each chromosome to identify a superior one among the population. The fitness function assesses a chromosome by simulating the solution and inspecting the assignment structure. Details of the fitness function is described in Section 3.3.

**Selection** Selection step identifies superior chromosomes that should survive until the next generation and pass their genes down to the next generation. On the one hand, the elitism selection passes on the fittest chromosome of the current generation to the next generation. It prevents degradation of the fittest chromosome. On the other hand, the tournament selection chooses a chromosome to be a parent of the next generation. It randomly selects  $k$  (tournament size) chromosomes from the current population, and picks the fittest one among them. In this study,  $k=5$  is used.

**Crossover** Crossover step generates chromosomes for the

next generation using two parent chromosomes selected by the tournament selection. We use uniform crossover which generates a new chromosome by randomly taking each gene from the parents. We repeat the tournament selection and crossover steps to make new chromosomes until the number of chromosomes in the next generation becomes equal to the predefined population size. Figure 4 shows an example of the uniform crossover.



Figure 4. An example of the uniform crossover

**Mutation** Mutation step maintains genetic diversity. With a certain probability of the mutation (*mutation rate*), each gene is mutated using one of the three mutation operators: *assigning* a random developer to the task, *removing* a random developer from the task, and *replacing* an assigned developer with a random developer. In this study, we use 0.05 for the mutation rate.

### 3.3 Fitness Function

A fitness function assesses each chromosome to identify a superior one among the population. The fitness function encodes the requirements of our GA, which means the better human resource allocation results in the higher fitness score. The formula for our fitness function is as follows.

$$FitnessScore = w_1 * CMscore + w_2 * CEscore + w_3 * CCscore + w_4 * BAscore$$

Each  $w_i$  represents the weight for each score, and each score encodes our practical considerations. CM (Cost Minimization) score assesses whether the project cost is minimized in terms of time, and CE (Concentration Efficiency) score represents how many developers are involved in multitasking at each time unit. CM and CE scores are calculated with the scheduling simulation. CC (Concentration Consideration) score assesses whether developers are assigned to tasks that have precedence relationships, and BA (Balance of Allocation) score represents whether developers are evenly allocated to tasks considering the task size and the staff level. The CM, CE, CC, and BA scores reflect the practical consideration  $C1$ ,  $C2$ ,  $C3$ , and  $C4$ , respectively.

#### 3.3.1 Scheduling Simulation

We inspect the estimated time span of an assignment result and how assigned developers perform each task, using

```

AllocationResult solution
List<Task> remainingTasks
List<Task> enabledTasks
List<Task> runningTasks
List<Task> completedTasks

Add tasks that do not have pre-task to enabledTasks
while(completedTasks.size() != totalNumOfTasks) {
    TimeTick++
    Assign developers to enabled tasks using solution
    running task ← enabled task

    for(runingTask in runningTasks){
        // Perform task
        Decrease remaining man-hour of running tasks
        if (runningTask.remainingMH <= 0){
            // Complete the task
            completed task ← running task
        }
    }
    for(remainingTask in remainingTasks){
        if (all preTask of a remainingTask is completed)
            // Enable the task
            enabled task ← remaining task
    }
}

```

**Figure 5. Scheduling simulation algorithm**

a scheduling simulation algorithm (Figure 5).

The allocation result (variable `solution`) is an input of this algorithm. At the beginning of the algorithm, every task is added to `remainingTasks` and then the task which does not have any pre-task is added to `enabledTasks`.

At the beginning of the while loop, the time tick increases. The start and finish time of each task are calculated with this time tick. The finish time of the last task is regarded as total time span of a project.

For each task in `enabledTasks`, developers are assigned as recorded in `solution`, and the task is moved to `runningTasks`. The remaining man-hour of each task in `runningTasks` is decreased by the sum of the assigned developers' ability. The ability of the developer is assumed to exist as an input. If a developer is working on more than one task, his ability on each task is divided by the number of assigned tasks, because his capability is divided among the tasks.

After decreasing the remaining man-hours of running tasks, running tasks with its remaining man-hour less than or equal to zero is moved to `completedTasks`. Tasks in `remainingTasks` that every pre-task is completed are moved to `enabledTasks`.

### 3.3.2 Fitness Scores

**Cost Minimization (CM) Score** The CM score assesses whether the solution finishes early. (C1) The formula for CM score is as follows.

$$CM\ score = \frac{minTS(S)}{TS(S)}$$

$$minTS(S) = \sum_{k \in types} \frac{\sum_{\{t_i \in T | type_i = k\}} effort_i}{max(ability_1^k, \dots, ability_m^k) * |D|}$$

The  $minTS(S)$  refers to the minimum time span, and  $TS(S)$  refers to the time span of the solution  $S$  which can be calculated by the simulation algorithm. The  $minTS(S)$  is calculated with the assumption that every developer has the highest ability among developers at the given task type and there is no multitasking overhead or communication cost. The CM score becomes higher if the time span of the solution comes to the minimum time span.

**Concentration Efficiency (CE) Score** To assess the burden of multitasking (C2), we calculate the number of tasks that a developer has to deal with at each time unit. At a certain moment, a developer might work on more than one task because the assigned tasks are performed at the same time, but such multitasking can cause inefficiency in practice, which delays the project progress behind the expected schedule. The formula for the CE score is as follows.

$$CE\ score = \frac{1}{|D|} \sum_{d_j \in D} \frac{|\{\forall t | \#involve(d_j, t) \geq 1\}|}{\sum_{t=1}^{finishTime} \#involve(d_j, t)}$$

$|\#involve(d_j, t)|$  refers to the number of tasks that developer  $d_j$  is involved in at time  $t$ . For each developer, we divide the total working time units by the sum of the number of involved tasks at those time units to get the partial CE score. For example, if a developer performs [2, 1, 0, 4] tasks on time unit 1 to 4, the developer level CE score is calculated by  $3 / 2+1+4$ . We average all the partial scores for each developer to get the CE score. The CE score comes to 1.0 if every developer performs only one task at every time unit that he works.

**Continuity Consideration (CC) Score** The CC score assesses how well the human resource allocation result considers the precedence relationship between tasks (C3). We count the number of unit allocation  $ua_x = \{t^x, d^x\}$  that do not consider continuity, in which the developer is not assigned any pre-tasks of the task  $t^x$ . The solution of human resource allocation  $S'$  is represented by a set of unit allocation ( $S' = \{ua_1, ua_2, \dots, ua_n\}$ ), and the CC score is calculated with the formula below.

$$CC\ score = \frac{|\{ua_x \in S' | (\exists ua_k \in S') \wedge (t^k \in PT^x) \wedge (d^k = d^x)\}|}{|S'|}$$

The numerator of the formula counts how many times the developer of a unit allocation ( $d^x$ ) is assigned to any pre-task of the task ( $PT^x$ ). The CC score comes to 1.0 if every unit assignment considers continuity.

**Balance of Allocation (BA) Score** We assess how evenly the developers are allocated to tasks considering the task size and staff level using the BA score (C4). The BA score uses *Shannon entropy* [17] which assesses the uncertainty in a distribution. The normalized *Shannon entropy* is defined

as:  $H = -\sum_{i=1}^n (p_i * \log_n p_i)$ , where  $p_i$  is the probability that each element occurs ( $p_i \geq 0$  and  $\sum_{i=1}^n p_i = 1$ ). The value is maximized when all  $p_i$ s have the same probability, i.e.,  $p_i = 1/n, \forall i \in 1, 2, \dots, n$ . We calculate the BA score with the formula below.

$$BA\ score = \frac{\sum_{s \in \text{staff level}} Entropy^s}{|\{\text{staff level}\}|}$$

$$Entropy^s = -\sum_{i=1}^n (p_i^s * \log_n p_i^s)$$

$$p_i^s = \frac{\frac{\#assigned_i^s}{effort_i}}{\sum_{t_k \in T} \frac{\#assigned_k^s}{effort_k}}$$

$Entropy^s$  represents the entropy value at each staff level, and  $p_i^s$  represents the normalized value for the number of assigned developers of the staff level  $s$  at task  $t_i$  ( $\#assigned_i^s$ ) divided by the effort of the task ( $effort_i$ ). The average of the entropy values for each staff level is calculated to assess the BA score. The BA score becomes 1.0 if all  $p_i$  are the same, meaning that the number of assigned developers to a task is proportional to the effort of the task.

## 4 Case Study

We assess how well our GA reflects the practical considerations, by comparing the result when the GA only considers cost minimization ( $Case_{time}$ , weights for the fitness score =  $\{1, 0, 0, 0\}$ ), and the result when considering all the objectives ( $Case_{all}$ , weights for the fitness score =  $\{0.25, 0.25, 0.25, 0.25\}$ ).

### 4.1 Experimental Setup

We generate three experiment sets. Set1, Set2, and Set3 consist of 11 tasks / 7 developers, 11 tasks / 10 developers, and 21 tasks / 10 developers respectively. Table 1 and Table 2 describe a task set  $T_1$  with 11 tasks and a developer set  $D_1$  with 7 developers. The detail of a developer set with 10 developers and a task set with 21 task set is omitted because of the space limit.

Our GA uses 100 population sizes, and we set the maximum generation count to 400. We inspect the tendency of the fitness value along different GA parameters, then we determine the appropriate parameters that GA explores enough search space to get the optimal solution. Selecting the optimal parameters is beyond the scope of this paper.

We compare 1) *the time span of the result*, 2) *the average time units that each developer works on more than one task*, 3) *the number of assignments not considering the task precedence*, 4) *the number of tasks that only one level developers are assigned*, and 5) *the mean and variance of (the number of assigned developers / effort<sub>i</sub>) for each task*. We repeat each experiment 100 times to compare the average value. Table 3 summarizes the results.

ID	$type_i$	$effort_i$	$PT_i$
$t_1$	Analysis	400	
$t_2$	Design	320	1
$t_3$	Design	240	1
$t_4$	Design	240	1
$t_5$	Implementation	240	2
$t_6$	Implementation	600	3
$t_7$	Implementation	160	4
$t_8$	Test	100	5
$t_9$	Test	80	6
$t_{10}$	Test	70	7
$t_{11}$	Test	80	8,9,10

Table 1. Task set  $T_1$

ID	$sl_j$	$ability_j^k$			
		<i>analysis</i>	<i>design</i>	<i>implementation</i>	<i>test</i>
$d_1$	3	1.25	1	1.25	1.25
$d_2$	3	1.25	1	1.25	0.75
$d_3$	2	0.75	0.75	1	1
$d_4$	2	0.75	1	1	0.75
$d_5$	1	1	0.75	0.75	1
$d_6$	1	0.75	1	0.75	0.75
$d_7$	1	1	0.75	1	0.75

Table 2. Developer set  $D_1$

### 4.2 Experimental results

**The time span of the result** We find that  $Case_{time}$  take 5.99% to 12.07% shorter time span than  $Case_{all}$ . We simulate the time span of each solution, but in practice, several issues can affect the actual time span, such as multitasking overhead, context switching cost, and hierarchical management efficiency. Considering that we minimize these practical issues that can delay the project schedule, we conclude that the difference is acceptable.

**The average multitasking time** We investigate how long developers are involved in multitasking, by studying the average time that a developer works on more than one task. We find that the average multitasking time of each developer in  $Case_{time}$  is 3.51 to 6.94 times of  $Case_{all}$ . High multitasking time represents that developers should work more than one task at the same time for a longer time in  $Case_{time}$  schedule than in  $Case_{all}$ .

**Assignments not considering precedence relationship** We assess whether the human resource allocation result reduces the inefficiency of context switching, by comparing the number of unit assignment (see section 3.3.3) which do not consider the task precedence relationship. We find that the number of assignments not considering precedence relationship in  $Case_{time}$  is 2.31 to 2.72 times of  $Case_{all}$ . The result indicates that developers are assigned to a task of different context from previous task more often in  $Case_{time}$

Metric	Set1		Set2		Set3	
	#task=11	#dev=7	#task=11	#dev=10	#task=21	#dev=10
	All	Time	All	Time	All	Time
Time (h)	342.71	322.17	239.70	225.04	846.30	744.14
Multitasking Time (h)	28.96	101.76	12.49	54.94	58.38	404.93
# no precedence assignments	7.91	18.31	10.92	27.08	15.11	41.17
# tasks only one level assigned	3.73	3.05	1.69	3.07	5.74	6.56
Mean (# assigned devs / $effort_i$ )	2.15e-02	3.50e-02	2.80e-02	4.70e-02	1.07e-02	1.88e-02
Variance (# assigned devs / $effort_i$ )	1.71e-04	7.47e-04	2.69e-04	1.48e-03	4.29e-05	1.24e-04

**Table 3. Experiment results**

than in  $Case_{all}$

**The number of tasks that only one staff level of developers are allocated** To investigate whether developers having different staff levels are evenly assigned to each task, we compare the number of tasks that only one staff level of developers are assigned in  $Case_{time}$  and  $Case_{all}$ . We find that the number is higher in  $Case_{time}$  than in  $Case_{all}$  for Set2 and Set3, but the result is reversed on Set1. Because the number of high level developer is limited and our GA minimize multitasking time in  $Case_{all}$ , there is no significant difference of the number in  $Case_{all}$  and  $Case_{time}$ . We find that the average multitasking time and the number of tasks with only one staff level developers are inversely proportional. The result shows that our GA in  $Case_{all}$  considers the staff level slightly better than  $Case_{time}$ , but minimize multitasking time in  $Case_{all}$  significantly.

**The mean and variance of (the number of assigned developers /  $effort_i$ ) for each task** To identify how many developers are assigned to each task proportional to the effort, and how stable the value is, we investigate the mean and variance of # of assigned developers /  $effort_i$ . Overall, the means and variances are higher in  $Case_{time}$  than in  $Case_{all}$ . This results indicate that more developers are assigned when we only consider time, which can cause communication overhead and longer multitasking time. Moreover, large variances indicate there are more cases that too many developers are assigned to a small task or small number of developers are assigned to a large task.

## 5 Discussion

We use the task type to represent a type of the task, rather than defining a skill set needed to do a task. Many previous approaches [7, 8, 18] used skill sets to represent required capability of a task, but we abstract the skill sets using the task type. Our assumption is that defining and inputting each skill set and the capability of developers for each skill is difficult for project managers, especially when the number of skill is large. A project manager can use our tool by categorizing tasks into several types, and defining proficiency of developers on each category type, which is simpler than

a skill set representation.

Our approach generates only one solution which is the fittest one, not considering relative proportion of each sub-scores. We can apply MOEA approach [13], which generates a set of non-dominated solutions when the fitness function comprises the weighted sum of multiple sub-scores.

## 6 Related Work

Duggan et al. [9] suggested a GA approach as an multi objective evolutionary approach to consider package precedence, full team utilization, and cross-communication overhead. They only considered that the successor package should be developed after the predecessor, not considering that the developers should be assigned to tasks with a precedence relationship to lessen the context switching cost. Barreto et al. [3] suggested a optimization-based approach to find teams satisfying constraints established by the software development organization. These work assumed that developers can work on only one task at a time, which was unrealistic in practice. Their approaches required a fine-grained project plan in which a task size was small enough to be finished by a developer.

Chang et al. [7] proposed a human resource allocation technique based on GA with the concept of time-line. The assignment was represented by a three-dimensional array with time, tasks, and employee axes. As time goes with the pre-defined time unit, the human resources were allocated to tasks. Genetic algorithm was used to search the schedule that minimized payment and delay penalties. Chen et al. [8] improved Chang et al.'s work [7] by introducing the event-based scheduler and the ant colony optimization technique. These approaches could allocate human resources in a fine-grained manner, but the allocation was often too fragmented. In addition, they only concentrated on minimizing the cost, not considering practical issues that we consider in this paper.

Gerasimou et al. [10] investigated the human resource allocation problem using a particle swarm optimization technique. Their fitness function evaluated whether each task finished before the deadline and whether an appropriate de-

veloper was assigned to a task considering the ability of the developer. Their approach allowed developers to work on more than one task at the same time, but did not limit or manage the number of tasks on which a developer can work.

Kang et al. [14] proposed a constraint-based approach considering constraints that affected the project schedule. Their constraints included continuous allocation to related tasks, minimization of sharing developers among tasks, restriction on the number of developers assigned to a task, and avoiding novice teams. These constraints encoded some of practical considerations of ours, but they assumed that each program module can always be developed in parallel, not considering the precedence between modules and the integration of the modules.

## 7 Conclusion

Many approaches have been proposed to find optimal human resource allocations. The majority of them only considered minimizing the expected cost of the plan, and not practical issues that can affect the actual schedule in practice. With a group of software experts, we elicited the practical considerations for the human resource allocation problem. We design a genetic algorithm to reflect the practical issues, by encoding them as a part of the fitness function. The fitness function consists of weighted sum of the four fitness scores considering cost minimization, concentration efficiency, continuity consideration, and balance of allocation. Our experiment shows that when the four fitness scores are considered altogether, our GA generates a practical human resource allocation result, in terms of less multitasking time, less assignments not considering task precedence, and more even allocations than an algorithm which the objective only aims at minimizing the time span.

As future work, we will find the optimal parameters and operations for GA, by studying the effect of mutation rate, tournament size, and weight values for the fitness function, and investigating different selection, crossover, mutation approaches. We will also identify more practical issues that can be applied to our approach.

## Acknowledgements

This work was partially supported by Defense Acquisition Program Administration and Agency for Defense Development under the contract.

## References

[1] E. Alba and J. Francisco Chicano. Software project management with gas. *Information Sciences*, 177(11):2380–2401, 2007.

- [2] G. Antoniol, M. Di Penta, and M. Harman. Search-based techniques applied to optimization of project planning for a massive maintenance project. In *Proceedings of the 21st IEEE International Conference on Software Maintenance, ICSM '05*, pages 240–249, Washington, DC, USA, 2005. IEEE Computer Society.
- [3] A. Barreto, M. Barros, and C. Werner. Staffing a software project: A constraint satisfaction approach. *SIGSOFT Softw. Eng. Notes*, 30(4):1–5, 2005.
- [4] B. W. Boehm. Software engineering economics. 1981.
- [5] B. W. Boehm, R. Madachy, B. Steece, et al. *Software Cost Estimation with Cocomo II with Cdrom*. Prentice Hall PTR, 2000.
- [6] C. K. Chang, M. J. Christensen, and T. Zhang. Genetic algorithms for project management. *Ann. Softw. Eng.*, 11(1):107–139, Nov. 2001.
- [7] C. K. Chang, H. yi Jiang, Y. Di, D. Zhu, and Y. Ge. Timeline based model for software project scheduling with genetic algorithms. *Information and Software Technology*, 50(11):1142 – 1154, 2008.
- [8] W.-N. Chen and J. Zhang. Ant colony optimization for software project scheduling and staffing with an event-based scheduler. *Software Engineering, IEEE Transactions on*, 39(1):1–17, Jan 2013.
- [9] J. Duggan, J. Byrne, and G. J. Lyons. A task allocation optimizer for software construction. *IEEE Softw.*, 21(3):76–82, May 2004.
- [10] S. Gerasimou, C. Stylianou, and A. S. Andreou. An investigation of optimal project scheduling and team staffing in software development using particle swarm optimization. In *ICEIS (2)*, pages 168–171, 2012.
- [11] A. E. Hassan. Predicting faults using the complexity of code changes. In *Proceedings of the 31st International Conference on Software Engineering*, pages 78–88. IEEE Computer Society, 2009.
- [12] J. H. Holland. *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. U Michigan Press, 1975.
- [13] H. Ishibuchi and T. Murata. A multi-objective genetic local search algorithm and its application to flowshop scheduling. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 28(3):392–403, Aug 1998.
- [14] D. Kang, J. Jung, and D.-H. Bae. Constraint-based human resource allocation in software projects. *Software: Practice and Experience*, 41(5):551–577, 2011.
- [15] H. R. Kerzner. *Project Management-Best Practices: Achieving Global Excellence*. John Wiley & Sons, 2014.
- [16] J. Li, G. Ruhe, A. Al-Emran, and M. M. Richter. A flexible method for software effort estimation by analogy. *Empirical Software Engineering*, 12(1):65–106, 2007.
- [17] C. E. Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55, 2001.
- [18] V. Yannibelli and A. Amandi. A knowledge-based evolutionary assistant to software development project scheduling. *Expert Systems with Applications*, 38(7):8403–8413, 2011.

# Mental models analysis based on fuzzy rules for collaborative decision-making

Pedro Ivo Garcia-Nunes  
University of Campinas  
Limeira, Brazil  
p063669@dac.unicamp.br

Ana Estela Antunes da Silva  
University of Campinas  
Limeira, Brazil  
aeasilva@ft.unicamp.br

Antonio Carlos Zambon  
University of Campinas  
Limeira, Brazil  
zambon@ft.unicamp.br

Gisele Busichia Baioco  
University of Campinas  
Limeira Brasil  
gisele@ft.unicamp.br

**Abstract** — The theories of bounded rationality present the need for a collaborative decision-making process based on the coalition between different opinions. Reduction of decision uncertainty is associated with a shared perception of reality. The identification of this common sense depends on the representation of each stakeholder's knowledge. Mental models (MMs) are structures that allow us to represent the knowledge of each agent involved in decision-making processes. The comparison between MMs is essential to obtain a shared description of the problem. This paper proposes a method to compare and analyze mental models. The method is based on a comparative technique for MMs and on the relevance of relations and elements of such models. It is summarized by a fuzzy knowledge base which proposes a quantitative and qualitative analysis of the models. This analysis allows the identification of similar models in order to arrive at a decision that considers several points-of-view and reaches a consensus resolution.

**Keywords** — *Decision-making; Fuzzy Rules; Mental Models.*

## I. INTRODUCTION

Organizational complex problems related to a high number of variables [1] imply the need for several problem solving perspectives that involve the analysis of conflicting information [2]. This context fosters uncertainty and makes the decision-making processes as complex as the issues to be solved by them themselves [3]. The need to consider different ideas, opinions and objectives was studied by the Carnegie School, and is mainly represented by theories of bounded rationality [4]. These theories relate increased efficiency and stability in organizations to a qualitative approach to their decisions. This approach comprises steps of a learning process [5] that is founded on the political coalition among decision makers. Thus, the satisfactory decision resides on the knowledge that is common to a representative number of stakeholders [6]. The identification of this common sense depends on knowledge representation.

Mental models (MMs) are structures that allow the representation of knowledge of each stakeholder involved in the decision-making process [7]. Therefore, the comparison of models is essential for obtaining a shared description of the phenomena. MMs are constructed by elements that denote concepts connected by cause-effect relations. The natural language on which these relations are based is one of the advantages of MM representation [8]. They are simple to be created and easily understood. However, the linguistic nature adds difficulty to the mathematical translation of these models, their elements and relations. A mathematical evaluation is needed to enable comparison between the models. This evaluation allows us to numerically measure the similarities between the models. The method used for mathematically comparing MMs is based on the work of [9]. Apart from a mathematical measure, the search for models with similar features should also consider their suitability to the problem description. Thus, this search must also observe the qualitative features used by each mental model.

In order to achieve this, the present paper proposes the creation of a fuzzy rule base for MM analysis based on quantitative and qualitative parameters. Quantitative parameters are obtained when the models are pair-compared (inter-model comparative analysis) through the distance method presented in Section II. Qualitative parameters are obtained when a model is solely analyzed (intra-model analysis)

through the fuzzy rule base described in the methodology section. Fuzzy rules should add flexibility [11] [12] to the analysis and take into account: the complexity of the decision-making process and the linguistic nature of MMs [13].

## A. Mental Models: Basic Concepts

The solution of a problem is necessarily related to the peculiar way in which each decision-maker understands and abstracts the problematic situation. This abstraction and knowledge generation comprise a cognitive process based on the integration of concepts and ideas that can explain the phenomena. According to [14], this cognitive process connects the decision-maker to the reality through the creation of an idiosyncratic knowledge that contains particular beliefs, values and principles. Although this knowledge is individual, it influences the entire decision-making process. The individual experience of the mind builds models that can also be used later on to predict similar events [15]. The mental model theory affirms that all human reasoning is established by means of logical artifacts that depict real events through the causal relations among them. These artifacts, or elements of causal relations represent concepts that describe the problem domain. Due to such characteristics, MMs are used here as representations of human reasoning, which are able to reproduce the idiosyncrasies of tacit knowledge that a decision-maker is willing to express [16]. This paper considers the representation of mental models based on System Dynamics (SD) notation [17] [18].

## II. RELATED WORK

Two approaches of related works are presented in this section. The first one relates to methods of quantitative comparison between MMs. The second concerns the application of fuzzy logic for describing the relations between elements of MMs.

### A. Mental model's comparison method

Since organizational learning concepts were developed [5] and the Carnegie Model of decision-making (theories of bounded rationality) [4] was proposed, the comparison and analysis of MMs as a way to share knowledge have been subjects of interest to many researchers [19]. According to [9], MM comparison is possible through the calculation of distance ratios (DRs) [10]. The element distance ratio (EDR) is given by the ratio of real and potential differences between the elements of two models. Figure 1 shows two MMs A and B. Each model has a number of elements  $V$  (elements  $a$  in model A and elements  $b$  in model B), loops and their polarities (reinforcement or balancing). For each model there is an adjacency matrix  $v \times v$  ( $a \times a$  in the case of model A and  $b \times b$  for model B). In each matrix, index  $i$  denotes rows and index  $j$  denotes columns. Each element of this type of matrix represents the existence (or non-existence) of a relation between two elements. The non-existence is represented by "0"; the reinforcement loop by "1" and the balancing loop by "-1". The EDR is presented in (1) where  $vuA$  is the number of unique elements of mental model A;  $vuB$  is the number of unique elements of mental model B; and  $vC$  is the number of common elements to the two models compared.

$$EDR(A,B) = \sum \text{diff}(i, j) / (4v_C^2 + (2v_C(vu_A + vu_B) + vu_A^2 + vu_B^2) - (2v_C(vu_A + vu_B))) \quad (1)$$

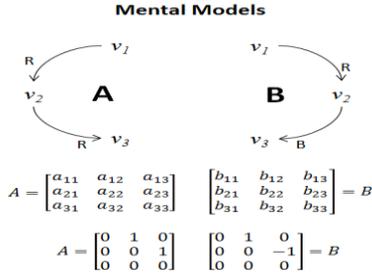


Fig. 1. Mental models and their respective representation matrices

The numerator of (1) represents the sum of relevant differences between the elements of the models. This number of significant differences is defined in (2), where  $V_A$  is the set of unique elements of model A;  $V_B$  refers to the set of unique elements of model B and  $V_C$  is the set of common elements to both models.

$$\begin{aligned} \text{diff}(i, j) &= 0, \text{ if } i = j \text{ and } a = 1; \\ &= \Gamma(a_{ij}, b_{ij}), \text{ if } i \text{ or } j \notin V_C \text{ and } i, j \in V_A \text{ or } i, j \in V_B; \\ &= |a_{ij} - b_{ij}| + \delta, \text{ if } a_{ij}b_{ij} < 0; \\ &= |a_{ij} - b_{ij}|, \text{ otherwise.} \end{aligned} \quad (2)$$

The conditions of the function  $\text{diff}(i, j)$  determine that the difference is "0" for the cells located on the diagonals of the matrix, because there is no self-loops, i.e., an element cannot relate to itself. The difference between the common elements ( $V_C$ ) that comprise loops in which the polarity is not the same is given by the difference of the values. This difference can be considered important or not by the decision-maker. Thus,  $\delta=1$ , if the agent considers the polarity difference to be important, or  $\delta=0$  otherwise. The difference between the unique elements of the two models ( $V_A$  and  $V_B$ ) must satisfy the conditions given by (3). Thus,  $\gamma=0$  indicates that the decision-maker does not consider the difference to be important, whereas a meaningful difference is indicated by  $\gamma=1$ . If the unique elements of the models are not related to any other element, the difference between the elements of the models is "0".

$$\begin{aligned} \Gamma(a_{ij}, b_{ij}) &= 0, \text{ if } \gamma = 0; \\ &= 0, \text{ if } \gamma = 1 \text{ and } a_{ij} = b_{ij} = 0; \\ &= 1, \text{ otherwise.} \end{aligned} \quad (3)$$

In (1), the EDR value belongs to interval  $[0, 1]$  where the minimum value represents full proximity of the elements and the maximum value expresses the absence of common elements. Although the comparison between the elements of two models is necessary, it is not sufficient to fully understand the differences between them [9]. Another comparison, which concerns the feedback loops and its polarities, is also useful. This comparison considers: the loop distance ratio (LDR) and the model distance ratio (MDR).

The LDR indicates the difference value between the loops of the models. Again, the indication occurs over the interval  $[0, 1]$  where the minimum value expresses full similarity, and the maximum full difference. The LDR calculation considers: the EDR of the elements from loop  $m$  of model A and of the elements from loop  $n$  of model B, the difference between the number of delays LDD ( $m, n$ ), and the difference between the polarities  $lpold(m, n)$ . In System Dynamics,

a delay indicates that the causality relation may not happen immediately. Equation (4) shows the elements of the LDR calculation.

$$LDR(m, n) = \eta * ldd(m, n) + \iota * lpold(m, n) + \kappa * EDR(A, B) \quad (4)$$

Parameter  $ldd(m, n)$  takes the value "0" if none or both of the compared loops have delays. If only one loop has a delay,  $ldd(m, n) = 1$ . Parameter  $lpold(m, n)$  takes the value "0" if the loops have the same polarity, and "1" otherwise. Thus, if both compared loops have the same polarity,  $lpold(m, n) = 0$ , otherwise  $lpold(m, n) = 1$ . The weights  $\eta$ ,  $\iota$  and  $\kappa$  represent the importance of each parameter. These weights are assigned to the terms of equation (4) by the decision-maker, considering  $\eta + \iota + \kappa = 1$ . The ratio between the sum of all LDRs and the total number  $n$  of loops results the MDR defined in (5). MDR is the consolidated index that indicates the level of similarity or difference between the models [13].

$$MDR(A, B) = \sum LDR(m, n) / n \quad (5)$$

The values obtained through (5) are submitted to a scale of comparison that consists of three possible conditions results for compared models [13]: (a) Identical, if  $MDR(m, n) = 0$ ; (b) Similar, if  $MDR(m, n) < 0.25$ ; and (c) Different, if  $MDR(m, n) \geq 0.25$ .

The distance method presented provides a quantitative comparison of MMs that considers: elements, loops and types of loops. However, to obtain support for a collaborative decision-making process by using models with similar features, it is also necessary to consider the qualitative analysis of each model, which, in turn, should take into account the adequacy of the model representation to the problem description. The methodology proposed in this paper (Section III) presents additional concepts to the method of distance ratios discussed in this paper but differs from it as the proposal takes into account both the quantitative and the qualitative analysis of the MMs.

### B. Fuzzy logic applied to mental models

When a mental model is structured by the decision-maker, the knowledge about the phenomenon is translated by means of cause-and-effect loops. The loops are endowed with imprecision, since elements can be influenced at different degrees [19]. This granularity increases because MMs are essentially based on natural language. Fuzzy logic is presented as a suitable alternative for the formal representation of the uncertainty associated with variables and linguistic terms used in MMs [11].

In 1986, Kosko introduced the idea of Fuzzy Cognitive Map (FCM) [20]. His proposal aimed at simulations based on the analysis of Axelrod's cognitive maps (CMs). FCM is a graphical structure used to represent the causality between concepts, i.e., the elements. They are MMs whose causality relations assume fuzzy values. Another proposal, the concept of Rule-Based Fuzzy Cognitive Map (RB-FCM), considers that the FCM approach does not explore the qualitative potential of Fuzzy Logic. Therefore, the RB-FCMs allow us to represent the causality between concepts by means of a fuzzy rule base [21]. The use of a rule base allows us to indicate the relation between concepts by associating the elements with linguistic variables that indicate the degree wherewith the concepts influence each other. A base is constituted by "If... then" rules. The antecedents and consequences of these rules have variables that are associated with membership functions corresponding to the linguistic values.

The FCMs [20] and the RB-FCMs [21] use fuzzy logic and the construction of a fuzzy rule base, respectively, in order to refine the representation of a mental model. These strategies are used to represent the causal relations between elements in order to construct and simulate MM relations. The current work also uses these strategies; however, it proposes that fuzzy logic and a fuzzy rule base are used for MM analysis. This analysis is not intended to build or

simulate models, but to add a qualitative approach to the method of MM comparison [9].

### III. METHODOLOGY: FUZZY RULE BASE FOR MENTAL MODELS ANALYSIS

As previously mentioned, this paper proposes a method based on the development of a rule base, whose variables are parameters of comparison and analysis of MMs. The base execution result is a value associated with each mental model. This value indicates the degree of adequacy of the model to represent a problem domain.

The rule base considers quantitative and qualitative parameters. The quantitative parameters are the ones presented in Section II: EDR (element distance ratio), LDR (loop distance ratio) and MDR (model distance ratio). The analysis is completed with qualitative parameters of this section.

#### A. Qualitative parameters of mental models analysis

The analysis of the internal features of a model takes into account the idiosyncratic nature of MMs [16] by considering the way wherewith decision-makers select and relate the concepts in order to describe reality. Some parameters that can aid this intra-model analysis, i.e. the analysis of the internal structure of models, were considered and are presented as follows.

The agent proximity parameter measures the degree of knowledge of the decision-maker about the elements of a mental model. This input is informed by decision-makers according to their degree of knowledge and expertise regarding the elements of the model. In a mental model, some elements have greater influence on problem description than others. The problem proximity parameter concerns the degree of influence of each element over the entire mental model that represents the problem domain. This other input is also informed by the decision-maker. Thus, each element is associated with the parameters: agent proximity and problem proximity. These two parameters generate a general proximity related to each element of the mental model. Furthermore, the comparison between elements of model A with the elements of another model must be considered in the calculation of the EDR. This comparison involves the intra and inter-model analysis of the element level, called element relevance.

Relevance refers to the appropriateness of a certain element to describe a problem [22]. The relevance of an element is obtained from the general proximity and the EDR. The next parameter to be considered for the intra-model analysis concerns the relations of a model. The loop relevance is determined by the relevance of the elements. A loop consists of two elements and its relevance stems from the relevance of these two elements. Thus, loop relevance consists in the association of the relevance of each element,  $v_1$  and  $v_2$ , in the example. The analysis is performed for all elements and progresses over all loops. Thus, the analysis emerges throughout the whole model. This time, the combination of intra-model analysis and inter-model comparison is made through the use of LDR and the search for an equivalent loop in another model. The presence of equivalent relations in two models in Fig. 1 allows us to assume that these relationships are more representative for a description of the problem than those which do not occur in another model. This is based on the fact that the recurrence of a particular descriptive structure shows a homogeneous and consensual perception of reality [23]. In this sense, the structure in question represents the ideas, views, visions, goals and interests of a significant number of stakeholders [6].

The representativeness analysis should progress through all the loops of mental model in order to obtain a general representativeness. The complete amount of loops determines the consolidated representativeness, i.e., the consideration of the set of each general representativeness. Finally, MDR can also be used to combine the intra-model analysis and the inter-model analysis, which can be done

by comparing the MM with the other model. The combination is performed through the use of the consolidated representativeness of model A and model B. These models are compared by means of MDR calculation used to obtain the representativeness of the analyzed model (model A).

#### B. Rule base and linguistic variables

The MM analysis method presented in this work is implemented by a knowledge rule base whose variables are the parameters for intra-model analysis and inter-model comparison. The base execution result is a value for each mental model. This value indicates the degree of adequacy of the model to describe the problem. All parameters mentioned in sections II and III are considered to construct the rule base. The fuzzyfication of these parameters adds flexibility to the result. The fuzzyfication steps are presented below.

Each parameter explained in sections II and III was transformed into a variable of the base. Table I shows each variable, its explanation and its level of analysis. The variable can refer to one of the three levels of analysis: element level, loop level (cause-effect relation) or model level. Linguistic values and their membership functions were assigned to each variable. The linguistic variables are composed of a set  $V = \{Low, Medium, High\}$ . The membership functions of linguistic values are presented in Fig. 2. For parameter MDR, three possibilities were considered [9] to compare models according to Section II: (a) Identical, if  $MDR(m, n) = 0$ ; (b) Similar, if  $MDR(m, n) < 0.25$ ; (c) Different, if  $MDR(m, n) \geq 0.25$ . In this scale of comparison, variable MDR is only associated with the linguistic terms "High" and "Low". As can be seen in the graph in Fig. 4, these terms are denoted by membership functions  $f_{MDR} = \text{"High"}$  and  $f_{MDR} = \text{"Low"}$ , respectively.

Each variable is associated with linguistic values represented by the membership functions of the graphs in Fig. 2 and Fig. 3. These variables comprise the antecedents and consequences of the rules. The chaining of rules enables the emergence of the analysis throughout the mental model. The antecedents are treated by operations of the Mamdani's inference method [24]. The two operations used are presented in (6) and (7). Equation (6) shows that the logical conjunction between two antecedent terms A and B results in a membership function  $f_C(z)$  associated with a consequent term C. This consequent term is obtained through the minimum value of the membership functions  $f_A(x)$  and  $f_B(y)$  associated with the linguistic values. The activation of more than one rule implies a composition between those rules through the maximum value of these functions, which can be calculated by (7). The base was designed in forward chaining with operators of Mamdani's inference method. The linguistic values of variables related to antecedent and consequent terms of the rules are shown in Tables II, III and IV.

$$f_C(z) = f_{A \wedge B}(x, y) = \text{MIN} [f_A(x), f_B(y)] \quad (6)$$

$$f_C(z) = \text{MAX} [f_{C1}(z), f_{C2}(z)] \quad (7)$$

Table II shows the consequent linguistic values related to the variable  $\text{General}_{Proximity}$ .  $\text{General}_{Proximity}$  is obtained from the intersection of the variables  $\text{Agent}_{Proximity}$  and  $\text{Problem}_{Proximity}$ , which are associated with the antecedent terms. This table also shows the linguistic values associated with the variable  $\text{Loop}_{Relevance}$ .  $\text{Loop}_{Relevance}$  results from the variable  $\text{Element}_{Relevance}$  referring to each element that comprises the loop. These terms have the same linguistic values associated with the variable  $\text{General}_{Representativeness}$ .  $\text{General}_{Representativeness}$  is obtained from the values related to the antecedent term variable  $\text{Loop}_{Representativeness}$ . The general representativeness of the model emerges from the evaluation of pairs of loops and their representativeness.

The linguistic values related to the variable  $\text{Consolidated}_{Representativeness}$  are the same to those presented in Table II. These values are obtained from the  $\text{General}_{Representativeness}$  values of the antecedent terms. The

general representativeness values are consolidated to form a global analysis of the model. Table III shows the linguistic values related to the variable  $Element_{Relevance}$ . This variable is obtained from the intersection of the variables  $General_{Proximity}$  and EDR. These variables are associated with the antecedent terms and the linguistic values are the same terms associated to variable  $Loop_{Representativeness}$ . This variable is obtained from  $Loop_{Relevance}$  and LDR antecedent terms. Table IV shows the linguistic values associated with  $Model_{Representativeness}$ . This variable is obtained from the intersection of variable  $Consolidated_{Representativeness}$  and variable MDR.

TABLE I. LINGUISTIC VARIABLES

Linguistic variable	Variable meaning	Level
$Agent_{Proximity}$	Index that refers to ability of the agent to act on the element	Element
$Problem_{Proximity}$	Index that refers to the ability of the element to describe the problem	Element
$General_{Proximity}$	Consolidation of $Agent_{Proximity}$ and $Problem_{Proximity}$ information	Element
EDR	Element Proximity Ratio	Element
$Element_{Relevance}$	Relevance of a particular element in the problem description, considering EDR	Element
$Loop_{Relevance}$	Relevance of a particular loop in the problem description	Loop
LDR	Loop Proximity Ratio	Loop
$Loop_{Representativeness}$	Representativeness of a particular loop to the problem description, considering LDR	Loop
$General_{Representativeness}$	General representativeness of the model through the consideration of more than one loop.	Loop
$Consolidated_{Representativeness}$	Consolidation of all the measures of general representativenesses of the model	Model
MDR	Model Proximity Ratio	Model
$Model_{Representativeness}$	Consolidated representativeness of the model for problem description considering MDR	Model

TABLE II. LINGUISTIC VALUES OF THE CONSEQUENT VARIABLE  $GENERAL_{PROXIMITY}$ ,  $LOOP_{RELEVANCE}$ ,  $GENERAL_{REPRESENTATIVENESS}$  AND  $CONSOLIDATED_{REPRESENTATIVENESS}$

$Agent_{Proximity}/Problem_{Proximity}$	High	Medium	Low
$Element_{Relevance}$			
$General_{Representativeness}$			
High	High	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

After obtaining the linguistic value associated with membership function  $f_{Model_{Representativeness}}$ , it is necessary to transform the fuzzy value into a crisp value. This defuzzification process represents the interpretation of the fuzzy set that results from the inference process.

TABLE III. LINGUISTIC VALUES OF THE CONSEQUENT VARIABLE  $ELEMENT_{RELEVANCE}$  AND  $LOOP_{REPRESENTATIVENESS}$

$General_{Proximity}/EDR$	High	Medium	Low
$Loop_{Relevance}/LDR$			
High	Medium	High	High
Medium	Low	Medium	High
Low	Low	Low	Medium

TABLE IV. LINGUISTIC VALUES OF THE CONSEQUENT VARIABLE  $MODEL_{REPRESENTATIVENESS}$

$Consolidated_{Representativeness}/MDR$	High	Low
High	Medium	High
Medium	Low	High
Low	Low	Medium

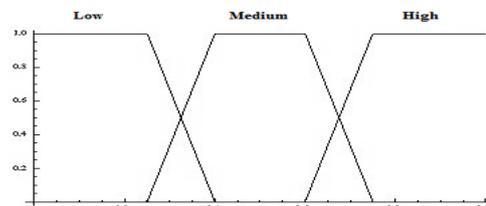


Fig. 2. Membership functions associated with the linguistic values

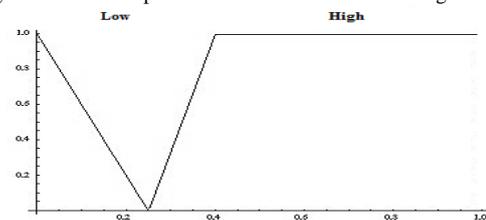


Fig. 3. Membership functions associated with MDR linguist values

The values of the area of this set and their corresponding degrees of membership are converted into crispy numbers. Thus, the linguistic values are converted to a single precise value through the center of gravity method. Equation (8) illustrates this method and allows us to determine the central point  $G(C)$  of the function area associated with the output linguistic value.

$$G(C) = \sum f_C(z_i) / f_C(z_i) \quad (8)$$

#### IV. EXAMPLE OF APPLICATION OF METHODOLOGY

An example of application of the method of Section III is presented here and involves three MMs: A, B and C. These models are proposed to describe the problem of soil erosion in any region. As can be seen in Fig. 4, the three MMs consider that erosion is influenced by vegetation. Model A and model C indicate the influence of deforestation on vegetation. Model A associates deforestation with logging, whereas mental model C indicates agribusiness as the main cause of deforestation. However, mental model B describes the influence of drought on deforestation as a consequence of global warming.

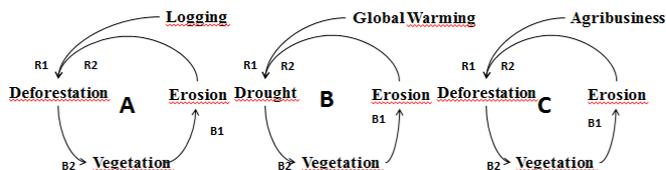


Fig. 4. Examples of mental models to describe the erosion problem

**A. Inter-model comparative analysis**

The quantitative comparative analysis of these models is given by the application of the method presented in Section III. All the models will be compared using this method. Table V shows a survey of the parameters relating to comparisons between models A, B and C. These parameters are applied to (1) to obtain the EDR for each model comparison. The EDR values are shown in Table V.

Then the analysis advances to the loops that comprise each model. The comparison between the loops of the models begins with the search for similar loops. Table VI presents: the verification of existence of these loops, the difference between the number of delays (Ldd), and the difference in the polarity of the loops (Lpold). These parameters are applied to (4) to calculate the LDR related to each comparison between loops. The calculation can only be performed between loops that have the same elements in both models; otherwise the LDR automatically assumes the value "1". LDR values are shown in Table V. MDR can be calculated by (5) using the LDR calculation and the number of n cycles in each model. The values of MDR corresponding to each comparison are shown in Table VII.

TABLE V. COMMON ELEMENTS (v<sub>C</sub>), EXCLUSIVE ELEMENTS FROM EACH MODEL (v<sub>uA</sub>, v<sub>uB</sub> E v<sub>uC</sub>) AND EDR ASSOCIATED TO EACH COMPARISON

Models Comparison	v <sub>C</sub>	v <sub>uA</sub>	v <sub>uB</sub>	v <sub>uC</sub>	EDR
A-B	2	2	2	-	0.083
A-C	3	1	-	1	0
B-C	2	-	2	2	0.083

TABLE VI. LOOPS COMPARISON AND LDR VALUES

Models	Loops	Equivalent loops	Ldd	Lpold	LDR
A-B	B1 <sub>A</sub> -B1 <sub>B</sub>	YES	0	0	0.042
	B2 <sub>A</sub> -B2 <sub>B</sub>	NO	-	-	1
	R1 <sub>A</sub> -R1 <sub>B</sub>	NO	-	-	1
	R2 <sub>A</sub> -R2 <sub>B</sub>	NO	-	-	1
A-C	B1 <sub>A</sub> -B1 <sub>C</sub>	YES	0	0	0.042
	B2 <sub>A</sub> -B2 <sub>C</sub>	YES	0	0	0.042
	R1 <sub>A</sub> -R1 <sub>C</sub>	NO	-	-	1
	R2 <sub>A</sub> -R2 <sub>C</sub>	YES	0	0	0.042
B-C	B1 <sub>B</sub> -B1 <sub>C</sub>	YES	0	0	0.042
	B2 <sub>B</sub> -B2 <sub>C</sub>	NO	-	-	1
	R1 <sub>B</sub> -R1 <sub>C</sub>	NO	-	-	1
	R2 <sub>B</sub> -R2 <sub>C</sub>	NO	-	-	1

TABLE VII. MODELS COMPARISON AND MDR VALUES

Model	Loops	LDR	n Loops	MDR
A-B	B1 <sub>A</sub> - B1 <sub>B</sub>	0.042	8	0.380
	B2 <sub>A</sub> - B2 <sub>B</sub>	1		
	R1 <sub>A</sub> - R1 <sub>B</sub>	1		
	R2 <sub>A</sub> -R2 <sub>B</sub>	1		
A-C	B1 <sub>A</sub> - B1 <sub>C</sub>	0.042	8	0.140
	B2 <sub>A</sub> - B2 <sub>C</sub>	0.042		
	R1 <sub>A</sub> - R1 <sub>C</sub>	1		
	R2 <sub>A</sub> -R2 <sub>C</sub>	0.042		
B-C	B1 <sub>B</sub> - B1 <sub>C</sub>	0.042	8	0.380
	B2 <sub>B</sub> - B2 <sub>C</sub>	1		
	R1 <sub>B</sub> - R1 <sub>C</sub>	1		
	R2 <sub>B</sub> -R2 <sub>C</sub>	1		

**B. Intra-model analysis**

The qualitative intra-model analysis occurs through the submission of models A, B and C to the base of rules presented in the fourth section. Table VIII provides the values assigned to the Agent<sub>Proximity</sub> and Problem<sub>Proximity</sub> variables. These variables are informed by the decision-makers when their models are built. Thus, the values attributed to the variables become the input parameters for the qualitative analysis.

These values are submitted to (6) of Mamdani's method. Some of the rules are activated by these values. For all the operations in this section, MIN [fA(x), fB(x)] = 1.

As demonstrating the activation of all rules would be too tiring for the reader, the algorithm of Fig.5 is used to illustrate it. The algorithm demonstrates the analysis of model A through the comparison to model B.

TABLE VIII. INPUT PARAMETERS

Model	Element	Problem <sub>Proximity</sub>	Agent <sub>Proximity</sub>
A	Logging	1	0.8
	Deforestation	1	1
	Vegetation	1	0.4
	Erosion	1	0.2
B	Global Warming	1	1
	Drought	1	0.8
	Vegetation	1	0.4
	Erosion	1	0.2
C	Agribusiness	1	0.8
	Deforestation	1	1
	Vegetation	1	0.4
	Erosion	1	0.2

**Input:** two mental models (A and B); a knowledge base consisting of 60 rules of inference, whose linguistic values of the variables are summarized in Tables II, III and IV.

**Output:** values corresponding to representativeness degree of each model.

1. Calculate EDR, LDR and MDR about the models A and B, using Equations (1), (4) and (5), respectively;
2. For each element of the mental model A, do:
  - 2.1. Evaluate General<sub>Proximity</sub> considering Agent<sub>Proximity</sub> and Problem<sub>Proximity</sub>, according to Table II;
  - 2.2. Evaluate Element<sub>Relevance</sub> considering General<sub>Proximity</sub> and EDR, according to Table III;
3. For each relationship between two elements of the mental model A, do:
  - 3.1. Evaluate Loop<sub>Relevance</sub> considering Element1<sub>Relevance</sub> and Element2<sub>Relevance</sub>, according to Table II;
  - 3.2. Evaluate Loop<sub>Representativeness</sub> considering Loop<sub>Relevance</sub> and LDR, according to Table III;
4. For each pair of loops of mental model A, do:
  - 4.1. Evaluate General<sub>Representativeness</sub> considering Loop1<sub>Representativeness</sub> and Loop2<sub>Representativeness</sub>, according to Table II;
5. For all pairs of loops of mental model A, do:
  - 5.1. Evaluate Consolidated<sub>Representativeness</sub> considering General1<sub>Representativeness</sub> and General2<sub>Representativeness</sub>, according to Table II;
  6. Evaluate Model<sub>Representativeness</sub> considering Consolidated<sub>Representativeness</sub> and MDR, according to Table IV;
  7. Apply G (C) in Model<sub>Representativeness</sub> using Equation (8);
  8. Repeat steps 2-7 considering the mental model B.

Fig. 5. Algorithm - Performs analysis of the representativeness of a mental model.

This algorithm must be performed for all possible pairs of mental models related to the given problem. Therefore numerical values can vary for each comparison because EDR, LDR and MDR depend on the models being compared. Considering that model representativeness also varies according to the EDR, LDR and MDR, each model has a value G(C) associated with each comparison. Therefore, the measurement of representativeness of each model can be obtained by calculating the arithmetic mean of all the values of G(C). Thus, the average value of G(C) of a model must consider the total number of m MMs to which this model is compared. Table IX shows the values of G (C) associated to each comparison and the average values of each mental model.

TABLE IX. MEASUREMENT OF MODEL REPRESENTATIVENESS

Model	Comparison	G(C)	Comparisons	Average
A	A-B	0.2	2	0.5
	A-C	0.8		
B	B-A	0.2	2	0.2
	B-C	0.2		
C	C-A	0.8	2	0.5
	C-B	0.2		

These results indicate that MMs A and C are more representative than mental model B. This is a reasonable indicator because these models describe the problem similarly. The description of model B presents the most significant differences with respect to the other two MMs. These differences imply less representativeness, which is confirmed in the values shown in Table XIII. It is important to notice that the example considered three models. As the number of MMs increases, the representativeness should also increase.

V. CONCLUSION

The uncertainty resulting from the increased complexity of problems emphasizes the boundaries of the decision maker rationality. Therefore, several descriptions of the phenomena of reality must be considered. This collaborative decision process presents challenges associated with the consensus among many decision makers through common knowledge identification. Thus, the shared decision making depends on the comparison of MMs from several decision-makers.

This paper proposes a method to address this linguistic dependence through an analysis that is grounded in fuzzyfication of model comparison rates. The fuzzyfication is obtained through a rule base design to analyze the elements and loops of the MMs. This work differs from other related works. Fuzzy logic and fuzzy rule base have often been used for the construction and simulation of MMs, whereas, in this paper, the fuzzy strategy was used for the composition of the model analysis method. Results showed that it is possible to use the methodology to compare MMs and that it is possible to identify more adequate MMs through the analysis of the mental model representativeness value.

Authors are working on an automatic tool which will support the following functionalities: the inclusion of MMs; comparison of MMs according to the method presented here and presentation of the representativeness values of each model. This tool should support decision makers in organizational processes. After the tool is finished, examples of larger MM bases will be executed in order to validate the method.

REFERENCES

[1] Q. J. Zhao and Z. Wen, "Integrative networks of the complex social-ecological systems," *Procedia Environmental Sciences*, vol. 13, pp. 1383-1394, 2012.  
 [2] S. Paletz, C. Schunn and K. Kim, "The interplay of conflict and analogy in multidisciplinary teams," *Cognition*, vol. 126, pp. 1-19, 2013.

[3] S. Wüstenberg, S. Greiff and J. Funke, "Complex problem solving — More than reasoning?," *Intelligence*, vol. 40, pp. 1-14, 2012.  
 [4] E. S. O'Connor, "New contributions from old sources: Recovering Barnard's science and revitalizing the Carnegie School," *European Management Journal*, vol. 31, pp. 93-103, 2013.  
 [5] K. U. Koskinen, "Problem absorption as an organizational learning mechanism in project-based companies: Process thinking perspective," *International Journal of Project Management*, vol. 30, pp. 308-316, 2012.  
 [6] R. M. Cyert and J. G. March, *A behavioral theory of the firm*, 1992.  
 [7] N. Ali, N. Chater and M. Oaksford, "The mental representation of causal conditional reasoning: Mental models or causal models," *Cognition*, vol. 119, pp. 403-418, 2011.  
 [8] Z. Wu and J. Xu, "A consistency and consensus based decision support model for group decision making with multiplicative preference relations," *Decision Support Systems*, vol. 52, pp. 757-767, 2012.  
 [9] M. Schaffernicht and S. Groesser, "A comprehensive method for comparing mental models of dynamic systems," *European Journal of Operational Research*, vol. 210, pp. 57-67, 2011.  
 [10] L. Markóczy and J. Goldberg, "A method for eliciting and comparing causal maps," *Journal of Management*, vol. 21, pp. 305-333, 1995.  
 [11] L. A. Zadeh, "Is there a need for Fuzzy Logic?," *Information Sciences*, vol. 178, pp. 2751-2779, 2008.  
 [12] D. Dubois and H. Prade, "Gradualness, uncertainty and bipolarity: Making sense of fuzzy sets," *Fuzzy Sets and Systems*, vol. 192, pp. 3-24, 2012.  
 [13] J. Salmeron, R. Vidal and A. Mena, "Ranking fuzzy cognitive map based scenarios with TOPSIS," *Expert Systems with Applications*, vol. 39, pp. 2443-2450, 2012.  
 [14] M. K. Kim, "Cross-validation study of methods and technologies to assess mental models in a complex problem solving situation," *Computers in Human Behavior*, vol. 28, pp. 703-717, 2012.  
 [15] K. Craik, *The nature of explanation*, 1943.  
 [16] P. N. Johnson-Laird, "Mental models and cognitive change," *Journal of Cognitive Psychology*, vol. 25, pp. 131-138, 2013.  
 [17] J. Forrester, *Industrial Dynamics*, Cambridge, MA: MIT Press, 1961.  
 [18] J. Mingers e L. White, "A review of the recent contribution of systems thinking to operational research and management science," *European Journal of Operational Research*, vol. 207, pp. 1147-1161, 2010.  
 [19] M. Glykas, "Fuzzy cognitive strategic maps in business process performance measurement," *Expert Systems with Applications*, vol. 40, pp. 1-14, 2013.  
 [20] B. Kosko, "Fuzzy Cognitive Maps," *International Journal of Man-Machine Studies*, vol. 24, pp. 65-75, 1986.  
 [21] J. P. Carvalho, "On the semantics and the use of fuzzy cognitive maps and dynamic cognitive maps in social sciences," *Fuzzy Sets and Systems*, vol. 214, pp. 6-19, 2013.  
 [22] E. Cosijn e P. Ingwersen, "Dimensions of Relevance," *Information Processing and Management*, vol. 36, pp. 533-550, 2000.  
 [23] M. B. Brown, "Survey article: citizen panels and the concept of representation," *Journal of Political Philosophy*, vol. 14, pp. 203-255, 2006.  
 [24] E. H. Mamdani e Assilian, "An experiment in linguistic synthesis with a fuzzy logic controller," *International Journal of Man-Machine Studies*, vol. 7, pp. 1-13, 1975.

# Software Requirement Prioritization using Machine Learning

Deepali Singh

Department of Computer Engineering and Applications  
GLA University  
Mathura, India  
deepali.panwar@gla.ac.in

Ashish Sharma

Department of Computer Engineering and Applications  
GLA University  
Mathura, India  
ashish.sharma@gla.ac.in

**Abstract**— Requirement engineering plays a very important role in software development life cycle (SDLC). Generally Software projects suffer with the problem of various types and categories of requirements and are also delimited by constraints like time and budget. To deal with this type of requirement complexity, project managers need to prioritize the requirements of the proposed software effectively. To decide about prioritization and consideration of a set of requirements is a strategic concern. This process is known as requirements prioritization. This paper proposes a novel requirements prioritization approach called Gradient Descent Ranking (GDRank), which combines project's stakeholders preferences with Functional and Non - Functional requirements, their ordering and approximations are estimated through machine learning techniques. For validation purpose the proposal is compared with various other prominent requirement prioritization methods.

**Keywords**—Quality Function Deployment (QFD), Requirements Prioritization, Machine Learning

## I. INTRODUCTION AND RELATED WORK

Fred Brooks [1] once said, “The hardest single part of building a software system is deciding what to build... No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later”. Hence, in order to develop quality and cost effective software, it is imperative to provide prioritization to customer's requirements to select the best possible set of requirements from a set of all requirements [2]. The software product quality is often determined by the capability to satisfy the necessities of the customers and users [3, 4].

Fundamentally, Functional and Non - Functional requirements are diverse in nature. The functional requirements illustrate the activities of the system as it relates to the system's functionality. It describes what a software system be supposed to do [5]. Whereas Non - Functional requirements place restriction on how the system will do so. Non - Functional requirements are habitually known as quality attributes in the field of software architecture. Quality attributes and functionality is orthogonal [6]. In this paper we study how to achieve a requirement prioritization while trying to assure both quality attributes and Functional requirements. The purpose of this paper is to present a novel idea for Requirement Prioritization using machine learning technique. We proposed a method called Gradient - Descent Ranking (GDRank). It

acquires a priority elicitation process which is very flexible. Further, a comprehensive literature survey of leading papers related to Requirement Prioritization is presented in this section and is shown in Table I. All approaches of requirement prioritization discussed in [7, 11, 13, 14, 15, 16, 17, 18, 19, 20] works on assignment of rank to requirements set for a given candidate requirement set.

TABLE I. PRIORITIZATION METHODS

Requirement Prioritization Methods	Criteria	Rank Technique
Quality Function Deployment [7]	Customer voice translated into engineer voice	Numerical Assignment
Planning Game [18]	Focus on Customer Preferences and Development Time	Numerical Assignment and Basic Ranking
Triage [14]	No. of Criteria e.g. Business Target, Development Time	Numerical Assignment and Basic Ranking
Fairness Analysis [17]	Focus on Stakeholders Objectives and Goals	Pareto Optimal Search Based Software Engineering
Planusage [16]	Focus on Stakeholders Objectives and Target	Basic Ranking
Cost-Value Approach [15]	Based on Significance for the customer and Development expenditure	AHP Approach
Win-Win Approach [13]	No. of Criteria e.g. Business Value Development Efforts	AHP Approach
AHP [11]	Pair-wise Evaluation and Hierarchies of Criteria	AHP Approach
Wiggers Method [19]	Based on Implementation Cost and Risk, and Value for the customer	Numerical Assignment and Method Rules
CBRank [20]	Based on Domain Adaptive	RankBoost, Machine Learning Technique
GDRank	Based on Domain Adaptive	Ranknet, Machine Learning Technique

## II. PROPOSED METHOD

The proposed GDRank method sets on an outline first commence in [12] that prop up in making decisions for ordering a set of items, like software requirements or product features. An iterative prioritization method is proposed by this

framework that can handle different ordering criteria as well as decisions by individual or group of decision makers (domain experts).

### A. Concept

We take a finite collection of Functional Requirements  $FReq = \{f_1^+, \dots, f_n^+\}$  and Non - Functional Requirements  $NReq = \{f_1^-, \dots, f_m^-\}$  that is to be ranked and for that we describe the Universe of a set of requirement pairs  $U = \{(f_i^+, f_j^-); i < j\}$ . We entitle the organized relation among two requirements that can be obtained from stakeholder priority. It is then formally define in terms of the function  $\theta(f_i^+, f_j^-)$ , where  $\theta: U \rightarrow \{-1, 0, 1\}$ , with the following denotation for its assessments:  $\theta(f_i^+, f_j^-)$  is equal to -1, if  $f_j^- < f_i^+$ ; is equal to 1, if  $f_i^+ < f_j^-$  and is equal to 0, when there is no preference oof order between  $f_i^+$  and  $f_j^-$ .

An Unordered Requirements couple is a couple of requirements  $(f_i^+, f_j^-)$  for which priority is not known yet as the domain expert has not assigned a priority yet.

### B. The Process of Prioritization

The GDRank prioritization procedure infuses human behavior with machine calculation. The procedure is drafted in Fig. 4, where five footsteps are characterized as rectangles. The common artifacts in effort and production are: the set of Requirements (FReq and NReq), the set of Ranking Functions (F), the Priorities by experts ( $\mathcal{E}_T$ ), encoding the requirement characteristics, and the Approximated Rank ( $H_T$ ) where  $T$  corresponds to the last process iteration or the Final Approximated Rank (H).

The process is based on five steps which are as follows:

1. Requirement Elicitation. We consider requirement Elicitation by Quality Function Deployment (QFD) approach. Translation of subjective quality criteria into quantifiable and measurable objectives that can be used in designing & manufacturing the product is the main Goal of QFD. In this “The voice of the customer translated into the voice of the engineer [7].” QFD procedure includes putting together a "House of Quality"[8] like one shown in Fig. 1, which is for the advancement of a climbing harness [9].
2. Balancing Functional and Non - Functional Requirement. Balancing of Functional and Non-Functional Requirements is done with the help of Pattern Driven Architectural Partitioning (PDAP) [10]. Generally, choices over architecture that satisfy a functional requirement may clash with a quality attribute, or reverse. Such sides effects ought to be take care by product architect and proper balancing decisions should be made. Architectural pattern helps us to avoid such problem to a certain level.
3. Pair Sampling. A set of Sampled Requirements Pairs is selected from the set of Requirements (Functional and Non – Functional) whose relative preference is unknown. Pair Sampling is done by the help of Analytical Hierarchy Process (AHP) [11].

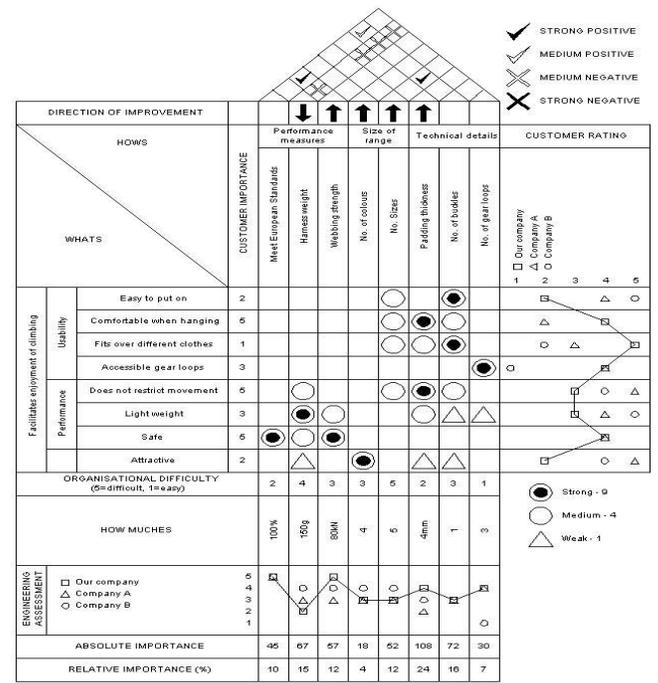


Figure 1. House of Quality defined by QFD [9].

Step1. Place n Functional Requirements in the rows and m Non-Functional Requirements in columns of an n x m matrix. We will assume four Functional Requirements: FReq1 to FReq4, and four Non-Functional Requirements: NReq1 to NReq4.

Step2. Carry out pair wise comparisons of all the Functional and Non-Functional Requirements. For this purpose the primary scale used is shown in Table II. Insert determined relative intensity value for each set of pair of requirements in the position (FReq1, NReq2) where the row of FReq1 meets the column of NReq2. Insert the reciprocal value in position (FReq2, NReq1) and insert “1” in all positions in the main diagonal. Thus, it look like as shown in Fig. 2:

	NReq1	NReq2	NReq3	NReq4
FReq1	1	1/3	2	4
FReq2	3	1	5	3
FReq3	1/2	1/5	1	1/3
FReq4	1/4	1/3	3	1

Figure 2. Requirement in Matrix form.

Step 3. To represent the criterion distribution, guess the eigenvalues of the matrix with the help of performing averaging over normalized columns as shown in Fig. 3:

	NReq1	NReq2	NReq3	NReq4	Sum
FReq1	0.21	0.18	0.18	0.48	1.05
FReq2	0.63	0.54	0.45	0.36	1.98
FReq3	0.11	0.11	0.09	0.04	0.34
FReq4	0.05	0.18	0.27	0.12	0.62

Figure 3. Averaging over normalized column.

Afterward divide each row sum with the number of requirements to normalize the sum of the rows. This calculation results in estimation of Eigen values of matrix and is referred to as the priority matrix.

$$\frac{1}{4} \begin{pmatrix} 1.05 \\ 1.98 \\ 0.34 \\ 0.62 \end{pmatrix} = \begin{pmatrix} 0.26 \\ 0.50 \\ 0.09 \\ 0.16 \end{pmatrix}$$

Step 4. Based on the estimated Eigen values, each requirement gets its relative value.

4. Priority Elicitation. The input is taken by the Pair Sampling step as a collection of Sampled Requirements Pairs produced and produces output on the basis of the Priorities stated by a domain expert (or decision makers) as a set of Ordered Requirements Pairs.
5. Priority Learning. Given a set of Ranking Function and partial elicited stakeholder priority, an approximation rank of the known preferences is produced by the learning algorithm and then the Final corresponding Approximated Rank for the requirement.

The output gives an estimate of the exact ranking, known as Approximated Rank, and may turn out to be the input for a further iteration of the process. The iteration stops when the result of the learning step is regarded as accurate, and the output is considered as the Final Approximated Rank.

TABLE II. SCALE FOR PAIRWISE COMPARISONS

SCALE FOR PAIRWISE COMPARISONS		
Relative intensity	Definition	Explanation
1	Of equal value	Two requirements are of equal value
3	Slightly more value	Experience slightly favors one requirement over another
5	Essential or strong value	Experience strongly favors one requirement over another
7	Very strong value	A requirement is strongly favored and its dominance is demonstrated in practice
9	Extreme value	The evidence favoring one over another is of the highest possible order of affirmation
2, 4, 6, 8	Intermediate values between two adjacent judgments	When compromise is needed
Reciprocals	If requirement i has one of the above numbers assigned to it when compared with requirement j, then j has the reciprocal value when compared with i.	

### C. The Priority Learning Technique

This step yields an approximation of a preference structure, adapting the Ranknet approach described in [12]. The Ranknet method is capable of producing more accurate rank forecast. Next, we provide an instinctive description of the Ranknet approach by drawing the Ranknet algorithm, displayed as pseudo code in Algorithm 1.

Algorithm 1. A sketch of the Ranknet algorithm  
Input:

$$\text{Req}' = \{r_1, \dots, r_n\}$$

The set of elicited Requirements

$$F = \{f_1, \dots, f_k\}$$

Partial orders defining constraints and priorities upon Req'

$$\mathcal{E} = \{(r_i, r_j); i < j \mid \theta(f_i^+, f_j^-) \neq 0\}$$

A subsample of obtained pair-wise

preferences

Output:

$$H(r) \text{ (H: Req}' \rightarrow \mathbb{R})$$

A ranking function described upon Req'

Begin

1.  $X = \text{initialize}(\mathcal{E})$   
Weighting of obtained pairs  $\mathcal{E}$
2.  $T = \text{Maximum Number Of Cycles}$   
Number of learning cycles
3. For  $i = 1$  to  $T$
4.  $w_i = \text{learn } w_i$  that minimizes the square error

$$E[\hat{w}] \equiv \frac{1}{2} \sum_{d \in D} (t - o)^2$$

Where  $w$  is weight,  $E$  is error value,  $o$  is the output of the linear unit and  $t$  is the training example

5. Calculate the gradient of  $E$  with respect to the vector  $\{w_0, \dots, w_n\}$

$$\nabla E[\vec{w}] \equiv \left[ \frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n} \right]$$

6.  $w = w + \Delta w$

Where

$$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

Here  $\eta$  is the learning rate which is positive constant, which find out the step size in the gradient descent search.

7. End For
8.  $H(r) = \text{synthesis of ranking function}$   
 $\sum_{i=0}^{k-1} W_i \cdot x_i$
9. return  $H(r)$

End

The GDRank algorithm performs  $T$  cycles in which the input is the set of ranking function  $F$  and the output is Final Approximated Rank.

## III. RESULTS

The Result is generated set of requirements, which are obtained on the basis of graph plotted between the no. of requirements and the accuracy in rank shown in Fig.5. In GDRank approach, exact requirements are elicited with the help of QFD approach. We consider the functional and Non-functional requirements of the obtained requirements and balance them. Therefore the ranks which are learned from this approach are more accurate as compared to the previous two approaches. Finally, in Fig. 6 the graph which is plotted between the no. of requirements and the error rate, describes the gap between the previous rank and the current rank.

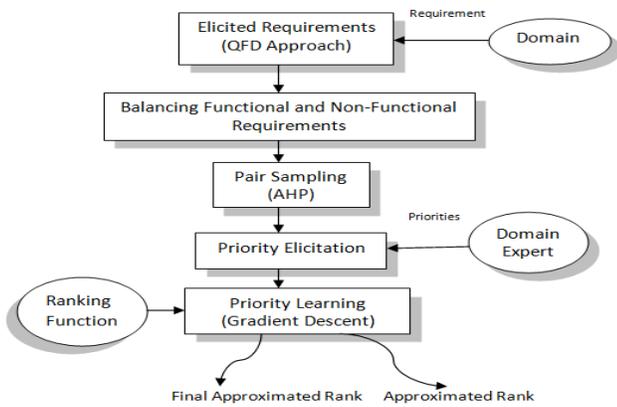


Figure 4. Basic steps of the Requirements Prioritization in GDRank.

Through this we can see the learning rate and the accuracy in results.

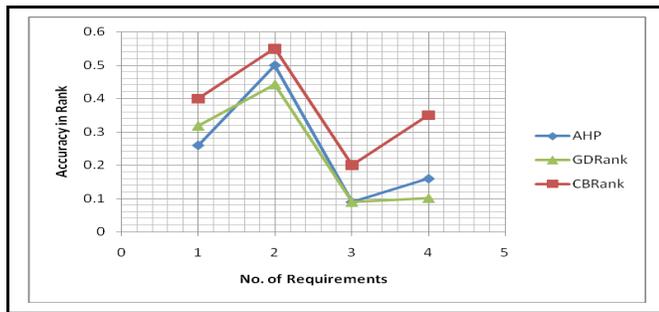


Figure 5. The plot of the no. of requirements and the accuracy in rank.

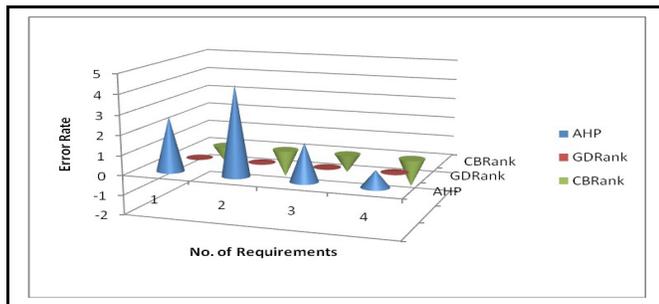


Figure 6. The plot of the no. of requirement and the error rate.

#### IV. CONCLUSION AND FUTURE WORK

In this paper, we present a complement description of the proposed GDRank procedure for requirements prioritization. The GDRank is used to learn the ranks of the ordered sets of requirements which provide the optimal error rates. The GDRank procedure has been sited with respect to the other methods which are available for requirements prioritizations, with particular reference to AHP approach and CBRank approach. Some assumptions are made in this paper that the

requirements are well elicited and no requirements are further going to be added. The potential advantage of GDRank method is the ranking with respect to Functional and Non-Functional Requirements. But on the other hand, as the set of requirements increases, then the efforts needed by the human evaluators when pair preferences are obtained grows rapidly. In future, the issues like requirement dependencies, renewing requirements rank when new requirements are included and when the view of the expert changes and Scalability must be further considered.

#### REFERENCES

- [1] F. P. Brooks, "The Mythical Man-Month"- Essays on Software Engineering, Addison-Wesley Longman, Boston, MA, USA, 1995.
- [2] P. Berander, "Prioritization of Stakeholder Needs in Software Engineering, Understanding and Evaluation," Blekinge Institute of Technology, ISBN: 91-7295-052-8. Licentiate Series No 2004:12, 2004.
- [3] B. Bergman and B. Klefsjö, "Quality - From Customer Needs to Customer Satisfaction," Studentlitteratur AB, Lund, Sweden, 2003.
- [4] G. G. Schulmeyer and J. I. McManus, "Handbook of Software Quality Assurance," 3rd Edition, Prentice Hall, Upper Saddle River, NJ, 1999.
- [5] A. Aurum. and C. Wohlin, "The Fundamental Nature of Requirements Engineering Activities as a Decision-making Process," Information and Software Technology, 45(14), pp. 945-954, 2003.
- [6] Pfleeger, Altee, and S. L., "J. M. Software Engineering: Theory and Practice," 3rd edition, Pearson Prentice Hall, 2006.
- [7] I. Sommerville, "Software engineering," (8th ed.) Addison Wesley Longman Publishing Co., Inc., 2007.
- [8] L. Bass, P. Clements and R. Kazman, "Software Architecture in Practice," 2nd Edition, Addison Wesley, 2003.
- [9] Hauser, J. R. and D. Clausing, "The House of Quality," The Harvard Business Review, May-June, No. 3, pp. 63-73, 1998.
- [10] Lowe, A. J. and Ridgway, "Quality Function Deployment," University of Sheffield, <http://www.shef.ac.uk/~ibberson/qfd.html> , 2001.
- [11] T.L. Saaty, "Fundamentals of the Analytic Hierarchy Process," RWS Publications, 1994.
- [12] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender, "Learning to Rank using Gradient Descent," Proceedings of the 22 nd International Conference on Machine Learning, Bonn, Germany, 2005.
- [13] G. Ruhe, A. Eberlein, and D. Pfahl, "Quantitative Winwin: A New Method for Decision Support in Requirements Negotiation," Proc. 14th Int'l Conf. Software Eng. and Knowledge Eng., pp. 159-166, 2002.
- [14] A. Davis, "Just Enough Requirements Management: Where Software Development Meets Marketing," Dorset House, 2005.
- [15] J. Karlsson and K. Ryan, "A Cost-Value Approach for Prioritizing Requirements," IEEE Software, vol. 14, no. 5, pp. 67-74, Sept./Oct. 1997.
- [16] T. Gilb, "Competitive Engineering: A Handbook for Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage," Elsevier, 2005.
- [17] A. Finkelstein, M. Harman, S.A. Mansouri, J. Ren, and Y. Zhang, "A Search Based Approach to Fairness Analysis in Requirement Assignments to Aid Negotiation, Mediation and Decision Making," Requirements Eng., vol. 14, no. 4, pp. 231-245, 2009.
- [18] K. Beck, "Extreme Programming Explained," Addison-Wesley, 1999.
- [19] K.E. Wiegers, "Software Requirements. Best Practices," Microsoft Press, 1999.
- [20] A. Perini, A. Susi, P. Avesani, "A Machine Learning Approach to Software Requirements Prioritization," IEEE Transactions On Software Engineering, Vol. 39, No. 4, April 2013

# Detecting Anomaly in the Usage of Database Attribute

Kaiping Liu, Hee Beng Kuan Tan, Arnatovich Yauhen Leanidavich  
School of Electrical and Electronic Engineering, Nanyang Technological University  
Block S2, Nanyang Avenue, Singapore  
{kpliu, ibktan}@ntu.edu.sg, YAUHEN001@e.ntu.edu.sg

**Abstract**— In database applications, database operations should be provided to maintain persistency of the data which is often represented as database attributes. Any missing, redundant or inconsistent operation performed on database attributes would indicate anomaly or even program bugs. Through characterizing operations performed in database transactions on database attributes, we extract a feature vector from code for each attribute. This paper proposes a clustering-based approach which analyzes the feature vectors to automatically detect anomalies in the usage of database attributes. Once an anomaly is detected, developers can perform investigation to take corrective actions if necessary. The evaluations on both industrial and open source database applications show that our approach is able to detect many types of anomalies in the usage of database attributes with a high detection rate (92.8% on average), and a low false positive rate (0.57% on average).

**Keywords**—*anomaly detection; database application; clustering; attribute usage.*

## I. INTRODUCTION

Database is a major component of many software systems. In database applications, adequate operations should be provided to maintain the persistency of data. Any missing, redundant or inconsistent operation performed on database attributes would indicate anomaly of a database application.

Anomalies are instances that cannot be classified under any normal behavior. The PHP code snippet shown in Fig. 1, is an example of anomaly in database operations.

In Fig.1 line 6, the attribute “family\_name” is selected in order to update the attribute “family\_name” in table tb\_user\_info in line 17. However, from the database schema, we found that the table “tb\_user\_info” requires non-null value for the attribute “family\_name” but the table “tb\_user\_account” does not have such requirement. Furthermore, from investigation of the code, there is no program define the value of the attribute “family\_name”. So the result of the SELECT operation contains a null value. Hence, the execution of the UPDATE operation would cause exception. It is desirable to detect such kind of anomaly. Our work aims to address this issue.

This paper proposes an approach to detect anomalies in the use of database attributes by means of abstracting and characterizing database operations. We present a clustering-

based anomaly detection algorithm, which takes as inputs a set of unlabeled database attributes and finds anomalies within them. We make two general assumptions: 1) the number of normal attributes in the training set greatly exceeds the number of anomalous attributes, thus implying that the normal attributes should form larger clusters compared to the anomalous attributes; 2) attributes having the same type of anomaly should be close to each other in feature space under some reasonable metric, while attributes with different type of anomaly will be far apart. The above assumptions mean that any anomalous attribute would appear to be an outlier in the dataset due to its rarity and abnormality.

```
1. <?php
2. function do_query($query)
3. {
4.     return mysql_query($query);
5. }
6. $sql = "SELECT family_name as fn FROM
tb_user_account WHERE user_id=";
7. if (isset($_POST['USER_ID']))
8. {
9.     $sql .= $_GET['USER_ID'];
10. }
11. else
12. {
13.     $sql .= '0';
14. }
15. $result = do_query($sql);
16. $data = mysql_fetch_array($result);
17.     $sql = "UPDATE tb_user_info SET
family_name=" . $data['fn'];
18. do_query($sql);
19. ?>
```

Figure1. PHP code example

Our approach groups the database attributes together into clusters using a distance-based metric. Once the database attributes are clustered, we identify small clusters and label them as anomalous clusters. Cross-project validations are performed on three industrial database applications and seven open source database applications to verify the approach. The results show that our approach is able to detect many types of anomalies with an average detection rate 93.0%, while maintaining a low false positive rate.

The paper is organized as follows. Section II introduces the extraction of the feature vector for attribute usage. Section III describes our approach. Section IV evaluates the proposed approach and reports the experiment result. Section V

discusses the performance of our approach. Section VI discusses the related work. Section VII concludes the paper.

## II. CHARACTERIZING DATABASE OPERATIONS USING FEATURE VECTORS

### A. Database Operations

Since transaction is the atomic processing unit in a database application, we characterize the database operations performed on database attributes on a per-transaction basis. We classify the database operations into the following types:

- Create (C): A value of an attribute is inserted.
- Null Create (NC): A record that contains the attribute is inserted without defining the value of the attribute.
- Control Update (COU): The value of an attribute is updated by a new value that is not influenced by the existing attribute value and inputs from user and database.
- Overriding Update (OVU): The value of an attribute is updated to a new value that is influenced by user input but is not influenced by the existing value of the attribute.
- Cumulating Update (CMU): The value of an attribute is updated to a new value that is influenced by the existing attribute value.
- Delete (D): The value of an attribute is deleted as a result of deletion of a record containing it.
- Use (U): The value of an attribute is used to support the insertion, updating or deletion of other database attributes or output to the external environment. It is not considered under this case if the value of an attribute is used to update itself.
- Other Update (OU): If there are multiple database operations performed on a database attribute in one transaction, the database operation is classified as Other Update.

We characterize the usage of a database attribute using an eight-element Boolean vector  $[m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8]$ , where  $m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8$  denote the existence of a transaction in which an operation performed on the attribute is of type C, NC, COU, OVU, CMU, D, U or OU respectively. This vector is called feature vector for attribute usage.

### B. Extraction of Feature Vector for Attribute Usage

To automatically extract the database operations performed on each attribute in a transaction, static program analysis is performed on each path in a database application. Since PHP is a widely used scripting language, This paper uses PHP programs as subjects for demonstrations and experiments. However, the approach can be applied to any database application as it is based on basic control and data dependency analysis with regard to standard SQL statements.

#### 1) Query extraction

In a database application, the SQL queries are commonly formed by dynamically concatenating several string literals and

variables. From the inter-procedural control flow graph of the example code in Fig. 1, two paths can be found that perform database transactions, and both of which start from the MAIN function's entry point, and reach the SQL execution function "do\_query". The actual queries executed in these two transactions are different.

We follow the technique used in structural program testing by generating a set of basis paths using the baseline method proposed in [1]. Specifically, we traverse the loop body only once. For each path in the basis set, whenever we encounter a query execution function like "mysql\_query", the definition of every part of its parameter which is a query string is retrieved and concatenated in order. As an example, for the code in Fig. 1, the extracted queries of the SELECT operation would be: 1) "SELECT family\_name as fn FROM tb\_user\_account WHERE user\_id='\_\_'"; 2) "SELECT family\_name as fn FROM tb\_user\_account WHERE user\_id=0".

#### 2) Extraction of Feature Vector

After the queries are extracted, we analyse each query to obtain the characteristics of database operations using an SQL grammar parser. All the CREATE TABLE queries are first parsed. Then, we analyse the queries according to their operation types as follows:

- SELECT: The SELECT query is parsed, table aliases are identified and restored by the actual table names, and the attributes are identified. The attribute names are extracted from the select list, JOIN expressions as well as the WHERE clause. The star-shorthand "\*" is regarded as referencing all attributes.
- INSERT: After parsing, the table name is identified. We then check whether there is column list in this query. When no column list is provided, it is assumed that values of all the attributes in this table are to be inserted. Those attributes declared in the schema as "auto incremental" or having not-null default values are also viewed as inserted by the query.
- UPDATE: We not only collect the attribute names that are updated in one query, but also identify the update pattern for each attribute. We analyze the value string to determine the update type, i.e. either COU, OVU or CMU for the attribute. For example, the assignment expression `name=$_POST['username']` is an example of OVU, whereas the expression `balance=balance-$withdraw` is an example of CMU.
- DELETE: We identify the table name, and mark all the attributes of this table as "Delete".

Besides, the attributes in the WHERE clause are characterized as "Use". If an attribute has multiple operations in this transaction it will be marked as OU in this transaction; otherwise, it will be marked as one of the other seven features as described in Section II.A. After all transactions have been analyzed, the characteristics of database operations performed on each attribute are merged to generate its feature vector. For example, if an attribute experiences the type "Create" for at least one time, the first element of the feature vector would be 1; otherwise, it is 0.

### III. DETECTING ANOMALIES IN THE USAGE OF DATABASE ATTRIBUTES

The extracted feature vectors are filtered and passed to the clustering algorithm. We use the training data to estimate the optimal parameters values. Based on the clustering result and optimal parameters values, normal and anomalous clusters are identified. Then the model can be used to detect anomalies in a new database application. The overview of the approach can be seen in Fig. 2.

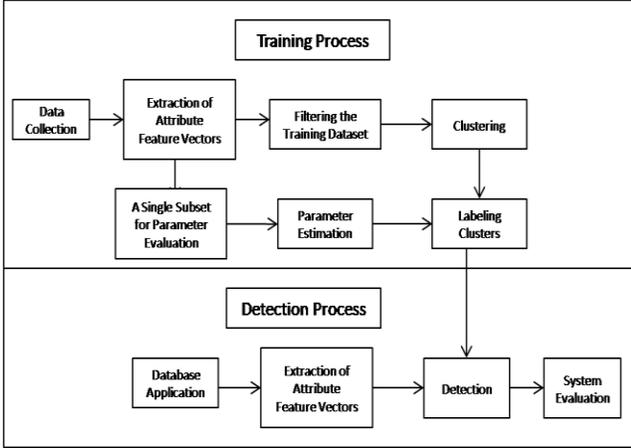


Figure 2. Overview of the proposed approach

#### A. Clustering Method

The extracted feature vectors have a relatively low dimensionality and the elements of the vector are binary value. Hence, we adopt a simple variant of single-linkage clustering method which proceeds as follows: given a fixed metric  $M$  and a constant cluster width  $W$ , the distance between  $C$ 's defining instance  $C$  and  $d$  can be computed as Euclidean  $dist(C, d)$ . Here  $C$  is the feature vector that defines the centroid of that cluster. If the distance is within  $W$ , the instance  $d$  will be assigned to the nearest cluster. Otherwise, a new cluster would be generated and the attribute feature vector would become its centroid. The Euclidean distance used to measure the distance between a cluster  $C$  and an instance  $d$  is defined as:

$$Dist(C, d) = \sqrt{\sum_{i=1}^n (c_i - d_i)^2}$$

where  $n$  is the dimension of the vectors, and  $c_i$  is the element of the centroid of  $C$  while  $d_i$  is the element of vector  $d$ .

#### B. Labeling Clusters

After clustering, it remains unknown which clusters consist of normal attributes and which ones contain anomalies because the attributes are unlabelled during clustering. In our work, we assume that an overwhelming majority of the training dataset is made up of normal attributes and small clusters are likely to consist of anomalies. Hence, we label the top  $P\%$  largest clusters as 'normal', and the remaining clusters as 'anomalous'.

#### C. Detection of Anomalies

Once the clusters are created and labelled, they can be used to detect anomalies in the use of database attributes. Given a new database application, we can extract the feature vector for

each attribute. For each feature vector  $v$ , we find the cluster that includes  $v$  and assign this attribute with that cluster's label. If  $v$  does not belong to any clusters discovered thus far, a new cluster would be generated and  $v$  would become its centroid. We will manually examine  $v$  and decide it is anomalous or normal.

### IV. EVALUATION AND RESULTS

#### A. Training Dataset Description

In our research, it was extremely difficult to find readily available dataset. Hence, we selected three matured, large-scale industrial database applications and seven open source database applications from sourceforge.net. The three selected industrial database applications come from different domains, including a membership management system, a school management system and a web-based e-commerce system. The open source systems include ChurchInfo, Front Accounting, CourseMS, Gliding Booking System, CiteCRM, AlumniServer and Hotel Booking Portal. We obtained a dataset with 8909 source code files, 1393K lines of code and 3302 database attributes in total. All the experiments were conducted on a desktop PC with an Intel Core Duo 2.4GHZ CPU and 4GB memory.

For data collection, we have built our extraction tool on top of PHC [2], an open source PHP compiler. Our tool performs data flow analysis on the paths of the inter-procedural CFG for each system, and extracts all the database transactions from them. Feature vectors for all the attributes are then formed.

#### B. Performance Measurement

Two measures were computed over all labelled attributes to access the performance of our approach.

- Detection rate = the number of anomalous attributes detected by the system / the total number of anomalous attributes presented in the testing dataset;
- False positive rate = the total number of normal attributes that were wrongly classified as anomalous / the total number of normal attributes.

To calculate these values, access to labels of attributes in the dataset was required. Based on the database schema, we extracted the attributes in the database. After analysing the program code and based on the attributes extracted, we can identify whether proper and adequate operations have been conducted on each attribute and then labelled each attribute with "normal" or "anomalous". We calculated these two measures over all labelled attributes.

#### C. Filtering the Training Dataset

We make use of 30% of the dataset (990 attributes) as our training dataset. Our first assumption states that the number of normal attributes greatly exceeds that of the anomalous attributes. Hence, in the resulting training dataset, the number of normal attributes (96.36%) greatly exceeds that of the anomalous attributes (3.64%).

#### D. Fixing Parameters

It is necessary to fix and optimize the values of two parameters first: the cluster width  $W$  and the percentage  $P$ .  $W$

determines the minimum distance between two attribute feature vectors assigned to the same cluster, while the percentage  $P$  decides the top  $P\%$  largest clusters that are labelled as 'normal'.

A series of tests was conducted on the training dataset for using a range of values of the two variables  $W$  and  $P$ . Table I shows the results for optimizing the values of  $W$ , and their corresponding measured performances. As we use the Euclidean distance measure, the cluster width  $W$  is set to a value slightly below each Euclidean distance value (such as  $\sqrt{1}, \sqrt{2}, \dots, \sqrt{8}$ ).

TABLE I. STATISTICS FOR EVALUATING  $W$

$W$	$P$	DR	FPR	$W$	$P$	DR	FPR
0.7	60%	94.4%	0.419%	2.2	60%	58.3%	1.992%
1.4	60%	88.9%	1.048%	2.4	60%	47.2%	4.193%
1.7	60%	80.6%	1.468%	2.6	60%	38.9%	5.241%
1.9	60%	69.4%	1.572%	2.8	60%	33.3%	6.289%

(DR=Detection Rate; FPR=False Positive Rate)

We decided to use  $W = 0.7$  in subsequent tests, since it produced a low false positive rate and high detection rate. To find the value for  $P$ , we conducted several tests on the same dataset. The results of some of the tests are shown in Table II.

TABLE II. STATISTICS FOR EVALUATING  $P$

$W$	$P$	DR	FPR	$W$	$P$	DR	FPR
0.7	80%	72.2%	0.314%	0.7	40%	97.2%	13.732%
0.7	70%	80.6%	0.419%	0.7	30%	97.2%	24.738%
0.7	60%	94.4%	0.524%	0.7	20%	100%	33.962%
0.7	50%	94.4%	6.499%	0.7	10%	100%	50.524%

From table II, it can be seen that the detection rates are the same when  $P = 50\%$  or  $P = 60\%$ , though the false positive rate is fairly lower when  $P = 60\%$ . Therefore,  $P$  of 60% was chosen.

The clustering results of the training dataset when  $P=60\%$ ,  $W=0.7$  are shown in Table III. There are 32 clusters in total. Based on the value of  $P$ , we labelled the first 19 clusters as normal and labelled the rest as anomalous. After further investigate of the anomalous attributes, we classified them into the following four types:

- Missing database operations (MI): For a database attribute, essential database operations (e.g., inserting a value of an attribute) are missing.
- Inconsistent database operations (IC): It is essential to provide an transaction in a database application to correct the effect of a transaction that has been executed with erroneous input. For a transaction that updates an attribute through cumulative update, the correction should also be made through cumulative update for control purpose. Correcting the result of a transaction that updates an attribute by "cumulative update" using "overriding update" is a common inconsistency fault.

- Redundant database operations (RD): For a database attribute, additional different types of database operations are performed on it.
- No Update (NU): For a database attribute, the program does not provide any operation to maintain or use it.

It can be seen from table IV that there are a total of 956 attributes (96.57%) labeled as "normal" (NM) and 34 attributes (3.43%) labeled as "anomalous" (MI, IC, RD or NU).

### E. Cross Validation Testing

Based on the parameters and clusters learned from the training process, we performed cross-project evaluation by using a variant of the cross validation method.

We formed five subsets from the entire dataset, each containing approximately 660 attributes. It should be noted that no information about the attributes' label was used during the clustering in the training process, therefore, the training set still can be used for testing. Since we require that the number of anomalous attributes should constitute a very small portion of the training dataset, it was found that there were two subsets failed to meet this requirement. Therefore for cross validation training only three of the five subsets are selected. The numbers of normal and anomalous attributes of each of the three subsets are shown in Table IV.

Each time, one of the three subsets was chosen as the training dataset to perform the clustering process. Then the clusters are labeled and used to test each of the other two subsets. In this manner, the testing is conducted 6 times in total. The results are shown in Table V.

Our evaluations show that the proposed approach can achieve a high detection rate with a low false positive. We believe that this technique is capable of detecting the anomalies in attribute usage in database applications.

### F. Some Examples

The attribute "subject" in the table "as\_newsletter" in AlumniServer is predicted as missing function. Which means this attribute is neither defined nor updated. However, it is referenced and is declared as "default null" in the database schema. This could be a potential fault because null value frequently causes unexpected result or failure of the system.

In the prediction results of ChurchInfo, we found several no update attributes in table "user\_usr". This table is used to store the information of users. There are eight attributes named "usr\_CalNoSchool1", "usr\_CalNoSchool2", ... , "usr\_CalNoSchool8" in this table. However, only the attribute "usr\_CalNoSchool1" among the eight has UPDATE operation. It is reasonable to argue that this could be inappropriate. After carefully searching the system, we spotted the following code snippet in the source code file Default.php, which updates the table "user\_usr":

TABLE III. CLUSTERING RESULTS OF THE TRAINING DATASET

Cluster	Center of clusters	# of attribute	Type	Cluster	Center of clusters	# of attribute	Type
1	<1,0,0,1,0,1,1,0>	113	NM	17	<1,0,1,0,1,0,1,0>	26	NM
2	<1,1,0,1,0,1,1,0>	94	NM	18	<1,1,1,1,0,0,1,0>	25	NM
3	<1,0,1,0,0,1,1,0>	82	NM	19	<1,0,0,0,1,0,1,0>	23	NM
4	<1,0,1,1,0,1,1,0>	76	NM	20	<1,0,0,1,0,1,0,1>	5	RD
5	<1,1,1,0,0,1,1,0>	71	NM	21	<1,0,0,0,0,1,1,0>	4	NU
6	<1,0,1,0,1,1,1,0>	65	NM	22	<1,0,0,1,1,0,1,0>	4	IC
7	<1,1,0,0,1,1,1,0>	56	NM	23	<0,1,0,1,1,1,0,0>	3	IC
8	<1,1,1,1,0,1,1,0>	48	NM	24	<0,1,0,0,0,0,1,0>	3	NU
9	<1,1,1,0,0,0,1,0>	42	NM	25	<1,0,0,1,1,1,1,0>	3	IC
10	<1,1,1,0,1,1,1,0>	41	NM	26	<0,0,0,0,0,1,0,0>	2	MI
11	<1,1,1,0,1,0,1,0>	39	NM	27	<0,0,0,0,0,0,0,0>	2	MI
12	<1,0,0,1,1,0,1,0>	37	NM	28	<0,0,1,0,1,1,1,0>	2	MI
13	<1,0,0,1,1,1,1,0>	32	NM	29	<0,0,1,0,0,1,1,0>	2	MI
14	<1,0,1,1,0,0,1,0>	30	NM	30	<0,0,0,1,0,0,1,0>	2	MI
15	<1,1,1,1,0,1,0,0>	28	NM	31	<0,0,0,0,0,1,1,1>	1	MI
16	<1,1,0,0,1,0,1,0>	28	NM	32	<0,1,0,0,0,0,1,0>	1	NU
Total number of attributes: 990							

TABLE IV. STATISTICS OF THE THREE SUBSETS

Training Set	#Normal attr	#Anomalous attr
T1	641	19
T2	637	23
T3	635	25

TABLE V. PERFORMANCE OF THE SYSTEM UNDER VARIOUS TRAINING AND TESTING DATASET COMBINATIONS

Training dataset	Testing dataset	Detection Rate	False Positive Rate
T1	T2	91.3% (21/23)	0.63% (4/637)
T1	T3	84.0% (21/25)	0.78% (5/635)
T2	T1	89.5% (17/19)	0.62% (4/641)
T2	T3	92.0% (23/25)	0.63% (4/635)
T3	T1	100.0% (19/19)	0.47% (3/641)
T3	T2	100.0% (23/23)	0.31% (2/637)

```

$ssSQL = "UPDATE user_usr SET .....
if ($_SESSION['dCalNoSchool1'] != '')
    $ssSQL .= ", usr_CalNoSchool1 = '". ...;
if ($_SESSION['dCalNoSchool2'] != '')
    //!wrong attribute name starting from here
    $ssSQL .= ", usr_CalNoSchool1 = '". ...;
if ($_SESSION['dCalNoSchool3'] != '')
    $ssSQL .= ", usr_CalNoSchool1 = '". ...;
.....
if ($_SESSION['dCalNoSchool7'] != '')
    $ssSQL .= ", usr_CalNoSchool1 = '". ...;
if ($_SESSION['dCalNoSchool8'] != '')
    $ssSQL .= ", usr_CalNoSchool1 = '". ...;

```

After investigation, we found that the developer in fact intended to update all these eight attributes according to some conditions. However, all the strings in the concatenation were

likely mistyped as “usr\_CalNoSchool1”, resulting in no update operation for the other seven attributes. Aided by the prediction results, developers or maintainers can take corresponding actions to deal with this.

G. Threats to Validity

It can be argued that the values of the parameters we evaluated may depend on the domain. Therefore, to vindicate our approach, we have chosen database systems of different sizes and complexities from different domains. Ideally, we would like both of our assumptions mentioned in Section I to be satisfied. In reality, of course this assumption may not be fully satisfied and this is also one of the primary reasons that our method fails to detect 100% of the anomalous. However, we still believe that the proposed approach can be applied to a variety of database applications, and the best way to prove our conclusions is to replicate and extend our experiments.

V. RELATED WORK

Data mining methods have been well utilized in software engineering especially intrusion detection. Portnoy et al. [3] proposed a clustering method to detect intrusion by regarding intrusion as anomaly against normal behaviors. Eskin, et al. [4] investigated the effectiveness of three algorithms in intrusion detection. Furthermore, Eskin et al. [5] also applied machine learning method to learn a mixture probability distribution in order to model the anomalies for intrusion detection. Oldmeadow et al. [6] carried out further research based on the clustering methods mentioned in [4] and showed improvements in accuracy.

Detecting anomalies in data has been studied in the statistics community as early as the 19th century. Chandola et al. [7] provided a comprehensive survey on the anomaly detection techniques. Our work is closely related to partitioning method which constructs *k* partitions of a database of *n* objects where each partition represents a cluster. Based on the

assumption that normal data instances belong to large and dense clusters, while anomalies either belong to small or sparse clusters, several approaches have been proposed [8-11]. All these studies show that most types of anomalies in programs do exhibit certain characteristics, and therefore by mining these characteristics, we can find the anomalies effectively. Our work also shows that anomalies in database operations can be discovered through data mining methods.

Besides the above mentioned methods, other methods have also been proposed for anomaly detection. For example, Xie et al. [12] proposed to use hidden semi-Markov model for the purpose of network DDOS detection. Vigna et al. [13] proposed to detect web-based attacks by combining analyses on HTTP request and SQL queries. Zhang et al. [14] computed the anomaly score of a data instance as the sum of its distances from its  $k$  nearest neighbors. Their method differs from ours in that they only focus on the syntactic level of the queries but not system semantic level. Our work shows that anomalies in database operations can be discovered through data mining methods.

Over the years, a lot of work has been done to investigate how to automatically detect faults [15-18]. In their methods, they adopted both the supervised and unsupervised pattern classification and multivariate visualization. Different from the above-mentioned approaches, our approach is to mine anomalies in database attribute usage from source code using unlabelled feature vectors. Our approach is able to efficiently detect anomalies in database applications. We have applied the proposed approach to ten large-scale database applications. These applications contain around 8900 source files and more than 1390KLOC. Hence, we believe our approach can be applied to large-scale database applications. Furthermore, our approach can be used in the verification and validation of database applications. To date, we believe that no such approach has been proposed before. In opposing to our earlier work on data lifecycle [19], this paper applies data mining techniques to systematically detect anomalies in database attribute usage, instead of using heuristic rules.

## VI. CONCLUSION

In this paper, we have presented an approach for detecting anomaly in the use of database attributes based on feature vectors extracted from database applications. The proposed approach is able to detect the anomalies of attribute usage for database applications with high accuracy while keeping the false positive rate reasonably low. On average, the detection rate is 92.8%, and the false positive rate is 0.57%. The proposed approach can be used in the verification and validation of database applications.

In future, we will conduct comprehensive experiments on a larger set of industry database applications to further validate the merits of the proposed approach. Currently, we are labeling the clusters based on the parameters we defined. Another possibility would be to label clusters which are outliers in the feature space as anomalous. Furthermore, we will also try to make the process of selecting optimal value of the parameter  $p$

automatically, by using statistical knowledge such as the mean value of the number of attributes in clusters.

## REFERENCES

- [1] A. H. Watson, T. J. McCabe, and D. R. Wallace, "Structured testing: A testing methodology using the cyclomatic complexity metric," *NIST special Publication*, vol. 500, pp. 1-114, 1996.
- [2] phc. (2012). *The Open Source PHP Compiler*. Available: <http://www.phpcompiler.org/>
- [3] L. Portnoy, E. Eskin, and S. Stolfo, "Intrusion detection with unlabeled data using clustering," in *In Proceedings of ACM CSS Workshop on Data Mining Applied to Security (DMSA-2001)*, 2001.
- [4] E. Eskin, A. Arnold, M. Prerau, L. Portnoy, and S. Stolfo, "A geometric framework for unsupervised anomaly detection," in *Applications of data mining in computer security*, ed: Springer, 2002, pp. 77-101.
- [5] E. Eskin, "Anomaly Detection over Noisy Data using Learned Probability Distributions," in *In Proceedings of the International Conference on Machine Learning*, 2000.
- [6] J. Oldmeadow, S. Ravinutala, and C. Leckie, "Adaptive clustering for network intrusion detection," in *Advances in Knowledge Discovery and Data Mining*, ed: Springer, 2004, pp. 255-259.
- [7] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys (CSUR)*, vol. 41, p. 15, 2009.
- [8] W. K. Robertson, F. Maggi, C. Kruegel, and G. Vigna, "Effective Anomaly Detection with Scarce Training Data," in *NDSS*, 2010.
- [9] B. Kidwell, "A Decision Support System for The Classification of Software Coding Faults: A Research Abstract," in *Proceedings of the 33rd International Conference on Software Engineering (ICSE), 2011*, 2011, pp. 1158-1160.
- [10] B. Livshits and T. Zimmermann, "DynaMine: Finding Common Error Patterns by Mining Software Revision Histories," in *ACM SIGSOFT Software Engineering Notes*, 2005, pp. 296-305.
- [11] Z. Li and Y. Zhou, "PR-Miner: Automatically Extracting Implicit Programming Rules and Detecting Violations in Large Software Code," in *ACM SIGSOFT Software Engineering Notes*, 2005, pp. 306-315.
- [12] Y. Xie and S.-Z. Yu, "A large-scale hidden semi-Markov model for anomaly detection on user browsing behaviors," *Networking, IEEE/ACM Transactions on*, vol. 17, pp. 54-65, 2009.
- [13] G. Vigna, F. Valeur, D. Balzarotti, W. Robertson, C. Kruegel, and E. Kirda, "Reducing errors in the anomaly-based detection of web-based attacks through the combined analysis of web requests and SQL queries," *Journal of Computer Security*, vol. 17, pp. 305-329, 2009.
- [14] J. Zhang and H. Wang, "Detecting outlying subspaces for high-dimensional data: the new task, algorithms, and performance," *Knowledge and information systems*, vol. 10, pp. 333-355, 2006.
- [15] J. A. Jones, M. J. Harrold, and J. Stasko, "Visualization of test information to assist fault localization," in *Proceedings of the 24th international conference on Software engineering*, 2002, pp. 467-477.
- [16] M. Renieres and S. P. Reiss, "Fault localization with nearest neighbor queries," in *Proceedings. 18th IEEE International Conference on Automated Software Engineering, 2003*, 2003, pp. 30-39.
- [17] S. Hangal and M. S. Lam, "Tracking down software bugs using automatic anomaly detection," in *Proceedings of the 24th international conference on Software engineering*, 2002, pp. 291-301.
- [18] A. Podgurski, D. Leon, P. Francis, W. Masri, M. Minch, J. Sun, et al., "Automated support for classifying software failure reports," in *25th International Conference on Software Engineering, 2003. Proceedings*, 2003, pp. 465-475.
- [19] K. Liu, H. B. K. Tan, X. Chen, H. Zhang, and B. Padmanabhuni, "Automated Extraction of Data Lifecycle Support from Database Applications," in *Proceedings of the 23rd International Conference on Software Engineering & Knowledge Engineering (SEKE/2011), July 7-9*, 2011, pp. 432-437.

# RefactoringScript: A Script and Its Processor for Composite Refactoring

Linchao Yang  
Waseda University  
Tokyo, Japan  
young@fuji.waseda.jp

Tomoyuki Kamiya  
Waseda University  
Tokyo, Japan  
kamiya7140@akane.waseda.jp

Kazunori Sakamoto  
National Institute of  
Informatics  
Tokyo, Japan  
exkazuu@nii.ac.jp

Hironori Washizaki  
Waseda University  
Tokyo, Japan  
washizaki@waseda.jp

Yoshiaki Fukazawa  
Waseda University  
Tokyo, Japan  
fukazawa@waseda.jp

**Abstract**—Refactoring is widely recognized as a method to improve the internal qualities of source code. However, manual refactoring is time-consuming and error-prone. Consequently, many tools to support automated refactoring have been suggested, but most support only unit and simple refactoring, making it difficult to perform composite refactoring (e.g., introducing a design pattern) where a refactoring set is applied at one position or the same refactoring operation is applied at multiple positions. In this paper, we propose a novel script language and its processor to describe how and where to refactor by a model expressing source code<sup>\*1</sup>. Evaluations indicate that our language and processor allow refactoring steps to be described as scripts, which can be easily replayed and reused for multiple projects.

**Keywords:** Refactoring; Code Manipulation;

## I. INTRODUCTION

Refactoring, which is defined as “a technique to improve the design of the internal structure of software without changing its external behaviors”, has become commonplace in recent years. Although the most common refactoring techniques (e.g. extract and change field names) have been organized into patterns, manual refactoring is time-consuming and error-prone. To resolve this problem, many automatic refactoring tools and methods have been proposed.

Here, we define the following two terms, *basic refactoring* and *composite refactoring* [6], to specify two types of refactoring. Basic refactoring refers to a simple refactoring that cannot be decomposed further. In this paper, basic refactorings indicate the standard refactoring features of Eclipse. On the other hand, composite refactoring refers to a combination of basic refactorings. Combinations include applying several refactorings in one place, the same refactoring to several places, or both.

Although many current tools support automatic execution of basic refactoring, there is no mechanism to define and apply composite refactoring. As suggested by Vakilian et al. [9] in their survey on the trend of using refactoring in Eclipse by examining recordings, composite refactoring is very common.

<sup>\*1</sup> The preliminary idea of RefactoringScript has been originally proposed in [12] in Japanese. [12] has proposed the RefactoringScript with JRuby implementation and did not achieve the statement-level operations. In contrast this paper introduces RefactoringScript with Scala implementation. Moreover, we have added the capability of manipulating statements to make RefactoringScript support more complicated refactoring.

The complex composite refactorings such as introducing a design pattern were explicated in [4] and [5].

However, manual composite refactoring has a high execution cost since programmers have to locate the target code and select a menu or use a keyboard shortcut to call a basic refactoring every time. Additionally, as the numbers of locations and refactorings increase, performing each basic refactoring correctly becomes more difficult, and the risk of omissions increases. Although many patterns have been established for refactoring, majority tools cannot record and reuse the processes of the patterns. Therefore, it is difficult to apply the frequently used composite refactorings to cross-projects. Eclipse can record and replay refactoring operations as a script, but it is used to help programmers upgrade an older version of a library once a newer version is distributed. It is impossible to create a script arbitrarily and describe the steps of refactoring freely.

In this paper, we propose RefactoringScript, which contains a script that can be used to describe the processes of composite refactoring and its corresponding processor to perform refactoring. We address the following research questions:

- RQ1 Is it possible to script and apply composite refactoring (locations and actions) concisely and accurately?
- RQ2 Compared to the case without this tool, is composite refactoring performed correctly?
- RQ3 Compared to the case without this tool, is the cost of composite refactoring reduced?
- RQ4 Is it possible to reuse the composite refactoring in cross-projects?

The contributions of this paper are:

- A RefactoringScript language to describe the processes of composite refactoring
- A RefactoringScript processor to apply scripted composite refactoring
- Implementation of a RefactoringScript language and its processor as an Eclipse plug-in
- Evaluation of a RefactoringScript language and its processor to show its utility

Our paper is organized as follows. Section II provides motivating examples. Then Section III describes the proposed

RefactoringScript language and its processor, while Section IV presents the results and discussion of our experimental evaluation. Finally, Section V concludes the paper.

## II. BACKGROUND

We consider three cases as motivating examples.

### A. Renaming Relevant Elements

According to [9], a common combination of refactorings is renaming both of a field and its related elements. For example, after changing a field name, the names of its accessors should also be changed.

However, current refactoring tools only change the definition and references when applying rename refactoring to a field. For instance, if rename refactoring is invoked to change the name of field “page” in Listing 1 to “pageCount”, the name of the corresponding accessor will not be changed automatically. Hence, programmers have to invoke the rename refactoring again for renaming the accessor. Figure 1 is an example of RefactoringScript to do these two refactorings.

### B. Applying Coding Conventions

Many projects have their own coding conventions. To enhance maintainability of the entire code, especially for team development, all team members must observe the coding convention established before development. Reference [7] and [8] summarize some underlying rules, which can be used and modified freely. For example, one rule (27) places the underscore prefix or suffix of the name for a private, protected field, while another rule (44) avoids overloading the method.

For a project with an already inflated scale, the following is necessary to apply these conventions:

a) Among the protected or private fields, extract all names without an underscore prefix (or suffix), and execute the rename method. (Example script is shown in Figure 2.)

b) Acquire all methods that have the same name and the same number of arguments from a specific class, and execute the rename method.

Coding conventions can be used in multiple projects. When a new coding convention is applied to an existing source code or the old coding convention of a project is changed, the execution cost for refactoring all relevant places is very high. Furthermore, as the number of places that need refactoring increases, the risk of making mistakes increases.

### C. Introducing Design Patterns

The transformation to introduce a design pattern involves many iterations of refactoring. For example, introducing a Visitor Pattern includes the following two types of refactoring:

a) Move Method

b) Rename Method (to avoid name collisions, add “visit” to the beginning of the name of the method that has been moved.)

Listing 1. Example of changing a field name and the corresponding accessor name (Left: original code; Middle: renamed field; Right: renamed accessor).

<pre>private int page; public int getPage() {     return page; }</pre>	<pre>private int pageCount; public int getPage() {     return pageCount; }</pre>	<pre>private int pageCount; public int getPageCount() {     return pageCount; }</pre>
--	--	---

```
1 val f = cls.fields.select(By.name("page"))
2 val m = cls.method("getPage")
3 f.rename("pageCount")
4 m.rename("getPageCount")
```

Figure 1. Example script of renaming field and the corresponding accessor

```
1 val fs = cls.fields.select(By.modifier(With.or("protected", "private")))
2
3 fs.foreach(f => {
4     if(!f.name.startsWith("_")) {
5         f.rename("_" + f.name)
6     }
7 })
```

Figure 2. Example script of applying coding conventions

Even if only the above two refactorings are considered, operations like “Find the methods by the specified signature from subclasses of a specified class, and move to another class” and “Rename method that has been moved with a new name based on the name of the target class” are necessary. These operations will be performed repeatedly, which is clearly a high cost. However, these refactorings can be formally scripted.

## III. REFACTORINGSCRIPT LANGUAGE AND ITS PROCESSOR

In this section, we describe the design of the proposed RefactoringScript language and its processor.

### A. Overview

On account of the excellent features of Scala, such as the seamless Java interoperability and the scripting language. We adopted Scala as our script language and developed a DSL base on it. In addition, we employed Eclipse JDT to perform the basic refactorings provided by Eclipse. The RefactoringScript consists of two components.

- RSCore<sup>\*2</sup>: The fundamental part, which includes elements of the RefactoringScript language and its processor.
- RSUI: The user interface part, such as an editor<sup>\*3</sup> to create or modify script, and a menu<sup>\*4</sup> to execute a script by inputting script into the interpreter of RefactoringScript in RSCore.

The procedure for a user to apply a script to a workspace, and the interaction between the user and RefactoringScript processor are as follows:

- (1) User creates and edits the script in the editor.
- (2) User activates the core component by specifying a script file.
- (3) Processor inputs the script into the interpreter.
- (4) Interpreter runs the script and applies it to the user’s workspace

\*2 <https://github.com/hugh3166/RSCore>

\*3 <https://github.com/hugh3166/RSEditor>

\*4 <https://github.com/hugh3166/RSlauncher>

(5) User is notified of the script execution result.

### B. Language

Herein, we describe the elements of the RefactoringScript language.

#### 1) Code Entity and Code Entity Collection

Java elements of JDT provide APIs, which are suitable for searching a particular element from the workspace. However, the Java elements do not have APIs that allow the conditions to be specified in detail to determine the specified elements. Two types of APIs are added to Code Entity (CE) which is a class based on Java Element:

- APIs to analyze and search. For example, the select method which we will describe later.
- APIs to trace the tree structure of the code in the description similar to the simple natural language. For example, we prefer to use `c.methods` rather than `c.getMethods()` to acquire all methods of `c`.

Table I shows the correspondence between Java Element and CE. An indentation in the table represents the containment relationship of the package or the class inheritance. It should be noted that `RSWorkspace` differs slightly from the other CEs; `RSWorkspace` represents a reference to the current workspace, and is a starting point to find the other CEs.

The Code Entity Collection (CEC) represents a set of CEs and provides APIs that can search for CEs included in the set. An example script to perform a search is introduced in the next unit.

#### 2) Query Selector and Qualifier

To search for a CE from CEC, we can use select method by combining *SearchParams*, *QuerySelector*, and *Qualifier* in the following format:

```
CEC.select(QuerySelector (Qualifier (SearchParams)))
```

```
QuerySelector ::=
```

```
”By.name”|”By.namereg”|”By.modifier”|”By.typename”
```

```
Qualifier ::= ””|”With.or”|”With.and”|”With.out”
```

*QuerySelector* is a keyword that specifies the Search Key, which can be one of: name, regular expression of name, access modifier, and type name. Table II shows each CE and the corresponding combination of the search key and query selector, where O indicates that *QuerySelector* can search CE using the search key, and X indicates it cannot. For example, a set of `RSProjects` can be searched by key elements with a name, but cannot with an access modifier. *Qualifier* is a keyword to specify how to interpret the given search parameters with OR, AND, or NOT. However, it can be omitted when a qualifier is not required (if there is only one search parameter). Figure 3 shows three examples of using the select method. The select method has also been used in Figure 1 and Figure 2 in the first line.

#### 3) Action

```
1 // Search CE which has "private" modifier
2 ms.select(By.modifier("private"))
3
4 // Search CE which has "private" modifier or "protected" modifier
5 fs.select(By.modifier(With.or("private", "protected")))
6
7 // Search CE which has "public" modifier and neither "int" type nor "String" type
8 ms.select(By.modifier("public")).select(By.typename(With.out("int", "String")))
```

Figure 3. Examples of using the select method

TABLE I. CORRESPONDENCE BETWEEN THE ELEMENTS IN JDT AND CE

org.eclipse.jdt.core	
IMember	RSMember
IType	RSCClass
IField	RSField
IMethod	RSMMethod
IPackageFragment	RSPackage
ILocalVariable	RSPParameter
IJavaProject	RSProject
org.eclipse.core.dom	
Statement	RSStatement
org.eclipse.core.resources	
ResourcesPlugin	RSWorkspace

TABLE II. QUERY SELECTOR

Search Key	Name	Regular Expression of Name	Access Modifiers	Type
<i>QuerySelector</i>	<i>By.name</i>	<i>By.namereg</i>	<i>By.modifier</i>	<i>By.typename</i>
RSStatement	X	X	X	O
RSField	O	O	O	O
RSMMethod	O	O	O	O
RSCClass	O	O	O	X
RSPParameter	O	O	X	O
RSProject	O	O	X	X
RSWorkspace	X	X	X	X

TABLE III. TYPES OF SUPPORTED ACTIONS AND CORRESPONDING PARAMETERS

Action	Receiver	Parameter
rename	RSField	New name
rename	RSMMethod	New name
encapsulate	RSField	-
introduce_factory	RSCClass	Destination Class
introduce_factory	RSMMethod	Destination Class
introduce_parameter_object	RSMMethod	Class Name of Object
pull_up	RSMMethod	Destination Class
push_down	RSMMethod	-
change_return_type	RSMMethod	New Type Name
extract_method	RSStatement	New Method Name
delete	RSEntity	-
move	RSEntity	Destination Class

A refactoring operation for CE / CEC is called an action. With parameters (params), an action is expressed in the following format:

CE/CEC.Action(params)

TABLE IV. COMPARISON OF NUMBER OF LINES OF SCRIPT IN REFACTORINGSCRIPT WITH JAVA

	Java	RefactoringScript
EX1	42	10
EX2	143	14
EX3	107	9
EX4	48	12

Unit: Lines

TABLE V. COMPARISON OF EXECUTION TIME FOR REFACTORING BY SCRIPT AND MANUALLY

Experiment	P1	P2	P3	P4	P5	Average
EX1(Manually)	7	-	5	-	-	6.0
EX1(Script)	-	10	-	5	14	9.7
EX4(Manually)	-	17	-	9	13	13.0
EX4(Script)	22	-	10	-	-	16.0

Unit: Minutes

TABLE VI. COMPARISON OF ACCURACY OF REFACTORING BY SCRIPT AND MANUALLY

Experiment	P1	P2	P3	P4	P5	Average
EX1(Manually)	27	-	30	-	-	28.5
EX1(Script)	-	30	-	30	30	30
EX4(Manually)	-	95	-	96	93	94.7
EX4(Script)	96	-	96	-	-	96

Unit: Places

TABLE VII. APPLIED REFACTORINGSCRIPT TO OPEN SOURCE PROJECTS

Project	Experiment	Number of Files	Number of Lines	Applying Places
P1	EX1	3	68	16 fields
P1	EX4	2	18	6 fields 12 methods
P2	EX3	6	20	6 classes 6 methods

```

1 import org.eclipse.jdt.core.dom.ASTNode
2
3 val proj = RSWorkspace.project("Ex0")
4
5 val cls1 = proj.pkg("p").classes.select(By.Name("Game1")).first
6 val mhd1 = cls1.method("startGame").first
7 val ifStmt1 = mhd1.body.statements.findByKind(ASTNode.IF_STATEMENT)
8
9 val cls2 = proj.pkg("p").classes.select(By.Name("Game2")).first
10 val mhd2 = cls2.method("startGame").first
11 val ifStmt2 = mhd2.body.statements.findByKind(ASTNode.IF_STATEMENT)
12
13 ifStmt1.extractMethod("extracted")
14 ifStmt2.extractMethod("extracted")
15
16 val mExtr1 = cls1.method("extracted").first
17 val ms = new RSCollection(Array(mExtr1, mhd1, mhd2))
18 ms.pullUp(cls2.superclass)

```

Figure 4. Example script of EX2

An action parameter may be specified as the minimum required when performing refactoring. Table III summarizes the types of the actions which have been supported and the parameters which can currently be specified. For example,

CE.rename("newname") will apply rename refactoring to CE. As we showed at line 3 and 4 in Figure 1, or line 5 in Figure 2.

### C. Processor

In this tool, the RefactoringScript language can be regarded as an internal DSL of Scala, and its processor is also implemented with Scala. Programmers only need to focus on the descriptions of searching and handling CE, because Scala expressions and the built-in functions or libraries of both of Java and Scala are available in the script. Moreover, the Scala Interpreter was incorporated directly into the processor without implementing a new interpreter.

## IV. EVALUATION

### A. Evaluation Design and Results

To evaluate the describability, accuracy, execution cost, and reusability of RefactoringScript, we conducted subject experiments and case studies for the four composite refactorings, which were selected by considering trends of refactoring [9] and coding conventions [7, 8].

EX1. Assign a prefix to the name of every private field for all classes in a specific package.

EX2. Generate template method from subclasses into a superclass.

EX3. Encapsulate classes with Factory.

EX4. Change the name of a specified field in a package and the name of the corresponding accessor.

#### 1) Describability

For EX1, 2, 3, 4, we measured the lines of code for processing refactorings in the Java language and the RefactoringScript language (Table IV). Note that the Java projects used as experimental objects are also the test data used for testing RSCore.

For EX2, we used the simplified experimental code in Listing 2. Two subclasses have the same method name, "startGame", but they have different conditional structures. For simplicity, we just extract the conditional structures to form two new methods with same name, and pull up the two "startGame" methods and one of the extracted methods into a superclass. Because the two "startGame" methods are the same method in the superclass, they can be regarded as a template method. Afterwards, the subclasses can change the behavior of the "startGame" by overriding the method "start" or "extract" without overriding the "startGame" method directly, such as the class Game2 shown at the right of Listing 2.

#### 2) Accuracy and Execution Cost

We conducted the following subject experiments to compare the accuracy and execution of composite refactoring cost between using RefactoringScript and simply using Eclipse refactoring. The experimental objects are the sample projects prepared for the experiments. It should be noted that in consideration of the similarity of operation difficulty and the influence of prior knowledge, we only used EX1 and 4 as the experimental objects. In addition, for simplicity, we chose int type fields to be the target fields, and wrote the script to extract the fields by type in EX4.

Experimental Subjects: Five Information Engineering undergraduate and graduate students (P1~P5)

Procedure: Divide subjects into two groups. Make one group conduct EX1 by Eclipse, then EX4 by RefactoringScript, and make the other group conduct EX1 by RefactoringScript, then EX4 by Eclipse. Then measure the time necessary to complete refactorings and the places where refactoring is applied correctly.

Tables V and VI summarize the results of this subject experiment. Table V shows that subject P1 took seven minutes to do EX1 by Eclipse, 22 minutes to do EX4 by script, while Table VI shows that subject P1 applied refactoring correctly at 27 of 30 places in EX1 by Eclipse, but 96 of 96 places in EX4 by script.

### 1) Reusability (Case Study)

We applied the EX1, 3, 4 to open source projects P1<sup>\*5</sup> and P2<sup>\*6</sup> with both manual refactoring and RefactoringScript, and then got the mechanical differences. By verifying the results with the goals, it is confirmed that the proposed method is able to process the object refactoring. Because the focus is on only the refactoring operation, we selected projects with a moderate scale as the experimental material. Table VII lists the projects, kinds of experiments, numbers of files, numbers of lines affected by refactoring, and the applied places.

## B. Discussion

### 1) Describability

RQ1 Is it possible to script and apply refactoring operations (applying places and actions) concisely and accurately?

In all four cases, the number of lines of script written in RefactoringScript is 1/4 to 1/10 of the script written in Java. The reduction is attributed primarily to two reasons:

- API allows CE to be flexibly searched, and conditional statements are less likely to become a nest structure.
- Processes not directly related to refactoring (e.g. acquiring workspace) do not have to be described.

In EX2, the project, package, class, and method entities are searched by name, but a statement is searched by the type defined at the ASTNode class in JDt. Figure 4 shows the example script of EX2. Additionally, even on a scale like EX3, the number of lines of script written in RefactoringScript can be as few as 10. Because the RefactoringScript language is specialized to describe the processes and search locations for refactoring, concise scripting can be realized.

### 2) Accuracy and Execution Cost

RQ2 Compared to the case without this tool, is composite refactoring executed correctly?

RQ3 Compared to the case without this tool, is the cost of composite refactoring reduced?

With regard to the execution cost, manual refactoring required slightly less time in each experiment. Based on the feedback from the subjects, this is likely because learning RefactoringScript takes some time. In fact, some of the subjects commented:

Listing 2. Example to generate template method (Left: before refactoring; Right: after refactoring)

<pre> package p; public class Game {     protected int playerCount = 0;     public void start(){         System.out.println("Game Start");     } }  public class Game1 extends Game {     public void startGame(){         start();         if (playerCount != 0){             System.out.println("Restart");         }     } }  public class Game2 extends Game {     public void startGame(){         start();         if (playerCount &gt;= 0){             playerCount++;             System.out.println("Join");         }     } } </pre>	<pre> package p; public class Game {     protected int playerCount = 0;     public void start(){         System.out.println("Game Start");     }      public void startGame(){         start();         extracted();     }      protected void extracted(){         if (playerCount != 0){             System.out.println("Restart");         }     } }  public class Game1 extends Game { }  public class Game2 extends Game {     protected void extracted(){         if (playerCount &gt;= 0){             playerCount++;             System.out.println("Join");         }     } } </pre>
--	---

- “I was confused by the script language idiom.”
- “I think that RefactoringScript can reduce the time once I learned how to write with it.” (In this experiment, the answers to the examples and script pieces required by the experiment were distributed as material.)

On the other hand, feedback regarding manual composite refactoring indicated a desire for an automatic method:

- “I do not want to refactor more complex objects manually.” (For example, when the number of applicable places is enormous).
- “Firstly, I prefer not to do simple mechanical work manually.”

Therefore, we believe that once programmers become familiar with RefactoringScript, the burden of refactoring can be reduced.

With regard to accuracy, all the scripts written by the subjects worked properly using the script case, whereas the manual composite refactoring with Eclipse contained the following mistakes: renamed fields that are not specified (EX1) and fields renamed correctly, but the names of the accessors were incorrect (EX4). These mistakes indicate that the script contributes to correct composite refactoring.

### 3) Reusability

RQ4 Is it possible to reuse the refactoring operations in

\*5 <https://github.com/shigenobu/acbook-wa710>

\*6 <http://code.google.com/p/jslideshare/>

other projects?

The scripts used in EX1 and EX4 can be applied to other projects without substantial modification. Most projects require the following changes: package name of the object and action parameters (e.g. in EX4, P1 changes field names ‘created’, ‘updated’, ‘executed’ to ‘createdAt’, ‘updatedAt’, ‘executedAt’ as well as the corresponding accessor names). However, these elements are project-specific and are the minimum parameters that the user must specify for each project.

### C. Limitations

#### 1) Lazy Evaluation

In this tool, it is impossible to reflect the effect of refactoring on the CE. When multiple actions are performed on the same CE, the specified CE must be searched in each case. This problem should be resolved by introducing the lazy evaluation, which is a mechanism to set aside the query search for CE until execution. In addition, there is almost no issue to apply many basic refactorings to a particular CE.

#### 2) Behavior Preservation

We have tried to reduce the risk of breaking code by using Eclipse’s refactoring features in our tool. The preconditions provided by JDT are checked before performing every basic refactoring, and warnings will be prompted as well if operations will cause compilation errors. However, there are still some operations that may change the behavior of programs [13]. Thus, we cannot ensure that the behavior will not be changed while executing a script. Soares et al. [13] proposed an automated approach for testing Java refactoring engines based on program generation. It will help us to obtain more complete and more proper preconditions. Lahiri et al. [14] proposed a semantic differencing tool to help the developers understand the impact of changes faster. It is very useful for verifying behavior changes in a narrow scope such as inner functions. However, our focus is on helping programmers to perform refactoring more easily. After refactoring, a comprehensive test to check the behavior of a program is still essential.

## V. RELATED WORK

Vakilian et al. [9] proposed Compositional Paradigm to implement composite refactoring by setting the operations in detail. Vakilian et al. [10] also reported that setting the dialog may disrupt coding workflow, bring about more overhead costs and decrease productivity. Mens et al. [11] reported that even if options can be finely set, extensions and settings to match the domain of the object are insufficient in current tools. RefactoringScript script was realized as an easy composite refactoring approach with minimized action parameters and simplified search methods.

Li et al. [15][16] proposed a DSL for scripting refactoring in Erlang. Similarly, it uses a script that contains refactoring processes and locations to perform composite refactoring. It also executes scripted refactoring operations and checks preconditions successively like RefactoringScript. However, their refactoring locations scripting uses template-based pattern matching. As a result, it is difficult to apply to Java or other programming languages. Generally, pure object-oriented programming languages like Java can be represented in a tree structure intuitively with containment relationships between

packages or classes, fields, and methods. Therefore, we believe it is more concise to use an interface searching a tree structure from the top down, such as XPath [17] and DOM selector of JQuery [18], in Java and many other languages. The search APIs of RefactoringScript are modeled after this.

## VI. CONCLUSION AND FUTURE WORK

We have proposed a RefactoringScript language and its processor to script refactoring processes and apply them to appropriate places. By implementing the CE searching API and Scala, we realized a user-friendly scripting language. This tool should significantly reduce the cost of applying refactoring to many places or repeatedly applying refactorings across projects. We are currently working on improving RefactoringScript and developing a refactoring library, which should cover the most popular refactoring processes, based on this tool. Furthermore, although the refactoring types supported by RefactoringScript are limited to the refactoring features provided in Eclipse, we intend to expand it and realize more flexible code deformation. Besides, we will also design a new experiment excluding the learning time to evaluate our tool more accurately.

## REFERENCES

- [1] Eclipse. <http://www.eclipse.org/>.
- [2] Eclipse Java development tools (JDT). <http://www.eclipse.org/jdt/>.
- [3] Martin Odersky. Scala. <http://www.scala-lang.org/>.
- [4] Martin Fowler. “Refactoring: Improving the Design of Existing Code” Addison-Wesley, 2000.
- [5] Joshua Kerievsky. “Refactoring to Patterns.” Prentice Hall. 2004.
- [6] Mel Ó Cinneide, Paddy Nixon. “Composite refactorings for java programs,” Technical report, 2000.
- [7] Oracle, Code Conventions for the Java Programming Language. <http://www.oracle.com/technetwork/java/index-135089.html>.
- [8] Kenji Hiranabe, Object Club. “Java Coding Standard”.(in Japanese) <http://www.objectclub.jp/community/codingstandard/CodingStd.pdf>.
- [9] Mohsen Vakilian, Nicholas Chen, Roshanak Zilouchian Moghaddam, Stas Negara, and Ralph E. Johnson. “A compositional paradigm of automating refactorings.” Technical report, University of Illinois, 2012.
- [10] Mohsen Vakilian, Nicholas Chen, Stas Negara, Balaji Ambresh Rajkumar, Brian P. Bailey, and Ralph E. Johnson. “Use, disuse, and misuse of automated refactorings.” In ICSE 2012, pp.233–243, Piscataway, NJ, USA, IEEE Press.
- [11] T. Mens and T. Tourwe. “A survey of software refactoring.” Software Engineering. IEEE Transactions on, Vol.30, No.2, pp.126–139,feb 2004.
- [12] Tomoyuki Kamiya, Kazunori Sakamoto, Hironori Washizaki, Yoshiaki Fukazawa, “Refactoring Script: A script for composite refactoring and its processor.” IPSJ Transactions on Programming, Vol. 6, No.3, 2013. (in Japanese)
- [13] Gustavo Soares, Rohit Gheyi, Tiago Massoni. “Automated Behavioral Testing of Refactoring Engines”, IEEE Transactions on Software Engineering Vol. 39, pp. 147-162, Feb 2013.
- [14] Shuvendu K. Lahiri, Chris Hawblitzel, Ming Kawaguchi, Henrique Rebêlo. “SYMDIFF: a language-agnostic semantic diff tool for imperative programs.” In CAV’12 pp. 712-717 Springer-Verlag Berlin.
- [15] Huiqing Li, Simon Thompson. “A domain-specific language for scripting refactorings in erlang.” In FASE’12, pp. 501–515, Berlin, Heidelberg, 2012. Springer-Verlag.
- [16] Huiqing Li and Simon Thompson. “Let’s make refactoring tools user-extensible!” In the Fifth Workshop on Refactoring Tools, WRT ’12, pp. 32–39, New York, NY, USA, 2012. ACM.
- [17] XML Path Language(XPath). <http://www.w3.org/TR/xpath/>.
- [18] JQuery. <http://jquery.com/>.

# Investigation for Software Power Consumption of Code Refactoring Techniques

Jae-Jin Park, Jang-Eui Hong, and Sang-Ho Lee

Department of Computer Science

Chungbuk National University

Cheongju, Chungbuk, 361-763, Republic of Korea

jjpark@selab.cbnu.ac.kr, jehong@chungbuk.ac.kr, shlee@chungbuk.ac.kr

**Abstract**—Code refactoring technique focuses on enhancing the maintainability of software to extend its lifetime. However, there are other efforts to improve software qualities like performance or reliability as well as maintainability by using code refactoring techniques. Recently, as low-power software has become one of the critical issues in mobile environments, developing energy-efficient software through code refactoring becomes an important one. This paper aims to investigate whether the existing refactoring techniques can support energy-efficient software generation or not. The refactored codes generated by the existing techniques can consume more power than original codes because they did not consider the power consumption in their refactoring processes. This paper analyzes the power consumption to investigate the energy efficiency of M. Fowler’s refactoring techniques. Our analysis result can provide useful information about energy-efficient refactoring techniques to software engineers, and support the development of software that has high maintainability and good energy efficiency.

**Keywords** - code refactoring; power consumption; energy efficiency; maintainability

## I. INTRODUCTION

Comparing with general software systems, mobile and embedded systems have a characteristic of a limited battery capacity in their operations [1]. Therefore, in order to provide sustainable service to users, those systems have to manage the efficient use of the battery power. During the last five years, software power analysis as well as hardware-based power analysis has been one of interesting topics in power analysis research [2,3]. It is continuously increasing due to the wide use of smartphones and tablet PCs [4-6].

Code refactoring is one of useful techniques to improve software maintainability, and it also was used to improve other software qualities such as performance or reliability of software [7,8]. In addition to such valuable approaches, the “energy efficiency” as a new software quality factor, should be considered in those refactoring techniques. For example, when software engineers refactor their legacy codes to improve maintainability, the refactoring can produce any code that consumes the power more than the original legacy code does. Therefore, it is difficult to intuitively apply the general or the existing refactoring techniques to mobile or embedded software which has low-power requirements. The motivation of this paper is to consider the improvement of energy efficiency as

well as maintainability [9,10] when we refactor legacy code or open-source code.

This paper investigates the power consumption conditions for the refactoring techniques suggested by Martin Fowler [11]. M. Fowler’s 68 refactoring techniques provide general and common concepts, and they also cover almost all industrial practices to software refactoring. Thus we believe that it is enough to decide which refactoring technique can be energy-efficient by doing that experiments and analysis of the power consumption for M. Fowler’s techniques.

In order to achieve our research goal, we firstly obtained the original codes from [11] and open-source site, and then developed the refactored codes – we will call it refactor code – from the original codes by the applications of Fowler’s refactoring techniques. Then, we estimated the power consumption of these two codes using a software power estimation tool, and briefly analyzed the reasons for the variance of the power consumption. These results can give useful information to software engineers when they want to refactor any legacy code with consideration of low-power consumption.

## II. RELATED WORK

The representative studies on the relationship between code refactoring techniques and software energy efficiency have been performed by M. Gottschalk [12, 13] and W. Silva [14].

M. Gottschalk [12] suggested various kinds of energy code smells that can be appeared in source code. He reappraised general code smells like “Loop Bug” and “Dead Code” in the view of power consumption. Also he proposed a method to minimize the power consumption through code restructuring. However his restructuring method is too general to support practical use, as one of the solutions of energy code smells.

M. Gottschalk [13] also considered power consumption issue for Android applications by using a similar approach with his previous work [12]. He defined the conditions of power waste for a mobile application, and then he proposed a code refactoring technique to resolve the waste conditions. He also showed that the proposed technique is appropriate to improve energy efficiency. But he did not describe the detail refactoring process enough to other’s use.

The study on the energy efficiency of a specific refactoring technique, “Inline-Method,” was performed by W. Silva [14]. He investigated the changes of aspect in the relationship between performance and power consumption by using “Inline-Method” refactoring technique. Even though the technique is common and easy to refactor legacy codes, there still remain various refactoring techniques to be used in practice.

The existing studies considered the power consumption issue of software such as code smells, performance, and power waste conditions. However, software engineers in practice want more available refactoring techniques satisfying both maintainability and energy efficiency to support open-source based or reuse-based software development.

### III. POWER CONSUMPTION ESTIMATION

We estimate the power consumption for every refactoring technique by M. Fowler. The estimation has been performed with original code and its refactor code, respectively to compare their power consumptions with each other. In this paper, energy efficiency means the degree of how less power can consume to provide the same service or functionality [15,16]. We represent the energy efficiency with the fluctuation of power consumption for intuitiveness.

#### A. Estimation Subjects

We estimate the power consumption for sources codes written in C++ programming language. The subjects of our estimation are 63 techniques which were selected from 68 techniques by M. Fowler. The excluded five are as follows:

- Change Unidirectional Association to Bidirectional,
- Change Bidirectional Association to Unidirectional,
- Replace Type Code with Subclasses,
- Replace Type Code with State/Strategy, and
- Encapsulate Downcast.

We excluded the 5 techniques from our estimation process because it is difficult to implement in C++, and the frequency of practical use is low.

#### B. Estimation Environment

To estimate the power consumption, we use a software power estimation tool, XEEMU [17] which supports C/C++ based estimation. The XEEMU is known as one of accurate tools that has 1.6% error rate in its estimation result when it is compared with actual measurement. Table I summarizes the platform specification of the XEEMU tool.

TABLE I. XEEMU PLATFORM SPECIFICATION

<b>CPU</b>	Intel XScale 80200 processor, 266 to 733 MHz in 66MHz
<b>Architecture</b>	ARM pipelined RISC
<b>RAM</b>	32 MB Micron SDRAM, 100MHz
<b>Cross Compiler</b>	arm-elf-g++ 4.1.1
<b>Measure. tap</b>	CPU core current, IO current, System peripherals

#### C. Design for Estimation

We prepare 63 example source codes to use in the estimation of power consumption for the refactoring techniques. Those codes are developed with the sample codes that are used in M. Fowler’s studies [11]. We customized the codes to be executed without any external input because it can cause the differences of power consumption for each code. Also, we added basic declaration statements into the refactor code to compile and execute the code.

Figure 1 shows the brief estimation process of the power consumption for refactoring techniques to get the variance of the consumptions in accordance with the application of M. Fowler’s techniques.

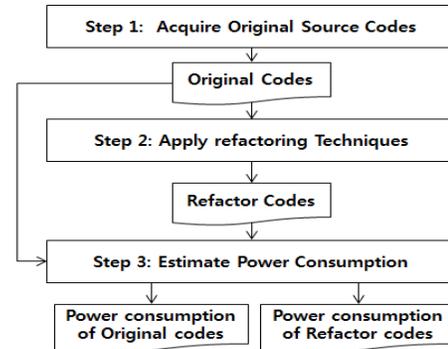


Figure 1. Estimation Process of Energy Consumption

#### D. Estimation Results

The power consumption values for the original code and the refactor code are listed in Table II to Table VII. Each table is grouped by catalogs of M. Fowler. The “Effect” field of the table represents the energy efficiency which is mentioned above. The field has the symbol,  $\uparrow$  when the power consumption is decreased, or the symbol,  $\downarrow$  when the power consumption is increased after a refactoring technique was applied. This variance is based on the output values of the XEEMU, and the values are truncated at 4 places of decimals.

The 26 out of the 63 techniques reveal the decrement of power consumption, and the 7 techniques have no changes in their consumption. Others consume more power than those of original code. Figure 2 shows the distribution of the power consumption values for original and refactor codes.

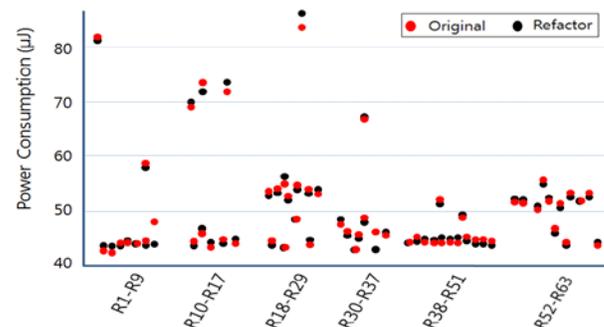


Figure 2. Power Consumption Distribution for M. Folwer’s Refactoring Techniques



R49	Replace Constructor with Factory Method	43.422	43.552	↓
R50	Replace Error Code with Exception	44.379	44.489	↓
R51	Replace Exception with Test	43.104	43.389	↓

TABLE VII. POWER CONSUMPTIONS FOR “DEALING WITH GENERALIZATION” TECHNIQUES.

No.	Refactoring Techniques	Consumption ( $\mu\text{J}$ )		Effect
		Original	Refactor	
R52	Pull Up Field	51.781	51.579	↑
R53	Pull Up Method	51.579	51.456	↑
R54	Pull Up Constructor Body	49.627	50.368	↓
R55	Push Down Method	51.827	51.827	-
R56	Push Down Field	50.164	49.673	↑
R57	Extract Subclass	51.574	52.204	↓
R58	Extract Superclass	54.885	55.967	↓
R59	Extract Interface	51.410	52.212	↓
R60	Collapse Hierarchy	52.400	51.967	↑
R61	Form Template Method	45.843	46.585	↓
R62	Replace Inheritance with Delegation	43.522	43.754	↓
R63	Replace Delegation with Inheritance	43.754	43.522	↑

#### IV. ESTIMATION RESULT ANALYSIS

Basically, code refactoring techniques have the intention of maintainability improvement. Therefore, approximately half the techniques make the extraction of new module to encapsulate original module, or the replacement of original module with well-organized module. However, these techniques can give negative effects in power consumption. Likewise, we can intuitively recognize that some refactoring techniques cause more power consumption than one of original. But there also exists exceptions of them.

##### A. Change Predictive Techniques

Those refactoring techniques which were included in our estimation use one or more methods, listed in the following, for their refactoring.

- Extraction or composition of module/variable,
- Movement of module/variable,
- Replacement of module/variable, and
- Introduction or removal of module/variable.

The module extraction of the above first method increases the power consumption - that is, decrement of energy efficiency because of the addition of the new module to the original code. It causes additional function calls and references to interact with each other. On the contrary, the module composition method decreases the power consumption because of the reduced interactions.

The above second method means that a module can move to an adequate location to make better structure of original code. It also decreases the power consumption due to removal of unnecessary relationships inherent in original code.

In the third method, the replacement means the substitution for original modules or variables with maintainable and understandable ones. Therefore, the power consumption varies depending on the changing forms. The last method is to encapsulate objects. It increases the power consumption when something is added and decrease when something is deleted.

However, some techniques that support the module movement do not improve the energy efficiency. The technique, R54: Pull Up Constructor Body, increases the power consumption, but the technique, R53: Pull Up Method, decreases the power consumption. Although the R9: Substitute Algorithm technique reveals to be consuming more power after the refactoring, as seen from our estimation results, it does not really mean the rise of energy efficiency. These special cases are explained in the following sections.

##### B. Substitute Algorithm Technique

The technique, “Substitute Algorithm,” is used to replace an original algorithm with a well-suited or particular algorithm. However, even though any algorithm was selected to the replacement, it can lead to more power consumption due to the complexity of the algorithm [18]. For example, the energy efficiency rises when we replace bubble sort algorithm with quick sort algorithm for data sorting, but the efficiency is down when the bubble sort algorithm is replaced with selection sort algorithm [19].

##### C. Pull Up Method vs. Pull Up Constructor Body

The techniques, R53: Pull Up Method and R54: Pull Up Constructor Body can be used to refactor the inheritance relationship because they move the common member functions of subclasses to functions of superclass [20]. Although these two techniques have the same concept, the signs of the power consumption are contrary to each other. The power consumption is decreased when we applied the R53 technique, while R54 technique increases the consumption.

When we use the method which moved from subclass after applying the R53 technique, the program counter is pointing to the method declaration in superclass. Therefore the number of low-level instructions is reduced, and it leads to the decrement of power consumption.

The R54 technique supports that the subclass wants to use the properties of superclass. After applying the R54 technique, the constructor of subclass defines and uses the constructor of superclass, repeatedly. This causes the number of low-level instructions to increase. So the power consumption also increased.

##### D. Energy Efficiency of the Techniques

We re-categorize M. Fowler’s refactoring techniques into 3 groups based on the energy efficiency of our estimation results as follows.

- Positive group for energy efficiency
- Even-handed group for energy efficiency
- Negative group for energy efficiency

Table VIII and Table IX list the positive group and the even-handed group for the energy efficiency, respectively. These techniques total 33 refactoring techniques that are extracted from 63 techniques to be estimated. This corresponds to 50% of the whole estimated refactoring techniques. The software engineer, who wants energy-efficient refactoring, can refer to Table VIII and Table IX to select the appropriate technique.

TABLE VIII. REFACTORIZING TECHNIQUES OF POSITIVE GROUP

No.	Code Refactoring Techniques
R2	Inline Method
R3	Inline Temp
R5	Introduce Explaining Variable
R10	Move Method
R11	Move Field
R13	Inline Class
R15	Remove Middle Man
R16	Introduce Foreign Method
R21	Change Reference to Value
R25	Encapsulate Field
R26	Encapsulate Collection
R29	Replace Subclass with Fields
R32	Consolidate Duplicate Conditional Fragments
R33	Remove Control Flag
R36	Introduce Null Object
R40	Remove Parameter
R41	Separate Query from Modifier
R42	Parameterize Method
R45	Replace Parameter with Method
R46	Introduce Parameter Object
R47	Remove Setting Method
R52	Pull Up Field
R53	Pull Up Method
R56	Push Down Field
R60	Collapse Hierarchy
R63	Replace Delegation with Inheritance

TABLE IX. REFACTORIZING TECHNIQUES OF EVEN-HANDED GROUP

No.	Code Refactoring Techniques
R6	Split Temporary Variable
R24	Replace Magic Number with Symbolic Constant
R27	Replace Record with Data Class
R34	Replace Nested conditional with Guard Clauses
R38	Rename Method
R48	Hide Method
R55	Push Down Method

## V. DISCUSSION

### A. Estimation Validity Threats

The first issue to discuss for the threats of our estimation validity is on the tool. We used a software energy analysis tool, XEEMU 1.3 to estimate the power consumption for M. Fowler’s refactoring techniques. The values of power consumption of software can be different whenever we use other hardware platforms to execute the software. The XEMMU tool analyzes the power consumption based on an ARM pipelined RISC architecture using XScale 80200 processor. However, even though we just estimated the power consumption using the XEEMU tool only, we can assert that our estimations are valid because it was done under consistent and/or same conditions. Estimating the power consumption based on another platform must be done in our next research step.

The second issue is on the relationship between the number of applied techniques and the change of energy efficiency. Is it possible to say that it is more energy efficient when we apply several refactoring techniques to an original code? The answer is not always. For example, the energy efficiency is improved when we apply both techniques, R3 and R40, to the Huffman algorithm, while the R2 and R42 techniques lead to bad energy efficiency when applied to the encryption algorithm. Therefore, we can decide the energy efficiency through adequate and repeat estimations for which a combination of refactoring techniques can improve the energy efficiency.

The last issue to be considered is on the difference of the power consumptions for the refactor codes. We developed a refactor code with a specific technique, but several versions of refactor codes can be developed using just one technique. For example, when applying the refactoring technique, “Introduce parameter object” to an original code, we can develop two or more refactor versions depending on how many parameters can be grouped into an object. The different versions can consume the different amounts of power. However the main concern of this paper is whether the refactoring technique can reduce the power consumption or not. Therefore we developed and selected the refactor code version toward maximizing the energy efficiency.

As one of additional considerations, we carefully developed the original codes and refactor codes manually. Even though some CASE tools generate the refactor code automatically, and the generation is more systematic, we considered to develop more profitable refactor codes than automatic generations in the aspect of energy consumption.

### B. Developing Energy-Efficient Refactoring Technique

When software engineers want to modify legacy code to enhance maintainability, they can refer to our analysis results of the power consumption to gain both maintainability and energy efficiency, even if they are limited to the techniques of M. Fowler. Table VIII and Table IX list the useful refactoring techniques in a view of power consumption to select adequate techniques by software engineers. However, if a software engineer wants to select the other technique which was excluded in both tables, it requires any other processing to

compensate the loss of energy efficiency. We propose the next two strategies to the compensation.

- Strategy 1: substitute it for a possible technique which has good energy efficiency.
- Strategy 2: redesign the refactoring process to improve energy efficiency.

The example of the strategy 1 is that the technique, R17: Introduce Local Extension can be substituted with the technique R16: Introduce Foreign Method in M. Fowler's one. Even though these two techniques have different notions to refactoring, they can be used to extend unamendable objects: the R17 extends the object using an inheritance mechanism, and the R61 uses a wrapper function for it. This substitute can easily put their energy efficiency into reverse.

It requires more in-depth study for the strategy 2. However, the technique, R8: Replace Method with Method Object basically generates a dynamic object instead of long function or long method. Thus the dynamic object performs its operations in place of the long function. In the refactoring process, we can supplement the existing process by adding a part of the technique R1: Extract Method. The revised technique R8 can reduce the power consumption in its refactoring process.

## VI. CONCLUSION AND FUTURE WORKS

Although code refactoring techniques, suggested by M. Fowler, have been used to enhance the maintainability of legacy code, we cannot be sure whether those techniques support energy efficiency. This paper estimated and analyzed the power consumption to M. Fowler's 63 refactoring techniques, and recommended 33 techniques among them as energy-efficient refactoring ones.

Even though our proposition can help software engineers to improve the qualities of their legacy code, there may exist some other techniques to be applied in their practice. Therefore it is important to consider energy efficiency as one of quality factors when we develop or maintain mobile or embedded software.

Our future work is to develop energy-efficient refactoring techniques as mentioned in Section 5.B. Especially we will define the detail refactoring procedure with the consideration of power consumption for the negative group of energy efficiency. Additionally, we will confirm our current estimation results through estimating the energy consumption of M. Fowler's refactoring techniques with another tool. We believe that this work gives practicality to software engineers in their code refactoring or maintenance works.

## ACKNOWLEDGEMENT

This research was supported by the Basic Science Research Program (2011-0010396) through the National Research Foundation of Korea funded by the MEST.

## REFERENCE

- [1] Jang-Eui Hong and Doo-Hwan Kim, "Task Extraction from Software Design Models to Improve Energy Efficiency of Embedded Software," *The KIPS Transactions: PartD*, 18D(1), pp.45-56, 2011. (in Korean)
- [2] Hu Jun, Li Xuandong, Zheng Guoliang, Wang Chenghua, "Modelling and Analysis of Power Consumption for Component-Based Embedded Software," *Proc. of EUC Workshop*, pp.795-804, 2006.
- [3] E. Senn, J. Laurent, N. Julien, E. Martin, "SoftExplorer: Estimating and Optimizing the Power and Energy Consumption of a C Program for DSP Application," *EURASIP Journal on Applied Signal Processing*, Vol.16, pp.2641-2654, 2005.
- [4] Doo-Hwan Kim, and Jang-Eui Hong, "Energy Component Library for Power Consumption Analysis of Embedded Software," *The KIPS Transactions: PartD*, 16D(6), pp.871-880, 2009. (in Korean)
- [5] Jong-Phile Kim, Doo-Hwan Kim, Jnag-Eui Hong, "Estimating power consumption of mobile embedded software based on behavioral model." *ICCE 2010, Digest of Technical Papers, IEEE*, pp.105-106, 2010.
- [6] H. Höpfner and C. Bunse, "Energy Awareness Needs a Rethinking in Software Development." *ICSOFT (2)*. pp.294-297, 2011.
- [7] Yejin Kwon, Zuna Lee, Youngbom Park, "Performance-based Refactoring: Identifying & Extracting Move-method Region," *Journal of KIISE: Software and Applications*, 40(10), pp.567-574, 2013. (in Korean)
- [8] Jae-Jin Park, Jang-Eui Hong, "An Approach to improve software safety by Code refactoring," *Proc. of Korea Computer Congress*, pp.532-534, 2013. (in Korean)
- [9] J. Jelschen, M. Gottschalk, M. Josefiok, C. Ptü, A. Winter, "Towards Applying Reengineering Services to Energy-Efficient Applications." *CSMR 2012, 16th European Conference, IEEE*, pp.353-358, 2012.
- [10] A. Vetro, L. Ardito, G. Procaccianti, M. Morisio, "Definition, Implementation and Validation of Energy Code Smells: an Exploratory Study on an Embedded System." *The 3rd Int' l Conf. on Smart Grids, Green Comm. and IT Energy-aware Tech...*, pp.34-39, 2013.
- [11] Fowler, Martin. *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 1999.
- [12] M. Gottschalk, M. Josefiok, J. Jelschen, A. Winter, "Removing Energy Code Smells with Reengineering Services." *GI-Jahrestagung*, pp.441-455, 2012.
- [13] M. Gottschalk, J. Jelschen, and A. Winter, "Energy-Efficient Code by Refactoring." *15. Workshop Software-Reengineering*, 2013.
- [14] Wellisson G. P. da Silva, L. Brisolará, Ulisses B. Correa, L. Carro, "Evaluation of the impact of code refactoring on embedded software efficiency." *Workshop de Sistemas Embarcados*, 2010.
- [15] Sunday Olayinka Oyedepo, "Efficient energy utilization as a tool for sustainable development in Nigeria." *Int' l Journal of Energy and Environmental Engineering* 3.1, pp.1-12, 2012.
- [16] Marc Rosen, "MA: Towards energy sustainability: a quest of global proportion." *Forum of Public Policy online: A Journal of the Oxford Round Table*, pp.1-20, 2008.
- [17] Z. Herczeg, D. Schmidt, A. Kiss, N. Wehn, T. Gyimothy, "Energy simulation of embedded XScale systems with XEEMU" , *Journal of Embedded Computing*, pp. 209-219, August, 2009.
- [18] Bob Steigerwald, Christopher D. Lucero, Chakravarthy Akella, Abhishek Agrawal, "Writing Energy-Efficient Software." *Energy Aware Computing, Intel Corporation*, pp.1-8, 2011.
- [19] C. Bunse, H. Hopfner, E. Mansour, S. Roychoudhury, "Exploring the energy consumption of data sorting algorithms in embedded and mobile environments." *MDM'09. The 10th Int' l Conf., IEEE*, pp.600-607, 2009.
- [20] Filip Van Rysselberghe, Matthias Rieger, Serge Demeyer, "Detecting move operations in versioning information." *CSMR 2006, The 10th European Conference., IEEE*, pp.271-278, 2006.

# Testing as an Investment

Xiaoran Xu<sup>12</sup>, Chunrong Fang<sup>1</sup>, Qing Wu<sup>3\*</sup>, Jia Liu<sup>1</sup>, Zhenyu Chen<sup>1</sup>

<sup>1</sup>State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China

<sup>2</sup>Rice University, Houston, Texas, USA

<sup>3</sup>School of Business, Nanjing University, Nanjing, China

\*Corresponding author: wuqing@nju.edu.cn

## Abstract

*Software testing is an expensive and important task. Plenty of researches and industrial efforts have been invested on improving software testing techniques, including criteria, tools, etc. These studies can provide guidelines to select suitable test techniques for software engineers. However, in some engineering projects, business issues may be more important than technical ones, hence we need to lobby non-technical members to support our decisions. In this paper, a well-known investment model, Nelson-Siegel model, is introduced to evaluate and forecast the processes of testing with different testing criteria. Through this model, we provide a new perspective to understand short-term, medium-term, and long-term returns of investments throughout the process of testing. A preliminary experiment is conducted to investigate three testing criteria from the viewpoint of investments. The results show that statement-coverage criterion performs best in gaining long-term yields; the short-term and medium-term yields of testing depend on the scale of programs and the number of faults they contain.*

**Keywords:** Investment, Nelson-Siegel model, Cost-yields, Testing criterion.

## 1 Introduction

Testing is an important and labor-intensive activity in software development life cycle. It always consumes from 30% to 50% of total development cost according to [1]. Many studies have been conducted to compare various testing criteria and tools aiming at providing guidelines of testing for practitioners[15]-[18]. Furthermore, a number of models, such as APFD, were proposed to evaluate testing criteria [2]. Commonly, these studies concern about the technical issues, such as fault detection capability, execution time, etc. However, in industries, it is the customers that invest and drive software development. For these cus-

tomers, business issues are more significant than technical ones. Therefore, we need to provide business evidences rather than technical guidelines for non-technical members in companies.

Investment knowledge is a branch of economics. It concerns about how to invest limited resources of individuals or organizations to financial assets, like stock, national debt and real estate, in order to gain reasonable cash flow and risk-benefit ratio. The key point of investment knowledge is finding the optimal equilibrium solution of personal assets allocation, under the guidelines of utility maximization criterion. Investment philosophy is widely accepted in daily life, among which Return On Investment (ROI) [3] is a way to consider profits of capital investment. Many ROI metrics were proposed in the past century[19], which are widely used to support marketing decisions in a lot of industries, including software industry [4]. In this paper, we innovatively propose the concept that testing is an investment and introduce a well-known model, Nelson-Siegel (NS) model [5], to calculate ROI of several testing criteria. NS model is a classical model to estimate the current term structure of interest and forecast the future term structure, which has been applied in many fields of finance [6][7]. In this paper, we will use it to model the relationship between investments and returns of testing.

The main contributions of this paper are as follows:

- To the best of our knowledge, it is the first time to use investment model to quantitatively investigate investments and returns of software testing.
- An experiment is conducted to analyze the yields of test activity and compare the yields among testing criteria: random, statement-coverage and branch-coverage.

The application of NS model has shown a potential prospect albeit our experiments just achieve a fundamental stage. The NS model, equipped with total economic indices such as fault price and test cost, could completely reflect

software testing to investment area. Thus a novel, practical and comprehensive testing evaluation method is created. This model can firstly evaluate and forecast the yields of software testing, secondly explore the greatest yields point and moreover calculate the cost and risk of remained faults according to the maximum long-term yields prediction. Details will be discussed in Section 4.

The rest of paper is organized as follows. Section 2 proposes the detail of our approach. The experiment design and the result analysis are presented in Section 3. We discuss the applicability of NS model in testing and some related parameters in Section 4. The conclusion and future work are presented in Section 5 and Section 6, respectively.

## 2 Approach

The NS model is proposed to optimally estimate, model and forecast the term structure of interest rates using a flexible, smooth and parametric function. It is capable of capturing many of the typically observed shapes that the yield curve assumes and is usually used in investment subject. If the instantaneous forward rate at maturity  $m$ , denoted  $r(m)$ , is given as

$$r(m) = \beta_0 + \beta_1 * e^{-m/\tau} + \beta_2 \left[ \frac{m}{\tau} * e^{-m/\tau} \right]. \quad (1)$$

Then the yield to maturity, denoted  $R(m)$ , is the average of the forward rates

$$R(m) = \frac{1}{m} \int_0^m r(x) dx. \quad (2)$$

So in Eq.1 the resulting function of integrating  $r(m)$  from zero to  $m$  and being divided by  $m$  (Eq.2) is

$$R(m) = \beta_0 + \beta_1 \frac{1 - e^{-\frac{m}{\tau}}}{\frac{m}{\tau}} + \beta_2 \left[ \frac{1 - e^{-\frac{m}{\tau}}}{\frac{m}{\tau}} - e^{-\frac{m}{\tau}} \right], \quad (3)$$

among which  $\beta_0$ ,  $\beta_1$ ,  $\beta_2$  and  $\tau$  are parameters to be estimated. It is easy to prove that

$$r(+\infty) = \beta_0, \quad (4)$$

and

$$r(+\infty) - r(0) = -\beta_1. \quad (5)$$

Thus,  $\beta_0$ ,  $\beta_1$ ,  $\beta_2$  are respectively the Level Factor, Slope Factor (the absolute value of  $\beta_1$ ) and Curvature Factor of the yield curve. From the perspective of dynamic period yields, they are also long-term, short-term, and medium-term components. The parameter  $\tau$  controls the speed of

other parameters decaying, especially the extreme value of the loading to which  $\beta_2$  attaches.

Assuming it can be applied to testing, NS model will be able to model, evaluate and forecast the structure of testing yield curve. To simplify and avoid contingent price difference, we assume that every test case in our approach has the same investment-one unit and, similarly, every fault has one unit return. Thus we respectively transform test cases and faults to testing investments and returns. In NS model,  $m$  refers to testing investments and  $R(m)$  stands for testing returns. Furthermore,  $\beta_0$  which is a constant, can be regarded as the maximum expected long-term yields in testing activity.  $\beta_1$  and  $\beta_2$  respectively relate to the short-term and medium-term yields of testing, namely, the faster the curve reaches the stable limit, the faster the returns are gained; the more convex the curve is, the greater the returns are. Since  $\tau$  controls the extreme value point of the loading to which  $\beta_2$  attaches and  $\beta_2$  has a completely positive correlation with the curvature of NS model curve,  $\tau$  refers to the point with extreme yields, which can be treated as the primary testing stopping point.

Nevertheless, the basic problem is whether the NS model can be applied to testing activities as we assumed. Our experiments of curve fitting described in Section 3, used non-linear least square method on two programs with more than 51,000 lines of codes, and showed that approximately 83% percent of the values of correlation coefficient  $R^2$  are beyond 0.9. This result verifies our assumption that NS model can be used in testing.

## 3 Experiment

### 3.1 Subject Programs

Two programs are used in our empirical study: NanoXML (version 2)<sup>1</sup> and Checkstyle (version 5.3)<sup>2</sup>. NanoXML is a small XML parser for Java and it is an easy-to-use, non-GUI based system, which can be built from source without external libraries. Checkstyle is a development tool to help programmers write Java code that adheres to a coding standard. The version of each program is selected randomly. These two programs are both real world open source programs and have self-validating test cases available. The details of subject programs are listed in Table 1.

### 3.2 Test Cases

Test cases are downloaded together with corresponding programs and they are all JUnit test cases. One test case has

<sup>1</sup><http://sir.unl.edu/portal/index.php>

<sup>2</sup><http://checkstyle.sourceforge.net>

**Table 1. Subject Programs**

Program	NanoXML	Checkstyle
LOC	7646	43407
# Branches	26	2449
# Faults	7	687
# Test Cases	214	162

been split into several smaller ones according to every independent testing method and the effectiveness of original test suite will not change since no test method has been added or removed.

In addition, the tool CodeCover<sup>3</sup> has been equipped on the source codes to help collect coverage information of a System Under Test (SUT). CodeCover is an open source testing tool which can measure statement, branch and MC/DC coverage. The information of branches and conditions can be accessed by the intermediate files generated by CodeCover. Thus we can generate coverage vector of each test case: respectively, use 0 and 1 to represent certain statement or branch is covered or not.

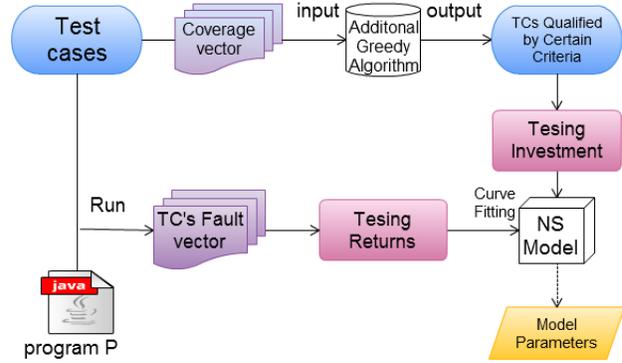
Then Additional Greedy Algorithm is utilized to pick test cases satisfied statement-coverage or branch-coverage criteria. The fundamental logic of Additional Greedy Algorithm is to select the next test case that covers the maximum number of statements or branches that have not yet been covered in past selections. The reason why we choose Additional Greedy Algorithm is that it has been proven of high effectiveness in a series of coverage-based test case prioritization techniques [13]. In addition, some other greedy algorithms, such as 2-Optimal Algorithm, Hill Climbing, Genetic Algorithms and so forth, cannot outperform significantly than the Additional Greedy Algorithm, which is considered as the cheaper-to-implement-and- execute algorithm [14]. Therefore, we chose to implement this algorithm.

### 3.3 Faults

Faults in NanoXML are all original faults while in Checkstyle are mutations. In [8], Andrews et al. verifies the feasibility of using mutation instead of real faults. Mutations of Checkstyle are generated by the tool Jumble<sup>4</sup>. Jumble is a class level mutation testing tool which is integrated with JUnit. The number of mutants depends on the number of test cases but not the size of programs since Jumble is associated with JUnit. Testing reports are produced by Jumble automatically so that we can extract the data and then set up fault vectors: using 0 and 1 to refer that this fault has been detected or not, the same principle with coverage vectors.

<sup>3</sup><http://http://codecover.org>

<sup>4</sup><http://jumble.sourceforge.net/>

**Figure 1. Framework of the Experiment**

### 3.4 Curve Fitting of Nelson-Siegel Model

The numbers of test cases which are qualified by random, branch-coverage or statement-coverage criterion are counted and recorded as the variable  $m'$ . Then  $m'$  is converted to the independent variable  $m$  followed the principle that one test case merits one unit in investment. So does the corresponding number of faults-detected to dependent variable  $R(m)$ . The nonlinear least square method is performed to fit the NS model curve, then the value of correlation coefficient  $R^2$  and parameters to be estimated, namely  $\beta_0$ ,  $\beta_1$ ,  $\beta_2$  and  $\tau$ , are recorded.

### 3.5 Experiment Setup

The main process of experiments we design and conduct is shown in Fig.1, which contains the following steps: (1) Instrument subject programs using CodeCover; (2) Run test cases of subject programs against instrumented version; (3) Collect coverage information to generate coverage vectors, including the granularities of statement and branch; (4) Run test cases of subject programs; (5) Collect mutation information and generate fault vectors; (6) Pick test cases which satisfy random, branch-coverage or statement-coverage criteria through Additional Greedy Algorithm; (7) Convert data from step 5 and 6 to testing investments and returns; (8) Operate curve fitting on NS model using the statistic data from step 7.

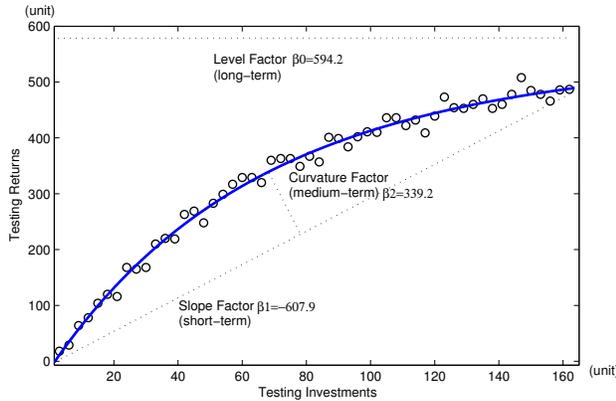
### 3.6 Results

The values of parameters  $\beta_0$ ,  $\beta_2$ ,  $\tau$  and correlation coefficient  $R^2$ , and the the absolute value of  $\beta_1$  are listed in Table 2. R, S and B respectively represent random, statement-coverage and branch-coverage criterion.

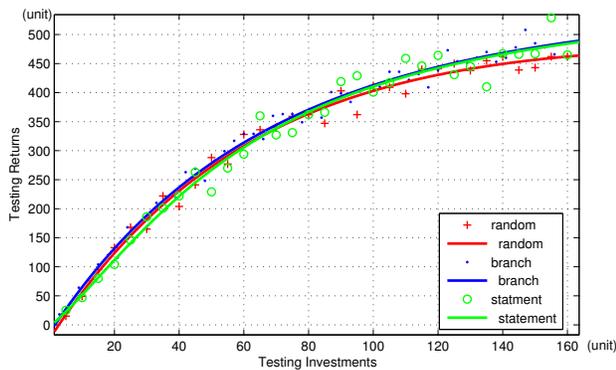
In Table 2, it is observed that 5 of 6 correlation coefficient  $R^2$  are greater than 0.94 and the rest is 0.846. In Fig. 2, the detail of the curve fitting on Checkstyle is depicted

**Table 2. Values of Parameters in NS Model and Correlation Coefficient  $R^2$**

Program	Criterion	$R^2$	$\beta_0$	$ \beta_1 $	$\beta_2$	$\tau$
Checkstyle	R	0.988	411.40	435.3	759.5	69.57
	B	0.991	594.20	607.9	339.2	55.81
	S	0.977	613.80	616.9	-429.2	19.82
NanoXML	R	0.846	6.86	8936.0	8902.0	1.01
	B	0.959	7.21	7.4	-8.6	1.66
	S	0.941	7.39	6.3	-13.1	14.95



**Figure 2. Values of Parameters in Checkstyle, Branch-Coverage Criterion**



**Figure 3. Curve Fitting Result of Checkstyle**

as an example. And Fig. 3 marks the parameters of curve fitting on Checkstyle branch-coverage criterion testing. For the limited space, only Checkstyle is shown.

The maximum expected long-term yields component  $\beta_0$  tends to be the smallest in random testing and the largest in statement-coverage criterion testing for both programs. For Checkstyle, the absolute value of short-term component  $\beta_1$  has a regulation that  $S > B > R$  and the parameter  $\tau$ , related to the extreme yields point, appears to be  $S < B < R$ . Inter-

estingly, in NanoXML,  $\beta_1$  and  $\tau$  present totally reversed trends. At last, the medium-term yields component  $\beta_2$  are positive in R, negative in S for both programs and has different sign in B.

### 3.7 Threats to validity

As a preliminary experiment, only two subject programs are involved in our study, which can be extended in the future. However, these subject programs, which are of reasonable scale, are widely used in other studies, and are representative to draw meaningful conclusions.

## 4 Discussion

### 4.1 Applicability of NS Model in Testing

From the experiment results in Section 3 that 5 of 6 correlation coefficient  $R^2$  are greater than 0.94 and the rest is 0.846, it can be safely concluded that the NS model is able to fit extremely well on the relationship between testing investments and returns. In other words, our introduction of NS model to testing is feasible and all of the following respective discussions of four parameters have reasonable and reliable fundament.

### 4.2 The Parameter $\beta_0$

As mentioned in Eq.4, the parameter  $\beta_0$  is the stable limit of  $R(m)$  and it means the maximum expected long-term yields in software testing. The experiment results show that, when investments are saturated, statement-coverage outperforms branch-coverage criterion and the latter outperforms random criterion in long-term yields. Comparatively, in most of previous researches, branch-coverage criterion tends to detect more faults than statement-coverage criterion since it has fine-grained requirements on coverage [9] [10]. However, stricter requirements naturally mean more test cases which lead to more investments. Compared with branch-coverage (high investments and high returns) and random (low investments and low returns), statement-coverage criterion which acquires a balance turns out to be the greatest yields choice.

### 4.3 The Parameter $\tau$

As explained in Section 2, the parameter  $\tau$  refers to the maximum yields point which can be treated as the primary testing stopping point. In Checkstyle, the value of  $\tau$  is  $S < B < R$ , which means the statement Coverage criterion will reach the extreme yields point at first, followed with

branch and random. However, it is interesting that the order of the value of  $\tau$  is just reversed in NanoXML. Considering the difference of program scale and the number of faults between Checkstyle and NanoXML, the reason might be that there are just 7 faults in NanoXML such that random test cases can detect them easily. At the same time, NanoXML has 7,646 lines of codes which means plenty of test cases are required to satisfy statement-coverage requirement, among which a large percentage is non-fault-detected. So the yields of statement-coverage criterion in NanoXML come later. On the contrary, Checkstyle has as many faults as 687 and its ratio of LOC to the number of faults is pretty smaller than the one in NanoXML. In other words, faults in Checkstyle are much more saturated than faults in NanoXML. So the advantage of random criterion that innately has the even-distributed probability in covering each fault, fades away and more coverage of statements stand out. To sum up, when testing investment is preliminary, it is recommended to use random criterion in fault-unsaturated programs while statement-coverage criterion in relatively fault-saturated programs, according to the parameter  $\tau$ .

#### 4.4 The Parameter $\beta_1$

The prophase yields of testing is positively related to the primary testing stopping point. That means the short-term component  $\beta_1$  should present consistent meaning with  $\tau$  and our experiment results just verified it. In terms of the absolute value of  $\beta_1$ , the bigger it is, the greater the short-term yields are. So in Checkstyle, the absolute value of  $\beta_1$  is  $S > B > R$ , which means the prophase yields is  $S > B > R$ . It is indeed consistent with the appearing order of the primary testing stopping points. And this consistency is also suitable in NanoXML.

#### 4.5 The Parameter $\beta_2$

The medium-term loading that  $\beta_2$  attaches is a function that starts out at zero, increases at medium maturities and decays to zero. Therefore,  $\beta_2$  is designed to create a hump-shape. Similar to the meaning of  $\beta_1$ , the medium-term component  $\beta_2$  has a positive correlation with the medium-term yields of testing. In our experiment, the value of  $\beta_2$  is positive in R, negative in S for both programs and has different signs in B. As we all know, random criterion is unconcerned with the structure of programs, so it displays a stable medium-term yields. In addition, it is supposed that the fluctuation in medium-term yields causes the negative value of  $\beta_2$ . Studies about  $\beta_2$  could be paid further efforts.

Another interesting observation in our experiment results is that in all of the yields comparison according to four parameters, branch-coverage criterion always lies in the inter-

mediate position, namely it performs neither the best nor the worst. Therefore, if certain testing situation needs a trade-off among short-term, medium-term, and long-term yields, branch-coverage criterion will be a preferable choice.

## 5 Conclusion

Testing is as an investment. Focusing on the business issues, investments and returns, rather than technical issues of software testing, we firstly introduce the model Nelson-Siegel to explore a novel and practical perspective of testing. Then, an experiment is conducted on two programs, with more than 51,000 lines of codes and nearly 700 faults, and the results indicate that the NS model can fit well on the relationship between investments and returns. Some useful observations are summarized as follows.

- Statement-coverage criterion has stronger ability to earn long-term yields than branch-coverage and random criterion.
- For large-scale programs with saturated faults, statement-coverage criterion tends to reach primary testing stopping point at first and has the best prophase yields, while random criterion does the best in small-scale programs with unsaturated faults.
- Random criterion outperforms the other two criteria in the medium-term yields of testing.
- Branch-coverage criterion can be a trade-off in choice of combining all of the short-term, medium-term, and long-term yields.

## 6 Future Work

Our experiment is preliminary but encouraging. The prospect of testing as an investment and the application of the NS model are promising, furthermore, there are some aspects that could be improved and studied in the future.

- Some parameters of the NS model should be confined within a concrete range to exclude non-practical situation such as negative investments.
- The transformation from testing data to economic data should be studied. To be more practical, the cost should take generation, execution and inspection of testing into calculation. Faults should be classified based on significance and type like GUI fault or logic fault. Practitioners who use this NS model can adjust the indices according to given programs.
- A base line can be depicted which means no gain and no lose to be compared with certain testing curve to figure out profit or deficit.

- NS model can be used to predict the risk in a released version. The number of expected-detected faults derived from  $\beta_0$  subtracts the number of have-already-detected faults equals the number of have-not-detected faults, which remain in codes. In other words, we have already got current yields and could predict the stable long-term yields, so the gap between them is the risk remained in a released version.

## Acknowledgment

The work described in this article was partially supported by the National Basic Research Program of China (973 Program 2014CB340702), the National Natural Science Foundation of China (No. 61373013, 61170067) and the Scientific Research Foundation of Graduate School of Nanjing University (2013CL13).

## References

- [1] S. Biffi, A. Aurum, B. Boehm, H. Erdogmus, and P. Grünbacher, “Value-based software engineering”, Springer Verlag, 2005.
- [2] G. Rothermel, R. Untch, C. Chu, and M. Harrold, “Test case prioritization: An empirical study”, in *Proceedings of IEEE International Conference on Software Maintenance (ICSM’99)*, pp. 179–188, 1999.
- [3] H. Erdogmus, J. Favaro, and W. Strigel, “Return on investment”, *IEEE Software*, vol. 21, no. 3, pp. 18–22, 2004.
- [4] D. Rico, “ROI of software process improvement: Metrics for project managers and software engineers”, J. Ross Publishing, 2004.
- [5] C. Nelson and A. Siegel, “Parsimonious modeling of yield curves,” *The Journal of Business*, vol. 60, no. 4, pp. 473–489, 1987.
- [6] F. Diebold and C. Li, “Forecasting the term structure of government bond yields”, *Journal of Econometrics*, vol. 130, no. 2, pp. 337–364, 2006.
- [7] F. Diebold, M. Piazzesi, and G. Rudebusch, “Modeling bond yields in finance and macroeconomics”, National Bureau of Economic Research, Tech. Rep. 05-008, 2005.
- [8] J. Andrews, L. Briand, Y. Labiche, and A. Namin, “Using mutation analysis for assessing and comparing testing coverage criteria”, *IEEE Transactions on Software Engineering*, vol. 32, no. 8, pp. 608–624, 2006.
- [9] P. Frankl and S. Weiss, “An experimental comparison of the effectiveness of branch testing and data flow testing”, *IEEE Transactions on Software Engineering*, vol. 19, no. 8, pp. 774–787, 1993.
- [10] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand, “Experiments of the effectiveness of dataflow- and controlflow-based test adequacy criteria”, in *Proceedings of the 16th International Conference on Software Engineering (ICSE’94)*. IEEE Computer Society Press, pp. 191–200, 1994.
- [11] A. Zaidman, B. Rompaey, S. Demeyer, and A. Deursen, “Mining software repositories to study co-evolution of production and test code”, in *Proceedings of the 1st International Conference on Software Testing, Verification, and Validation (ICST’08)*, pp. 220–229, 2008.
- [12] S. Sinha, A. Orso, and M. J. Harrold, “Automated support for development, maintenance, and testing in the presence of implicit flow control”. in *Proceedings of the 26th International Conference on Software Engineering (ICSE’04)*, pp. 336–345, 2004.
- [13] G. Rothermel, R. H. Untch, C. Chu and M. J. Harrold, “Prioritizing test cases for regression testing”, *IEEE Transactions on Software Engineering*, vol. 27, no. 10, pp. 929–948, 2001.
- [14] Z. Li, M. Harman, and R. M. Hierons, “Search algorithms for regression test case prioritization”, *IEEE Transactions on Software Engineering*, vol. 33, no. 4, pp. 225–237, 2007.
- [15] S. Yoo and M. Harman, “Regression testing minimization, selection and prioritization: a survey”, *Software Testing, Verification and Reliability*, vol. 22, no. 2, pp. 67–120, 2012.
- [16] S. Sampath, R. Bryce and A. Memon, “A Uniform Representation of Hybrid Criteria for Regression Testing”, *IEEE Transactions on Software Engineering*, vol. 39, no. 10, pp. 1326–1344, 2013.
- [17] W. E. Wong, J. R. Horgan, S. London and H. Agrawal, “A study of effective regression testing in practice”, in *Proceedings of the 8th International Symposium on Software Reliability Engineering (ISSRE’97)*, pp. 264–274, 1997.
- [18] Q. Yang, J. J. Li and D. M. Weiss, “A survey of coverage-based testing tools”, *The Computer Journal*, vol. 52, no. 5, pp. 589–597, 2009.
- [19] K. E. Emam, “The ROI from software quality”, CRC press, 2005.

# A Proposal for the Improvement of Project's Cost Predictability using Earned Value Management and Historical Data of Cost – An Empirical Study

Adler Diniz de Souza, Ana Regina Rocha

COPPE/UFRJ - Universidade Federal do Rio de Janeiro  
Programa de Engenharia de Sistemas e Computação  
Av. Horácio Macedo, 2030, Prédio do Centro de Tec.,  
Bloco H, Sala 319, Caixa Postal 68511 – CEP 21941-914  
– Rio de Janeiro, RJ  
[adlerunifei@gmail.com](mailto:adlerunifei@gmail.com)  
[darocha@centroin.com.br](mailto:darocha@centroin.com.br)

Djenane Cristina Silveira dos Santos

UNIFEI - Universidade Federal de Itajubá  
Ciência da Computação  
Av. BPS, 1303, Pinheirinho, Instituto de Ciências Exatas,  
Departamento de Matemática e Computação, CEP 37500-  
903 – Itajubá, MG  
[djenane.cris@gmail.com](mailto:djenane.cris@gmail.com)

**Abstract**—This paper proposes an extension of the EVM technique through the integration of historical cost performance data of processes as a means to improve the predictability of projects cost. The proposed technique was evaluated through an empirical study, which evaluated the implementation of the proposed technique in 17 software development projects. The proposed technique has been applied in real projects with the aim of evaluating the accuracy and variation of the  $CPI_{Acum}$  and consequently the EAC. Then it was compared to the traditional technique. Hypotheses tests with 90% significance level were performed, and the proposed technique was more accurate and more stable (less variation) than the traditional technique for calculating the Cost Performance Index - CPI and the Estimates at Completion - EAC.

**Keywords:** *Earned Value Management, Cost Performance Index - CPI, Measurement and Analysis, High Maturity*

## I. INTRODUCTION

Currently, PMI estimates that approximately 25% of global GNP is spent on projects and that about 16.5 million professionals are directly involved in project management worldwide. This volume of projects and changes in the global scenario, increasingly competitive, generate the need for faster results, with higher quality, lower costs and shorter times [13].

To assess whether or not a project will achieve its cost and time objectives, various measures are collected during its execution, and various performance indexes are produced and periodically analyzed. When deviations above tolerable are found in some performance index, corrective actions are implemented in order to improve them. Among the main techniques to analyze cost and time performance, the Earned Value Management - EVM - is considered the most reliable [9].

EVM is a technique that integrates scope, time and cost data for measuring project performance and predicts its final cost and time based on the current team performance. The technique gained great importance when, in 1967, the U.S Department of Defense started requiring its use as a means to control the costs of contracted projects. [10].

Several formulae derived from EVM measurements are available and have been studied in the past 15 years [9].

However, traditional management methods are not sufficient to predict the performance of processes involved in current and future projects. [4].

Particularly in Software Engineering some model references like CMMI-Dev [15] and ISO/IEC 12207 [8] requires the gathering of measures and a development of indicators of the most important processes, responsible for achieving the business goals of organization.

This paper proposes an improvement in the EVM, it can be used like a performance model to predict the final cost of software projects, based in the CPI historical data of some processes.

## II. EARNED VALUE MANAGEMENT – EVM

The earned value management technique allows the calculation of variances and performance indices of cost and time, which generate forecasts for the project, given its performance to date, making the implementation of actions aimed at correcting any deviations possible. This allows manager and project team to adjust their strategies, make balances based on goals, in the current performance of the project, in trends, as well as in the environment in which the project is being conducted [1].

According to [12], EVM plays a crucial role in the success of projects, responding to managerial issues that are considered critical, such as: i) how efficiently are we using our time?, ii) when the project will be finalized? iii) ) how efficiently are we using our resources? iv) how above or below the budget it will be at the end of the project, given the current productivity of the team?

EVM is based on three basic measures, which are derived to generate other measures and performance indexes. These basic measures are: i) Planned Value -  $PV_{Acum}$ : which represents the Planned Costs accumulated up to a certain date, ii) Earned Value -  $EV_{Acum}$ : which represents the Budgeted Cost of Work Performed by a certain date, and iii) Actual Cost -

$AC_{Acum}$  : which represents the Actual Cost of the Work Performed by a certain date. [12].

The basic measures discussed do not allow making cost and time predictions to complete the project and answer the questions previously shown. For this, it is necessary to generate performance indexes, among which the most used are Schedule Performance Index -  $SPI_{Acum}$  and Cost Performance Index -  $CPI_{Acum}$ .

The SPI is an indicator of actual progress compared to planned progress of a project [13]. It shows how efficiently the project team is using their time [1], and is calculated by:

$$SPI_{Acum} = \frac{EV_{Acum}}{PV_{Acum}}, \quad (1)$$

CPI is a measure of the work performed compared to the actual cost or progress achieved in the project. [13]. It shows how efficiently the project team is using their resources [12], and is calculated by:

$$CPI_{Acum} = \frac{EV_{Acum}}{AC_{Acum}}, \quad (2)$$

$CPI_{Acum}$  is considered the most critical EVM index because it measures the cost efficiency of the work performed [13], and can be used to provide a cost projection.

As the project progresses, the project team can develop a forecast for the Estimate At Completion - EAC, which may be different from the Budget At Completion - BAC, based on project performance [18]. EAC provides the final cost estimation and is given by the following equation (where the premise is that the cost performance will remain the same):

$$EAC = \frac{BAC}{CPI_{Acum}}, \quad (3)$$

Thus, the  $CPI_{Acum}$  is used to two purpose: i) to measure the cost efficiently (only analyzing the index), or to ii) make cost projections, thought EAC (see equation 3).

The Figure 1 illustrates the measures and indexes discussed as well as the projections that may be made from indexes shown and Table 1 illustrates the other components of the earned value management technique that were not discussed and / or will not be used in the context of this work, but are important to your understanding.

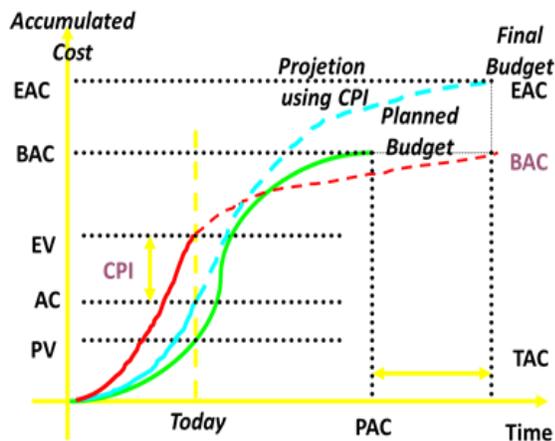


Figure 1 –Measures and performance indexes

TABLE 1 –OTHER EQUATIONS AND DEFINITIONS OF THE EVM TECHNIQUE

Equation	Definition
$BAC = \sum PV$	Budget At Completion, represents the baseline of the project cost.
SAC or PAC	Schedule At Completion or Plan At Completion, represents the final time to complete the project.
$TAC = PAC / SPI_{Acum}$	Time At Completion represents the time projection calculated based on SPI.
$EAC = AC_{Acum} + BAC - EV_{Acum}$	When the premise is that $CPI_{Acum}$ is equal to 1.
$EAC = AC_{Acum} + [(BAC - EV) / (CPI_{Acum} \times SPI_{Acum})]$	When one desires to consider both $CPI_{Acum}$ and $SPI_{Acum}$ indexes

Related works are presented by Solomon [17] that evaluated which elements of the EVM technique has corresponding specific practices in the CMMI. These elements were then used to improve the implementation of specific practices of process areas in the levels 2 and 3 of the CMMI. Solomon [18] also proposed an extension of EVM that can be applied to IT projects. He used quality data of projects to improve its cost projection.

### III. PROBLEM DESCRIPTION

The Earned Value Management technique makes use of the CPI to make cost projections at the end of the project. This index is the subject of several discussions on its applicability and reliability to make projections, as reported in works carried out by [2], [6], [9] and [24].

The major focus of the discussion is the  $CPI_{Acum}$  stability. Finding out whether the  $CPI_{Acum}$  is stable or not is important because it is used to make cost projections ( $EAC = BAC / CPI_{Acum}$ ).

According to [3], stability can be defined as a state of statistical control that provides with a high degree of confidence, the performance prediction of some variable in the immediate future.

Florac [5] states that the stability of a process is considered by many as the core of the management of processes, and it is essential for companies to produce products according to what has been planned and to improve processes in order to produce better and more competitive products.

A study reported in [2] evaluated the  $CPI_{Acum}$  stability of several projects of the Department of Defense (DoD), and found that the index was stable after 20% of project execution. This study [2] generalized the result, concluding that any project could use the EVM technique reliably, after 20% of project execution. This information was used as a criterion for retaining or canceling projects in the U.S. government, which showed  $CPI_{Acum}$  below 0.9 after 20% of project execution, because according to the study, the stability of the index was evidence that a project with poor CPI was unrecoverable (note that the upper and lower limits defined here, by DoD, represents a specification limit. They are not a natural limit of

the process. Thus, the stability of the  $CPI_{Acum}$  is discussed here in terms of the customer or as specification limit).

However, several other studies have questioned the generalization of these results in different contexts (projects developed outside the scope of DoD), and showed different results, i.e., they showed instability in cost performance indexes for most of the project [6], [9], [21], [24].

Claiming that the  $CPI_{Acum}$  is unstable and varies widely during the execution of a project avoids making accurate projections of cost estimate at the end of the project (EAC), unless one knows or has any expectation that this variation is due to factors already known.

The proposal of improving the earned value management technique presented in the next section is based on the premise that any project is composed of a set of processes, which have different performances. This assumption was confirmed by several studies reported in [5], [14] and [11] and by results shown in Table 2.

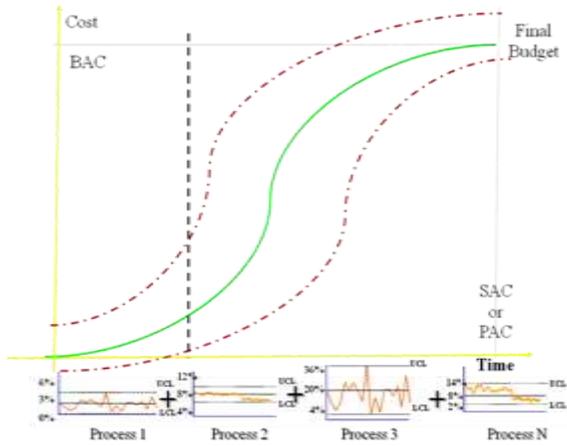


Figure 2 – Justification for the CPI instability

Thus, one reason for the  $CPI_{Acum}$  instability is the natural variation of the processes used, as illustrated in Figure 2, which shows that the  $CPI_{Acum}$  is actually the result of the performance of different processes, thus, it is not expected that the  $CPI_{Acum}$  is constant or equal to 1, but that varies due to the implementation of each specific process. Thus, it is possible to compose an equation that considers the historical performance data of processes, which have not yet been executed in the present project, (but will be futurely executed in the project), increasing the cost predictability, even for projects with unstable  $CPI_{Acum}$ .

#### IV. PROPOSAL OF THE HISTORICAL EVM

The  $CPI_{Acum}$  can be used with two purposes: i) to measure the cost efficiency (only analyzing the index), or ii) to make cost projections through EAC (see equation 3). This proposal shall be used only to perform cost projections using the EAC.

If it is obvious that in the Budget At Completion - BAC is no longer viable, the project manager must prepare an Estimate at Completion - EAC. Developing an EAC forecast involves finding estimates or forecasts of future events and conditions for the project based on information and knowledge available

at the time of prediction. Information on work performance includes past performance of the project and any information that could impact it in the future [13].

This paper proposes the integration of the EVM with  $CPI_{Acum}$  historical data of processes. This integration consists in gathering and using the  $CPI_{Acum}$  historical data of each process of the software lifecycle, with traditional measures of the EVM technique, calculated separately to each process.

The  $CPI_{Acum}$  historical performance of the processes is important because it will be used to predict the future behavior of the project  $CPI_{Acum}$ , which consists of the individual performance of each process.

Thus, given its performance and each EVM individual measure of processes, it is possible the equation below to calculate the  $CPI_{Acum}$  projected to the final of project execution:

$$CPI_{Exp} = \frac{EV_{AcumProject} + \sum_1^N (BAC_{PN} - EV_{AcumPN}) + \sum_1^N BAC_{PN}}{AC_{AcumProject} + \sum_1^N (AC_{ExpectedPN} - AC_{AcumPN}) + \sum_1^N (AC_{ExpectedPN})} \quad (4)$$

Where:

- **$EV_{AcumProject}$  and  $AC_{AcumProject}$ :** are respectively the traditional  $EV_{Acum}$  and  $AC_{Acum}$  of the project, that can be calculated using the traditional EVM equations;
- **$BAC_{PN}$ :** can be calculated adding every PV activities of the process. It can be calculated using the equation below:

$$BAC_{PN} = \sum_1^N PV \text{ Activity } N \text{ of Process } PN \quad (5)$$

- **$EV_{AcumPN}$  and  $AC_{AcumPN}$ :** are respectively the  $EV_{Acum}$  and  $AC_{Acum}$  of each process. It can be calculated adding every EV and AC of executed activities of the process, like the equation (5) presented previously;
- **$AC_{ExpectedPN}$ :** is the  $AC_{Acum}$  expected by each process after it be executed. The  $AC_{ExpectedPN}$  use the historical  $CPI_{Acum}$  of each process and can be calculated using the equation below:

$$AC_{ExpectedPN} = \frac{BAC_{PN}}{Historic \text{ CPI of } PN} \quad (6)$$

This equation represents an evolution in relation to the preliminary first version that was evaluated through project simulations and the results were presented in [19] and [20]. Now it new version was evaluated through an empirical study. The methodology used in the empirical study and the results will be presented in the next sections.

#### V. PREPARATION

Measures from 17 software development projects were collected among March 2009 and January 2010. According to [4], in addition to collecting basic measures, information on the characteristics of projects must be recorded, such as total estimated size, programming language used, profile of the project team, development environment and process version used. This information will allow grouping the measures collected into different project categories, maintaining

homogeneity among members of each group. If the group is not homogeneous, the analyses can be compromised and lead to inappropriate conclusions.

Thus, projects that were part of this study had the following characteristics:

- They were from the same client, which was a multinational telecommunications company,
- They used a lifecycle model, which included 4 processes, namely: i) the Elaboration of Use Case Tests - UCT, ii) the implementation of functional requirements - IMP, iii) testing of these functional requirements - TES, using test cases produced and iv) correction of reported errors - COR;
- They were developed using the same technology (MS Visual Basic and Active Server Pages) and
- They were developed by professionals of similar profiles, which were interspersed among the projects.

All size, effort and cost estimates of projects evaluated were performed using the Use Cases technique [16] after the development and validation of documents of use cases, approved by the client and the development team.

As the largest cost component in a software project are the hours required for product development, all the basic measures and traditional EVM indexes were calculated based on estimated hours and actual hours, measured after the execution of activities.

For each activity planned in the projects, planned costs - PV (through the estimated effort for the execution of the activity) and actual costs (through real effort calculated after performing the activity) were calculated. Based on this information and on the project progress, CPI's<sub>Acum</sub> for processes and for the project as a whole were calculated.

TABLE 2 – PROJECT INFORMATION

Projects	CPI <sub>Acum</sub> of Processes				Periods	
	UCT	IMP	TES	COR	Period	
Project 01	2,02	1,48	1,62	2,95	11/03/09 a 01/04/09	P1
Project 02	2,48	4,46	3,43	2,98	16/03/09 a 06/04/09	
Project 03	1,84	-	2,71	1,53	23/03/09 a 16/04/09	
Project 04	1,75	1,14	1,85	7,05	26/03/09 a 17/04/09	
Project 05	3,90	4,30	1,06	1,06	20/04/09 a 19/05/09	
Project 06	1,54	-	1,50	1,85	20/04/09 a 19/05/09	
Project 07	1,46	1,22	1,08	1,61	20/04/09 a 13/06/09	
Project 08	1,98	1,61	1,81	2,17	29/04/09 a 15/05/09	
Project 09	-	2,43	1,43	1,00	29/04/09 a 20/05/09	
Project 10	1,98	1,77	2,39	2,14	21/05/09 a 09/06/09	
Project 11	2,20	1,67	2,20	10,0	15/06/09 a 30/06/09	P3
Project 12	1,19	1,23	4,34	2,50	29/06/09 a 10/07/09	
Project 13	3,31	2,32	3,10	N.D.**	29/07/09 a 10/08/09	P4
Project 14	2,24	4,37	2,40	1,85	11/08/09 a 20/08/09	
Project 15	1,01	1,48	-	1,54	12/08/09 a 20/08/09	
Project 16	-	1,90	5,25	3,44	21/08/09 a 04/09/09	P5
Project 17	3,78	1,33	3,08	2,57	01/09/09 a 18/09/09	

(\*) Note: data of CPI<sub>Acum</sub> of: i) Elaboration of Use Case Test, ii) Implementation of requirement, iii) Test and iv) Correction of bugs, respectively.

(\*\*) Note: Non Defined – N.D. There was no bug in this project and consequently no effort to fix them.

The CPI<sub>Acum</sub> determined at the end of execution of each process of the lifecycle of projects, as well as the duration of each project is shown in Table 1.

The proposed technique uses CPI<sub>Acum</sub> historical data in each utilized process in a software-lifecycle model to perform safer CPI<sub>Acum</sub> projections. Projects participating in this study were executed on different dates, and therefore were considered different periods for performing statistical analyses and for data availability for the performance of projections using this technique. During the study, each specific period used the average CPI<sub>Acum</sub> of the processes of projects previously executed.

## VI. VALIDATION TECHNIQUE

One of the aims of this study was to answer the following question: "Does the earned value management traditional technique have higher EAC accuracy and lower CPI variation than the earned value management with historical performance?"

Variation means that repeated measurement values are grouped together and exhibit little dispersion. According to [13] accuracy means that the measured value is very close to the correct value.

Thus, to measure the accuracy of the techniques, the EAC (Estimate At Completion) of each technique in each activity of the projects evaluated was compared with the AC (Actual Cost) measured at the end of the project execution using the equation below:

$$Error_{EAC \text{ Activity}(N)} = \frac{EAC_{Activity(N)}}{AC_{Final}}, \quad (7)$$

The average error shown by each technique in relation to the final AC calculated was also evaluated, using the equation below:

$$Average \text{ Error}_{EAC} = \frac{\sum_{i=1}^N \text{Error}_{EAC \text{ Activity}(N)}}{N} \quad (8)$$

To measure the variation of the techniques, the CPI<sub>Acum</sub> variation was evaluated, which was calculated by the techniques, in relation to the last CPI<sub>Acum</sub> estimation, or, how much the CPI<sub>Acum</sub> estimation varied in relation to the previous one using the equation below:

$$\text{Variation}_{CPI_{Activity}(N)} = \frac{CPI_{Activity(N)}}{CPI_{Activity(N+1)}} \quad (9)$$

The variation was measured by project activities, and for the hypotheses testing, the average variation of projects was calculated using the equation below:

$$\text{Average Variation} = \frac{\sum_{i=1}^N \text{Variation}_{CPI_{Activity}(N)}}{(N-1)}, \quad (10)$$

The EAC Error and CPI variation was calculated to both technique (traditional and the proposed technique).

Thus, the following hypotheses were established to evaluate the accuracy of the techniques:

- **H0<sub>Accuracy</sub>**: the traditional earned value management technique provides accuracy equal to the traditional earned value management technique with historical performance. ( $Error_{EAC_{Trad.}} - Error_{EAC_{Hist.}} = 0$ )
- **H1<sub>Accuracy</sub>**: the traditional earned value management technique provides accuracy lower than the traditional earned value management technique with historical performance. ( $Error_{EAC_{Trad.}} - Error_{EAC_{Hist.}} > 0$ )

A similar hypothesis was identified and tested to evaluate the  $CPI_{Acum}$  variation in both techniques.

The techniques shown in section iv were assessed through an empirical study, in which the goal was to measure the variation and accuracy of both techniques and compare them.

At the end of each time of execution, historical data of processes ( $CPI_{Acum}$  in each process as seen in the table 1) were collected, analyzed and used through the proposed technique to calculate a new performance cost index  $CPI_{Hist}$  (equations 4 to 6). The average error of each project showed in the figure 3, was calculated using the equations 7 and 8.

Figure 3 shows on X axis each of the 8 projects evaluated, and on Y axis, the average errors of  $EAC_{Hist.}$  and  $EAC_{Trad.}$  for each project. The gain of accuracy using the proposed technique was 57.7% compared to the traditional technique in project 16, represented as point 7 in Figure 3, and 11.3% lower than the traditional technique in project 13, represented as point 04, also in Figure 3.

No errors were collected from the 9 first projects, since they were supposed only to form the historical-data basis to perform index projections for the second period projects.

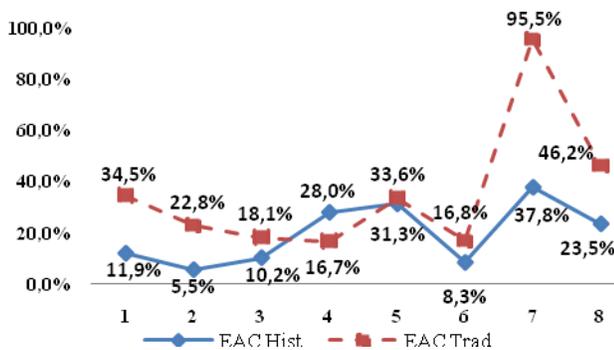


Figure 3 – Average error (Accuracy of techniques)

The project 13 (point 4 in the Figure 3) presented the worst accuracy among the evaluated projects, when the proposed technique was used. This result occurred because the “Correction” process was not executed, since the project did not have any bug (see Table 2). Since the process was not executed, the  $AC_{Final}$  of the project was lower than expected, and consequently the  $CPI_{Acum}$  was greater than expected. The worst result displayed by the proposed technique in this project was caused by an abnormal behaviour in a specific process.

Figure 4 show the average variation in the techniques. The average variation of each project showed in the figure 4, was calculated using the equations 9 and 10. It could be observed that the proposed technique showed no average variation

greater than the traditional technique in any project in any of the periods tested, even when the historical database was still small.

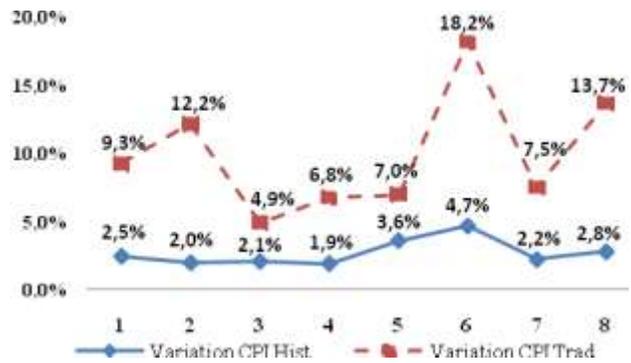


Figure 4 – Average Variation of  $CPI_{Acum}$

In attempt to evaluate the previously shown hypotheses, statistical tests based in the data of the figures 3 ( $EAC_{Trad}$  and  $EAC_{Hist}$ ) and 4 ( $CPI_{Trad}$  and  $CPI_{Hist}$ ) were performed to confirm that the differences in accuracy and variation found in applying the proposed techniques were significant. The Action tool was used to perform the hypotheses tests of T paired samples, with significance level of 90%.

TABLE 3 – ACCURACY AND VARIATION HYPOTHESIS TEST

Hypothesis	Tests	T	P	Conclusion
<b>H0<sub>Accuracy</sub></b>	$Error_{EAC_{Trad.}} - Error_{EAC_{Hist.}} = 0$	2,219	0,03	Refute H0
<b>H0<sub>Variation</sub></b>	$Variation_{CPI_{Trad.}} - Variation_{CPI_{Hist.}} = 0$	5,26	$5,8 \times 10^{-4}$	Refute H0

The analysis of data in table 3 and figures 3 and 4 allows inferring that the proposed technique provides greater accuracy in cost estimations, considering the average error of EAC, and lower variation in  $CPI_{Acum}$ .

## VII. VALIDITY THREATS

According to [22] the internal validity observes if the treatments really cause the expected results. In this study, the expected results are: i) decrease the  $CPI_{Acum}$  variability and consequently the EAC variability and ii) Decrease the error in the EAC estimate. Both expected results were achieved with the application of the proposed technique.

However, it's necessary to consider that, the technique was validated through an empirical study using industry data. This data were from only one software factory with similar projects.

The proposed technique suggests a similar scenario for your application (data of projects that were executed using defined and stable processes, and preferentially with the same technology). However it is important a more widely study with more companies, in different domain of application.

According to [22], the conclusion validity evaluates the statistical significance. The main problem in this study is the number of available projects to conduct the hypothesis test. This is a known problem in Software Engineering. Thus, the result cannot be considered conclusive, but just a clue that the technique works. Before using the technique the company it's

recommended to make a similar study, intending to determine if the proposed technique provides better results to its projects.

#### VIII. CONCLUSION

This paper described the evolution of the EVM technique. The proposed technique integrates historical data of cost performance as a mean to improve the project cost predictability. An empirical study was carried out based on data from 17 projects of a software factory with the aim of identifying whether the technique showed better accuracy and variation in CPI and EAC compared to the traditional technique. To evaluate the proposed technique, several hypothesis tests about the research questions shown in section vi were performed.

The empirical study showed that the proposed technique is more accurate and more stable (less variation) than the traditional technique, when using historical database with  $CPI_{Acum}$  of each lifecycle process. All hypothesis tests conducted with historical database composed of at least 9 projects showed significant results, at 90% significance level.

As future works, we propose the use of the proposed technique in other companies, with different contexts, in order to generalize the results obtained here.

#### IX. REFERENCES

- [1] ANBARI, F.T., 2003, "Earned Value Project Management Method and Extensions", *Project Management Journal*, v. 4, pp. 12.
- [2] CHRISTENSEN, D., SCOTT R. HEISE, 1993, "Cost Performance Index Stability", *National Contract Management Journal*, v. 25, pp. 7-15.
- [3] DEMING, W.E., 1993, *The New Economics for Industry, Government, Education* Cambridge Mass.: Massachusetts Institute of Technology, Center for Advanced Engineering.
- [4] FENTON, N., MARSH, W., CATES, P., FOREY, S., TAYLOR, M., *Making Resource Decisions for Software Projects*, Proceedings of the 26th International Conference on Software Engineering, Escócia, 2004.
- [5] FLORAC, W.A., A. D. CARLETON, 1999, *Measuring the Software Process: Statistical Process Control for Software Process Improvement*, Addison-Wesley.
- [6] HENDERSON, K., OFER ZWIKAEI, 2008, "Does Project Performance Stability Exist A Re-examination of CPI and Evaluation of SPI(t) Stability", *Cross Talk*.
- [7] IRANMANESH, H., N. MOJIR, S. KIMIAGARI 2007, "A new formula to "Estimate At Completion" of a Project's time to improve "Earned value management system", *International Journal of Project Management*.
- [8] ISO/IEC12207, 2008, "Systems and software Engineering - Software life cycle processes".
- [9] LIPKE, W., 2006, "Statistical Methods Applied to EVM: The Next Frontier", *Department of Defense - USA*, v. 19, pp. 32, June.
- [10] LIPKE, Z., ANBARI, HENDERSON, 2009, "Prediction of project outcome, the application of statistical methods to Earned Value Management and Earned Schedule performance indexes", *International Journal of Project Management*.
- [11] PFLEEGER, N.E.F.A.S.L., 1997, *Software Metrics: A Rigorous and Practical Approach* Boston, PWS Publishing Company.
- [12] PMI, 2005, *Practice Standard Earned Value Management* Pennsylvania, PMI.
- [13] PMI, 2009, *Project Management Body of Knowledge - PMBOK* Newton Square, Project Management Institute.
- [14] PUTNAM, L.H., 2003, *Five Core Metrics: The Intelligence Behind Successful Software Management*, Dorset House.
- [15] SEI, S.E.I., 2006, "CMMI® for Development (CMMI-DEV), V1.2, CMU/SEI-2006-TR-008", SEI.
- [16] SMITH, J. *The Estimation of Effort Based on Use Cases*. Rational Software, White paper. 1999.
- [17] SOLOMON, PAUL J., 2002, *Using CMMI to Improve Earned Value Management*, Technical Note CMU/SEI 2002-TN016, SEI.
- [18] SOLOMON, PAUL J., 2002, *Performance-Based Earned Value*, *Cross Talk The Journal of Defense Software Engineering*, V.18, p. 37 - 43.
- [19] SOUZA, A. D. ; ROCHA, A. R. C., 2012. A proposal for the improvement of the technique of Earned Value Management utilizing the history of performance data.. *Proceedings of the Twenty-Fourth International Conference on Software Engineering & Knowledge Engineering - SEKE*. p. 753-759.
- [20] SOUZA, A. D. ; ROCHA, A. R. C. . A proposal for the improvement the predictability of project cost using EVM and Historical Data of Cost. In *35th International Conference of Software Engineering-ICSE, 2013*, ACM SRC, San Francisco.
- [21] VANDEVOORDE, S., MARIO VANHOUCHE, 2006, "A comparison of different project duration forecasting methods using earned value metrics", *Project Management Journal*, v. 24, pp. 289 - 302.
- [22] WÖHLIN, C., RUNESON, P., HÖST, M., OHLSSON, M. C., REGNELL, B., WESSL, A., 2000. *Experimentation in software engineering: an introduction*. Kluwer Academic Publishers.
- [23] WHEELER, D.J.; CHAMBERS, D.S., 1992, *Understanding Statistical Process Control*, 2a. edição, SPC Press, Inc..
- [24] ZWIKAEI, O., ET AL, 2000, "Evaluation of Models for Forecasting the Final Cost of a Project." ", *Project Management Journal* v. 31.1 pp. 53-57.

# Evaluating the Use of Model-Based Requirement Verification Method: An Empirical Study

Munmun Gupta, Daniel Aceituna, Gursimran S. Walia, Hyunsook Do  
North Dakota State University, Fargo, ND, USA  
{munmun.gupta, daniel.aceituna, gursimran.walia, hyunsook.do}@ndsu.edu

**Abstract**—Requirements engineering is a critical phase in software development that describes the customer needs and the specifications for the software solution. Requirements are gathered through various sources and the output is a list of requirements for a software product to be developed, written in Natural Language (NL). NL requirements are fault prone because stakeholders can interpret NL differently due to the inherent imprecision, ambiguity, and vagueness of NL. To address these problems, a model-based requirements verification method called NL to state transition diagram (STD) is proposed. This paper evaluates the ability of the NLtoSTD method to detect faults when used on NL requirements and to improve the software reliability. Overall, the result shows that the NLtoSTD is an effective requirements verification method.

## I. INTRODUCTION

To ensure software reliability, it is important to detect and prevent different types of faults during the development of various software artifacts. Requirements are gathered from different stakeholders (technical and non-technical) and recorded in natural language (NL), that describes the customer needs and the specifications for the software solution. The output of this phase, a *software requirement specification* (SRS - a means of communication among stakeholders), is especially fault-prone due to the inherent imprecision, ambiguity, and vagueness of NL. Requirement faults if undetected propagate to the later phases where they are difficult to find and fix [1-3].

To ensure high-quality requirements, numerous fault-based verification approaches have been developed and validated for fault-detection effectiveness [3, 8-11]. In particular, *software inspection*, have been empirically validated [3, 9] for early detection of faults in software artifacts. However, it is estimated that the software development effort is still spent on fixing problems that should have fixed early in the lifecycle [1, 2]. This rework stems from the fact that inspectors can have different interpretations of the requirements and may not notice the ambiguities and inconsistencies among other problems.

Model based approach, if applied to NL requirements can be used for verification of NL specifications [6, 10]. However, building a model from NL requirements is highly subjective. Consequently, an erroneous translation of NL requirements can result in the wrong model due to the inherent incompleteness and ambiguities of NL [11] and, thus, can eventually produce software that stakeholders do not want. To address this, several researchers have proposed modeling techniques using an automated NL translation approach [4,6,10,12]. These methods include approaches based on translating goals to state machines [4], and scenarios to state machines [12]. Automation can certainly improve the translation process, but complete and

error-free automation of this process is not possible because, often, NL requirements can be interpreted in multiple ways..

To address this problem, we propose a method that translates NL requirements into a State Transition Diagram (STD) in an incremental manner (NLtoSTD) and expose ambiguities, incompleteness, and inconsistencies in NL requirements. The NLtoSTD is carried out in two steps, where the first step turns each NL requirement into a STD building block (*NLtoSTD-BB*) and the second step then construct the STD using the STD-BBs (*STD-BBtoSTD*). Requirements engineers and stakeholders can detect faults during each step (NLtoSTD-BB and STD-BBtoSTD) and direct mapping from NL to model is preserved in the translation process. Each NL requirement becomes a segment of the STD so that adjustments made to the model can be directly made to the requirements, and visa-versa. The results from the previous study [5] validated the NLtoSTD-BB method and helped us make revisions. This paper presents an empirical study that evaluates the fault-detection ability of the revised NLtoSTD-BB method, and extends the research by evaluating the fault-detection ability of the STD-BBtoSTD method (used for the first time). Therefore, the complete NLtoSTD method (i.e., *NLtoSTD-BB + STD-BBtoSTD*) is evaluated in this paper.

## II. BACKGROUND

This section describes the basic concepts of the NLtoSTD method, the revised NLtoSTD-BB, and STD-BBtoSTD step.

### A. *NLtoSTD: Basic Concepts*

The basic idea of our NLtoSTD method is to translate the NL requirements into an STD, so that the ambiguity, incompleteness, and inconsistencies or any problem) in the NL requirements can be easily detected. The NL to STD translation first translates NL requirements into STD-BBs (*NLtoSTD-BB*) and then creates an STD using the STD-BBs (*STD-BBtoSTD*). A high-level overview of the NLtoSTD method is shown in Fig. 1. Fig.1 highlights the “*NLtoSTD-BB*” translation (Step1) and the “*STD-BBtoSTD*” construction (Step2). We hypothesize that the NLtoSTD helps discover the problems in the NL requirements by examining the individual STD-BBs and the resulting STD.

As shown in middle part of Fig. 1, the three elements that make up a STD-BB (i.e., *current state* (*Sc*), *next state* (*Sn*), and *transition* (*T*)) are precisely extracted from an individual requirement. The selection of these elements was based upon the characteristics of an ideal requirement and an inspection scheme that can help detect the problems that are otherwise left undetected using traditional inspection methods. As illustrated

This work was supported in part by NSF CAREER Award CCF-1149389 to North Dakota State University.

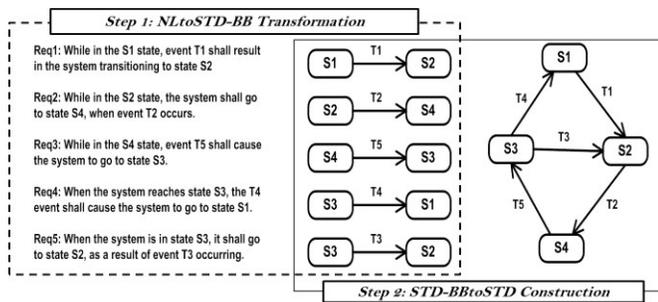


Fig. 1. NLtoSTD Method Overview

in Fig. 1, each requirement is stated so that it directly maps to an STD-BB. Each requirement explicitly states its precondition in the form of the current state ( $Sc$ ) and its post condition in the form of the next state ( $Sn$ ). However, typical NL requirements do not explicitly state current and next states, thus a requirement’s preconditions and post conditions are often inferred causing ambiguities and incompleteness. Similarly, the absence of the explicit transition ( $T$ ) can cause difference in the interpretations of a requirement by different stakeholders. The NLtoSTD method requires the stakeholders to identify the aforementioned three elements so that they can detect the requirement faults while building the STD and ensures that the requirements are as consistent and concise as possible.

### B. Step 1 - NLtoSTD-BB: Application for Fault Detection

The first step of the NLtoSTD method transforms each NL requirement into an individual STD-BB. The STD-BBs act as a formalized version of the NL requirements and can lead to the detection of faults for two reasons: (1) a formalized version of the NL requirements has only one specific interpretation, exposing *ambiguities* in the NL requirements, and, (2) a formalized version exposes *missing* requirements more readily, as compared to a fault-checklist inspection of requirements.

#### 1) Original NLtoSTD-BB [5]

As shown in Fig. 1, each NL requirement is translated into an STD-BB by extracting three elements  $\{Sc, T, Sn\}$ . The basis for this transformation is that a functional requirement should describe an entity transitioning from one state to another. For example, the requirement “While the car is moving forward, the driver shall be able to stop it by applying the brake.” would describe the Car (an entity) transitioning ( $T$ ) from moving ( $Sc$ ) to stopping ( $Sn$ ), using an STD-BB.

In the above example requirement, the three elements are explicitly stated, yielding definable values for  $Sc$ ,  $T$ , and  $Sn$ . In practice, however, requirements often ambiguously imply one or more values for  $Sc$ ,  $T$ , and  $Sn$ , thus identifying a value for each element would not be obvious. For instance, the prior requirement may have stated: “The driver shall stop the car by applying the brakes.” Note that  $Sc$  is not explicitly stated as “moving” but, rather, implied. In our original STD-BB, we used question marks (???) to denote an element that is not documented. Thus, in this example, we would define the three elements as  $\{Sc: ???, T: Applying Brake, Sn: Stop\}$ . It may be safe to assume that the car is moving prior to stopping, but it requires an assumption. Undocumented assumptions can be erroneous and can cause serious defects (especially when the developers lack appropriate knowledge of the application

domain). In this example, it is not clear whether we assume “moving forward,” “moving backward,” or both. It is important to document what may seem obvious, instead of allowing the possibility of an erroneous assumption. Therefore, the NLtoSTD-BB helps to expose undocumented assumptions.

We developed a set of three questions to help users systematically identify the three elements for each requirement during this step [5]. Asking these three questions identifies explicit or undocumented values for  $\{Sc, T, Sn\}$ , resulting in an STD-BB. While the ambiguities and incompleteness may not be obvious in the NL requirements, they are made obvious in an STD-BB that stakeholders can work towards its completion.

#### 2) Revised NLtoSTD-BB

The NLtoSTD-BB used during the Sudy 1 showed that the method was significantly more effective than the fault-checklist based inspection, when the subjects correctly extracted the STD-BBs. The variations in performance during study [5] prompted us to re-evaluate the way that the three elements ( $Sc, T, Sn$ ) were determined. This section dicuss the changes and the revised NLtoSTD-BB method.

Fig. 2 illustrates the revised NLtoSTD-BB method using an example requirement. In the revised NLtoSTD-BB method, the three changes are briefly discussed along with their reasoning.

The **first change** is that we explicitly added an entity to a state to represent  $Sc$  and  $Sn$  as follows: entity (state). Allowing for multiple entities would alleviate the problem encountered with requirements that are not atomic. Separating the concepts of an entity and its given states, also makes it easier to derive  $Sc$  and  $Sn$ , since the user could first decide which entity is being affected and then determine the entity states before and after the effect. Fig. 2 shows that the revised NLtoSTD-BB method, can help identify three entities: unit, battery, and user. The **second change** in the revised NLtoSTD-BB method is allowing users to make an assumption. As shown in Fig. 2, “unit (normalOp?)” has a question mark. This indicates that the user can assume that it is the intended state and label it for follow up. This was done to improve the method’s ability to expose ambiguities. The **third change** is to allow users to add conditions when they describe the transition ( $T$ ). This alleviates the problem when a requirement seems to state more than one transition. The revised NLtoSTD-BB method with five elements (*entity, entity’s current state, entity’s next state, transition, condition for transition*) is evaluated in this paper.

#### C. Step 2 - STD-BBtoSTD: Application and Tool Support:

The final step in the NLtoSTD method is the construction of STD based on the STD-BBs. The idea is to be able to both simulate the behaviours described in the requirements and to

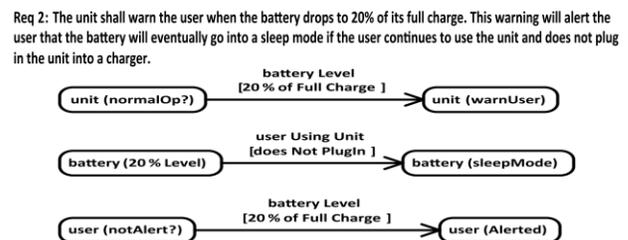


Fig.2. Revised NLtoSTD-BB

analyze these behaviours through the production of path traversals through the STD. This would potentially expose inconsistent and/or incorrect behaviours, that can be readily traced back to the requirements. An incomplete STD would expose incompleteness in the requirements verbage. Thus, the STD-BBtoSTD phase exposes potential faults by analyzing the STD’s static and dynamic properties.

The STD’s behavior was simulated by computer in order to expose faults that may not be evident unless the STD is enacted. There STD can be analyzed automatically by examining the path traversals, for desired behavior. The STD’s construction was implemented through a software tool. The user enters the STD-BB data in an Excel spreadsheet, which is then read directly into the tool (by the tool’s use of COM automation). The STD is then displayed in a separate window, and can be kept opened as more STD-BB data is being entered. The tool updates the STD, as changes are made to the STD-BB data. This allows the user to view the STD, make changes to the STD-BBs that would correct any STD structural faults, and see the results of those changes in real time. The subject can record faults during this step.

### III. EMPIRICAL STUDY

This study evaluated NLtoSTD-BB translation step at finding incompleteness and ambiguities during an inspection of NL requirements. This study also evaluated if additional new faults can be found during the construction of the STD. The complete NLtoSTD method was evaluated using a *repeated-measure design* in which different student teams (with varying number of members) developed requirement documents for different systems. Next, each participant individually inspects the requirement document (that was developed by them) using the NLtoSTD-BB step and kept a log of ambiguous, and missing requirements for respective documents. During the next step, the student individually worked to create STD from the STD-BBs (using an automated tool) and then analyzed the resulting STD to log new faults that were not found previously.

#### A. Research Questions and Hypotheses

The following research questions were investigated:

**RQ1:** Is the NLtoSTD-BB effective at detecting incomplete and ambiguous requirements during requirements inspection?

**RQ2:** Does creating the STD model result in detection of the faults in addition to those found during the NLtoSTD-BB?

**RQ3:** What are the problems faced by the subjects when using the complete NLtoSTD method?

#### B. Variables and Measures

Each subject performed an individual inspection of their requirements document using the NLtoSTD method. Our dependent variables include: *Effectiveness* - # of faults found and *Efficiency* - # of faults found per hour.

#### C. Participating Subjects and Requirement Artifacts

Sixteen computer science graduate students at North Dakota State University (NDSU) worked in teams to develop requirements document for different projects. Some students dropped the course resulting in this irregular size of student teams. There were two phases to this study. **First**, each team

TABLE I. ARTIFACTS DEVELOPED BY STUDENT TEAMS

Doc	Team #	No. of Subjects	System Description	Size (pages)
A	1	4	Parking lot availability system	25
B	2	4	Web portal for student residence	22
C	3	3	Virtual story board system	28
D	4	2	Matbus application for android	25
E	5	3	Professional development	32

developed a requirement document for a particular software system (Table I). **Second**, each subject inspected the requirement document developed by their team using the NLtoSTD-BB followed by STD-BBtoSTD method.

#### D. Study Procedure

The study details are provided in the following subsections.

1) *Phase I – Development of SRS documents:* The participants divided into 5 different teams of three or four participants developed the requirements documents for their identified software system. Details are provided in Table I.

2) *Phase II– Inspection using NLtoSTD:* During this step, the students in each team individually inspected their own SRS document using the NLtoSTD method.

a) *Training 1 -- NLtoSTD-BB:* The participants were first trained on how to map the NL requirements to STD-BBs. Next, the participants were instructed how to document the building block elements on a spreadsheet using few examples. Next, the participants were instructed how to record “ambiguities” and “incompleteness” or any other requirement faults that are found during the application of NLtoSTD-BB. The subjects were asked to translate few requirements into STD-BBs and document the faults using the same spreadsheet.

b) *Step 1-- Inspection using NLtoSTD-BB:* The participants used the information from Training 1 and individually inspected their own requirement document using the NLtoSTD-BB translation. This step resulted in a list of 16 individual spreadsheets that contained the STD-BB elements and the faults found (one per participant).

c) *Training 2 - tool support for STD Creation:* During this session, the participants learned about the STD tool. The subjects were instructed how to load the BBs (from step 1) into the tool and then, how to construct an STD from the BBs. The subjects were then instructed to examine the constructed STD. Finally, the participants learned to record the fault type in the fault spreadsheet. To ensure that subjects understood, the subjects practiced these steps through an example system.

d) *Step 2 – STD-BBtoSTD and inspecting STD:* The subjects constructed the STD diagram from STD-BB. The output of this step was 16 individual STD diagrams (one per participant). The resulting STD diagrams were analyzed for potential incompleteness, inconsistencies in the behaviours, or any other requirement faults. The participants analyzed and recorded the faults during and after the creation of STD. The students also documented the reason and classification of the fault (*incompleteness, ambiguous, inconsistency or other*) in the fault spreadsheet. This step resulted in 16 individual fault

lists. Finally, subjects provided feedback about the NLtoSTD-BB and the STD-BBtoSTD. An in-class discussion with subjects helped researchers better understand the results.

### E. Data Collection

The *quantitative* data included the *ambiguous*, *missing*, and *inconsistent* faults found by each subject in their SRS document during: a) translation of NL requirements to the STD-BBs, and b) construction of STD using the BBs. Each subject was provided 50 minutes during NLtoSTD-BB step and 30 minutes during STD-BBtoSTD step. The timing data was used for analyzing the efficiency values. The *qualitative* data included student's rating of NLtoSTD by answering a multi-question questionnaire based on a 5 point likert-scale. We also collected feedback post-study with participating subjects.

## IV. DATA ANALYSIS AND RESULTS

This section analyzed the data collected during NLtoSTD-BB, STD-BBtoSTD, post-study questionnaire and interviews.

### A. RQ 1: Effectiveness and Efficiency of NLtoSTD-BB

This section reports the *effectiveness* and *efficiency* of the NLtoSTD-BB during the requirements inspection. Before analyzing the fault data, the researchers determined the validity of the faults for each subject by reading through the fault spreadsheet reported by each participant to remove any false-positives (or if any faults were unclear). Next, the number of "Missing Functionality (MF)" and "Ambiguous Information (AI)" faults reported by each subject during the application of NLtoSTD-BB for their respective documents were counted.

Since each subject individually inspected their own document, the document (for each team) was inspected by all the subjects belonging to that team. Fig. 4 organizes results by the total number of AI and MF faults found by the member belonging to each team. Main observations from Fig. 4 follow:

- Fifteen out of sixteen subjects found faults (AI or MF) during the NLtoSTD-BB based inspection of their requirements document. The subjects (numbered 6) reported a lot of faults but none of them represented real problems;
- There were no consistent differences in the total number of AI vs. MF faults found by the subjects within each team. This was major improvement from the results in our previous study [5] where, the subjects using the NLtoSTD-BB method consistently found larger number of MF faults than the AI faults. This is a positive result that the improved heuristics were able to find both types of faults when constructing the STD-BBs.
- For each document, the average number of faults was calculated for each team by dividing the total number of unique faults by the number of subjects who inspected the document. The results showed that teams 1 through 5 found an average of 15, 12, 18, 9, and 22 faults respectively. This demonstrates an improvement in the performance of student teams from the first study [5] when using the original NLtoSTD-BB method (teams found an average of 7 faults at most) as well as a improvement when considering the inspection results in [5] when using the fault checklist method (an average of 5 faults).

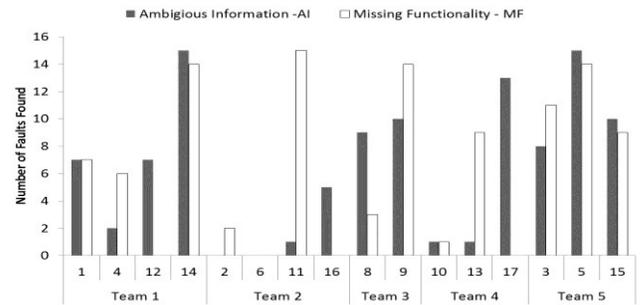


Fig. 4. Number of AI and MF Faults found by Subjects using NLtoSTD-BB

Therefore, based on these results, fault detection *effectiveness* of the NLtoSTD-BB method has improved from its first evaluation. Additionally, the distribution of faults is more consistent across both fault types (AI and MF). The revised NLtoSTD-BB heuristics were able to highlight hidden ambiguities in individual requirements which are otherwise not detected when performing a traditional inspection process.

Regarding the *efficiency* (faults/hour), the student teams found an average of 19, 7, 10, 10, and 26 faults per hour respectively. This is also an improvement in comparison to the results from the first study [5]. The high efficiency values reported in this study validate the ease of use of the revised NLtoSTD-BB method. Therefore, the NLtoSTD-BB is an effective and efficient method for verifying NL requirements.

### B. RQ 2: Fault Detection during the STD Creation

The translation of BBs into STD can highlight the ambiguities and incompleteness in the NL requirements by examining the gaps (or disconnections) and inconsistent path traversals in the STD, and to identify the *inconsistencies* in the requirements that are not a focus during the NLtoSTD-BB.

To investigate the validity of this step, we counted the number of new MF, AI and INC faults reported by each subject after the creation and analysis of STD for their respective documents. The result on the number of new AI, MF, and Inc faults found by each subject belonging to a team is shown in Fig. 5. Interestingly (Fig. 5), each subject found at least one new fault (AI or MF or Inc) after creating the STD. As expected, a larger number of "Inc" faults are found during this step as compared to the AI and MF faults. This is also consistent across all the teams. The subjects felt it was easy to observe the inconsistencies when looking at a complete STD as opposed to translating individual NL requirements (one at a time) to STD-BBs. Since, the loading of BBs to create the STD is an automated process (using a tool); it is not surprising that subjects were able to find additional faults by focusing their attention on examining the STD and recording faults.

To better understand the *effectiveness* of STD-BBtoSTD, we compared the percentage contribution of the STD-BBtoSTD relative to the overall NLtoSTD for each team. This was done by dividing the # of unique faults found during STD-BBtoSTD by sum of total # of unique faults found during NLtoSTD-BB and STD-BBtoSTD combined. The student teams 1-5, after the creation of STD, found 18%, 22%, 14%, 23%, and 12% of total faults respectively. To further verify the usefulness of the creation and analysis of STD, a one-sample t-

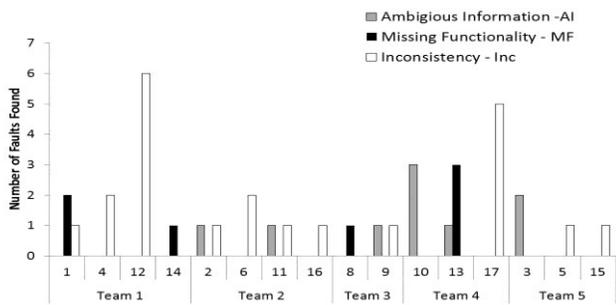


Fig. 5. Number of Faults found after the Creation of STD using the STD-BBs test was run separately for each team to determine whether the number of faults found during re-inspection using STD was significantly greater than zero (0). The result was found to be statistically significant ( $p < 0.001$ ) and indicate a benefit of creating STD using the BBs, for finding requirement faults.

### C. Difficulties Faced by Subjects Using the NLtoSTD Method

The qualitative data collected during this study evaluated the usability of the NLtoSTD-BB method. To do that, the participants were asked to rate the difficulty level for finding the “Entity”, “Initial Status”, “Changed Status”, “How is Status Changed”, and “Conditions for Change” for NL requirements.

Using a 5-point likert scale (1-very difficult to 5-very easy), the participants rated the difficulty level for each of the five elements of an STD-BB. A One-sample Wilcoxon Signed-Rank test determined whether the medians ratings were significantly greater than 3 (midpoint of the scale). The results showed that the NLtoSTD-BB method received positive ratings (i.e., median value greater than or equal to three), but not statistically significant. The subjects also rated the difficulty level during the construction of STD and analyzing the constructed STD for faults. The results showed that the STD creation received significantly positive ratings ( $p < 0.05$ ).

The complete NLtoSTD method was also evaluated using the feedback from subjects on the following seven attributes: *Simplicity*, *Ease of Understanding*, *Ease of Use*, *Intuitiveness*, *Comprehensiveness*, *Usefulness*, and *Ease of finding faults*. Each subject rated the attributes on a 5-point scale. The results from Wilcoxon Signed-Rank test revealed that the NLtoSTD received significantly positive ratings on *Ease of Understanding*, *Ease of Use*, and *Ease of finding faults*.

Overall, the subjects felt that the NLtoSTD process helped them understand the major problems in requirements, and that the effort spent during the NLtoSTD inspection process was worthwhile. The potential improvements regarding the tool and the guidance to help analyze the STD diagram will be implemented in future evaluations.

## V. DISCUSSION OF RESULTS

**RQ 1:** The NLtoSTD-BB method helped inspectors find inherent ambiguities and incompleteness in requirements. The comparison of the results against the previous research results [5] revealed that the subjects were able to find larger number of total faults (on average), and the distribution of faults across fault types (MF and AI) was more consistent. The results also showed that the NLtoSTD-BB method helped find the faults faster (i.e., efficiency) when compared to the results in [5].

**RQ 2:** Based on the results, additional MF and AI fault types are uncovered during the examination of STD constructed from the BBs. In particular, the creation of STD aids inspectors at detecting a large number of “Inc” that are otherwise not apparent when looking at individual requirements. The construction of STD is useful for overall inspection effectiveness using the complete NLtoSTD method.

**RQ 3:** The subjects provided insights in to the use of the NLtoSTD method and improvements that can help improve the performance in future studies. The subjects mentioned that the tool should guide the NLtoSTD-BBs translation and should at least highlight parts of STD that are completely disconnected. We plan to make this process as much automated as possible without losing the promise of inspections.

## VI. CONCLUSION AND FUTURE WORK

Based on these results, the NLtoSTD method is effective method to detect AI, MF, and Inconsistency fault types during an inspection of NL requirements. We also identified the areas of improvement that would benefit the performance of subjects using the method. Our future work would include more replications, with a classic control group design so that we can understand how many faults found during the Phase III are solely due to the STD creation and not just due to the re-inspection. We also wish to automate as much of the heuristics as possible, including the NLtoSTD building block portion. The STD analysis could be automated as well, using a reasoning engine written in a logic language such as Prolog, and this has already been achieved to a certain degree.

## REFERENCES

- [1] B. Boehm and V. Basili. Software fault reduction top 10 list. *IEEE Computer*, pages 135–137, January 2001.
- [2] B. Boehm. *Software Engineering Economics*, Prentice-Hall, 1981.
- [3] B. Brykczynski, A survey of software inspection checklists, *ACM SE Notes*, 24(1):82,1999.
- [4] C. Damas, B. Lambeau, P. Dupont, and A. Lamsweerde, Generating annotated behavior models from end-user scenarios,” *TSE*, 31(12):1056-1073, 2005.
- [5] D. Aceituna, H. Do, G. Walia, and S. Lee. Evaluating the use of model-based requirements verification method: A feasibility study. *EmpiRE*, 2011, pages 13-20, August 30, 2011.
- [6] D. Popescu, S. Rugaber, N. Medvidovic, and D. Berry, Reducing ambiguities in requirements specifications via automatically created object-oriented models, *Monterey Workshop on Computer Packaging*, pp. 103-124, 2007.
- [7] F. Chantree, B. Nuseibeh, A. de Roeck, and A. Willis. Identifying nocuous ambiguities in natural language requirements. *RE*, pp 59–68, 2006.
- [8] R. Chillarege, I. Bhandari, J. Chaar, M. Halliday, D. Moebus, B. Ray, and M. Wong, Orthogonal fault classification - A concept for in-process measurements. *TSE*, 18(11): 943-956, 1992.
- [9] S. Sakhivel. Survey of requirements verification techniques. *Journal of Information Technology*, pp. 68-79, 1991.
- [10] L. Kof, R. Gacitua, M. Rouncefield, and P. Sawyer, Ontology and model alignment as a means for requirements validation, *ICSC*, pp. 46-51, 2010.
- [11] D. Barry. Ambiguity in natural language requirements documents. *Lecture Notes in Computer Science, LNCS*, volume 5320, pages 1-7, 2008.
- [12] E. Letier, J. Kramer, J. Magee, and S. Uchitel, Monitoring and control in scenario-based requirements analysis. In *Proceedings of the 27<sup>th</sup> International Conference on Software Engineering*, pages 382–391, 2005.

# DBPD: A Dynamic Birthmark-based Software Plagiarism Detection Tool

Zhenzhou Tian, Qinghua Zheng, Ming Fan, Eryue Zhuang, Haijun Wang, Ting Liu\*

Ministry of Education Key Lab For Intelligent Networks and Network Security

Department of Computer Science and Technology, Xi'an Jiaotong University, Xi'an, 710049, China

{zztian, fanming.911025, hjwang.china}@stu.xjtu.edu.cn; {qzhzheng, tingliu}@mail.xjtu.edu.cn; zhuang8225@126.com

**Abstract:** *With the burst of open source software, software plagiarism has been a serious threat to the software industry. In this paper, we present the demo tool DBPD: **D**ynamic **B**irthmark-based **S**oftware **P**lagiarism **D**etection. Major features of DBPD could be summarized as: 1) dynamic birthmark. The execution process of software is captured to generate the birthmark reflecting intrinsic properties of software; 2) high availability. It is available for cross-platform and binary executable's plagiarism detection; 3) customizable. The birthmarks, similarity calculation metrics and detection criteria are configurable. The DBPD is implemented using C++ and Java, and currently can work under both Windows and Linux system. Three dynamic birthmarks are implemented in DBPD to identify the software according to its instruction, stack operation and system call.*

**Keywords:** software plagiarism detection, dynamic birthmark

## I. INTRODUCTION

Free and open source software projects allow users to use, change and distribute software under certain types of license such as the well-known GPL. However, driven by the huge commercial interests, some individuals and companies incorporate third party software or libraries into their own products without respecting the licensing terms. Recent incidents include the lawsuit against Verizon by Free Software Foundation for distributing Busybox in its FIOS wireless routers [1], and the crisis of Skype's VOIP service for the violation of licensing terms of Joltid. The unavailability of source code and the existence of powerful automated semantic-preserving code transformation tools, make the plagiarism an easy to implement but difficult to detect thing.

Software birthmark, a set of characteristics extracted from a program that reflect the program's intrinsic properties and that can be used to uniquely identify the program, is a promising way for solving the plagiarism detection problem. However, despite the tremendous progress of birthmark based plagiarism detection approaches, seldom tools are publically available. The rare few tools as far as we find are SandMark [2], Stigmata [3] and Birthmarking [4]. The former two are static birthmark based which are believed to be fragile faced with semantic-preserving code obfuscation techniques, and the last one is dynamic birthmark based which is believed to have better performance than the previous two static birthmarks, yet they all suffer the problem of language dependence, since they're

\*Ting Liu is the corresponding author. The research was supported in part by National Science Foundation of China (91118005, 91218301, 61221063, 61203174), 863 Program (2012AA011003), The Ministry of Education Innovation Research Team (IRT13035), Key Projects in the National Science and Technology Pillar Program of China (2012BAH16F02).

only valid for java programs. Also, there are some mature tools such as the JPlag [5] that target at source code which is not always available, since plagiarists are not likely to provide their source code before certain evidences are collected. Thus more powerful and practical tools are in urgent needs to fill the gap of birthmark based plagiarism detection research and practice.

It is a generally accepted fact that dynamic birthmarks being abstractions of runtime behaviors are believed to be more accurate reflections of program semantics than static birthmarks. Therefore, we implement a demo tool DBPD for plagiarism detection using dynamic birthmark techniques. Three dynamic birthmarks are implemented in DBPD to identify the software, including DKISB (dynamic key instruction sequence birthmark) [6], SODB (call stack operation dynamic birthmark) [7] and SCSSB (system call short sequence birthmark) [8]. Since all of them can work directly on binary executables, DBPD can analyze various programming languages.

## II. TOOL OVERVIEW AND IMPLEMENTATION

### A. Tool Overview

Fig.1 shows the overview of the DBPD. It consists of three main modules: the dynamic analysis module, the birthmark generator, the similarity calculator and decision maker. The modular architecture qualifies it with good scalability of easily introducing new kinds of dynamic birthmark methods.

Given two binary executables the plaintiff (original program), the defendant (suspicious program) and a set of inputs, DBPD executes both programs with the same input one by one. Meantime, the dynamic analysis module monitors the executions, performs dynamic analysis and collects execution profiles containing three kinds of events: key instructions, stack operations, and system calls. After sequences of both plaintiff and defendant programs are available, they are fed into the birthmark generator where noises are filtered, valid

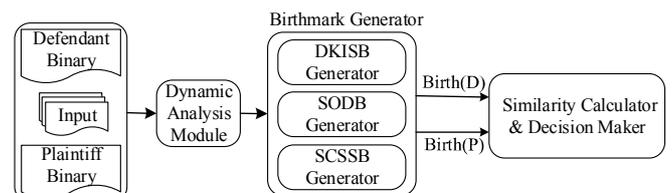


Fig. 1 Design overview of DBPD



# A MAPE Loop Control Pattern for Heterogeneous Client/Server Online Games

Satoru Yamagata<sup>1</sup>, Hiroyuki Nakagawa<sup>2</sup>, Yuichi Sei<sup>1</sup>, Yasuyuki Tahara<sup>1</sup>, Akihiko Ohsuga<sup>1</sup>

<sup>1</sup>Graduate School of Information Systems, The University of Electro-Communications  
Tokyo, Japan

{s-yamagata, sei, tahara, ohsuga} @is.uec.ac.jp

<sup>2</sup>Graduate School of Information Science and Technology, Osaka University  
Osaka, Japan

nakagawa@ist.osaka-u.ac.jp

**Abstract**—Online games are recently gained attention according to the increase of the bandwidth of the Internet. On the other hand, when we develop an online game, we have to consider the time lag problem caused by the heterogeneous environments. In this poster paper, we try to apply MAPE loop model, which consists of four key activities (Monitoring, Analysis, Planning, and Execution) for adaptation, to help reduce the problem in the heterogeneous online game environment. In particular, we propose a MAPE loop control pattern for the heterogeneous client/server model. We experimentally embed the pattern into an online game application, and the results demonstrate that our MAPE loop control pattern will help to reduce the time lag problem in the online game application.

**Keywords**- client/server online game; self-adaptive systems; MAPE loop control pattern; time lag

## I. INTRODUCTION

The number of online games and their users are rapidly increasing. They provide rich and up-to-date contents; however, they require real-time processing and fast interactivity. The time lag problem by network communication delay is one of the most serious problems. The quality of an online game goes down under the situation where the time lag occurs. In particular, match-type fighting game and FPS (First Person shooter) game require fast reaction speed and operation accuracy of game player in real-time. This paper focuses on inversion of an order according to the difference of the time lag of individual client. Moreover, online games need high permanency. Consequently, online game systems require that server applications receive much attention because of their ability to cope with the changes of environment, failures, and unanticipated event. The systems with such capability are called self-adaptive systems [1][3]. Self-adaptive systems should have certain decision mechanisms and using control loops is one of the approaches. The MAPE loop is one of control loops, which consists of four key activities (Monitoring, Analysis, Planning, and Execution) for adaptation [2][5][6]. Self-adaptive systems include these components as subsystems. However, practical systems including these components still have many problems, such as those design methods, domain characteristics and few actual adaptation examples [2].

In this paper, we propose a novel MAPE loop control pattern that can reduce the time lag influence. This pattern can be applied to the client/server type systems.

## II. PROPOSED PATTERN

We propose a MAPE loop control pattern for client/server type systems, such as client/server online games (Figure 1). In this pattern, we deploy a Monitor and analyze component collects the performance data of the client, and the Analyze component analyzes the current situation the client. We also deploy a Plan and Execute components in the server, so that the server can determine and change to a suitable behavior at runtime.

### A. The outline of a proposal pattern

The bottom of Figure 1 illustrates an instance of our MAPE loop pattern with four clients. We can apply this pattern to client/server applications that requires high accuracy in real-time but are deployed in heterogeneous environment. Heterogeneous environment means a distributed environment where the performance of hardware and software, or network bandwidth is different. In such environment, the next situation of whole system is hard to predict. Therefore, it is not desirable to deploy Analyze component to server. This pattern is for the applications that perform complicated processing and the

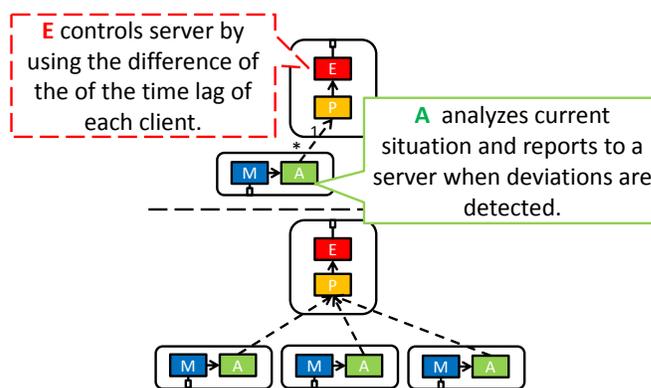


Figure 1. Upper: Proposed control pattern. Bottom: Instance

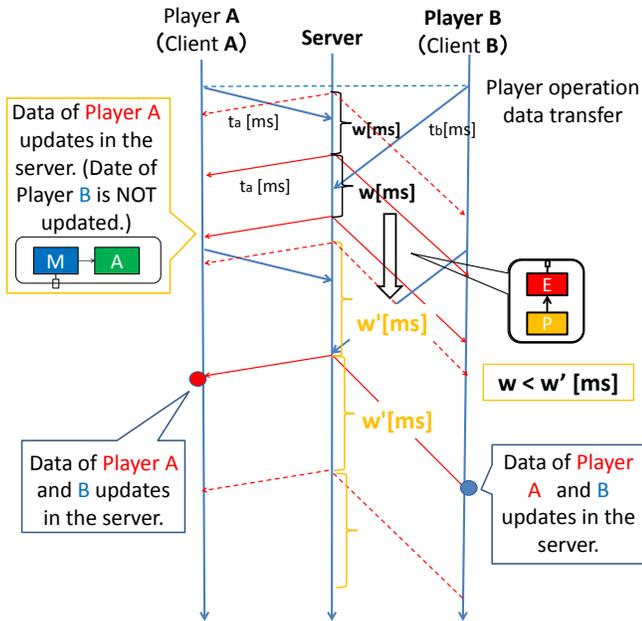


Figure 2. The online game system by a proposal pattern

server manages individual clients. Therefore, it is desirable that the server grasps the analysis result of each client so that, the server grasps the analysis result of each client so that, the server changes the system configuration and behavior. From these points, our pattern deploys Monitor and Analyze components in individual client nodes, and deploys Plan and Execute components in the server node. As a result, the Monitor components can acquire individual clients' data and the Analyze components can analyze the current situation of individual clients, which is difficult to recognize at the server side at runtime.

### B. The self-adapting scenario to a time lag of an online game

This pattern is for the applications that perform complicated processing and the server manages individual clients. When the server can update all clients' data all at once and the clients can reflect the updated result in their side, the game keeps valid state. In this scenario, the server appropriately updates the updating cycle in consideration of the communication delay of individual client. When the delays of the client A and the client B are largely different, the server changes the updating cycle from  $w$  to  $w'$  ( $w < w'$ ). This change enables the server to receive more clients' data in the same cycle (Figure 2). Monitor component in individual clients collects individual network delays. Based on the collected data, the Analyze components analyze whether inconsistency. Based on the analysis result, Plan component in the server determines whether the updating cycle should be changed. Execute component changes the updating cycle according to the determination.

### III. EVALUATION

We experimentally implemented a virtual online game environment in Java and applied our MAPE loop pattern on application. In order to easily change delays of individual

clients, we implemented the environment in a desktop PC. The delays are realized by using sleep method. In this environment, each client sends the selected command to server. The server updates to the current status based on the sent data from clients and send update status to the clients at the end of the updating cycle. In this experiments, we count inconsistencies that are occurred when there exists unreachable client data, which caused by the large difference of time lag among clients. We prepared two (experiment 1) and three (experiment 2) clients. In each experiment, we compared two applications; one was implemented without our MAPE loop pattern and the other included the pattern. Table 1 shows the experimental results. The results demonstrate that our MAPE loop pattern is effective to reduce the inconsistencies by suitably control updating cycles.

TABLE I. EXPERIMENTAL RESULT

	without self-adaptation	self-adaptation
experiment 1	46	2
experiment 2	18	3

### IV. CONCLUSION AND FUTURE WORK

We proposed a MAPE loop control pattern for the client-server applications in heterogeneous environments. In particular, we focused on online game applications and their time lag problems. We have experimentally evaluated the effectiveness of our MAPE loop pattern through the experiments in a virtual online game environment. In future work, we try to improve the accuracy of the updating cycle by taking into account the prediction of environment by introducing a machine learning mechanism. We also plan to introduce a programming framework that helps developers to embed the MAPE loop pattern into existing online game applications.

### REFERENCES

- [1] Hallsteinsen, S., Stav, E., Solberg, A. and Floch, J.: Using Product Line Techniques to Build Adaptive Systems, in Proc. 10th (SPLC), 2006
- [2] Kephart, J.O., Chess, D.M.: The vision of autonomic computing, Computer 36(1), pp. 41-50, 2003.
- [3] Morin, B., Barais, O., Jezequel, J.-M., Fleurey, F. and Solberg, A.: models@run.time to support dynamic adaption, , IEEE Computer, Vol.41, No,10(2008), pp.44-51.
- [4] Robinson, W. and Puroo, S. : Specifying and Monitoring Interaction and Commitments in Open Business Processes, IEEE Software, Vol.26, No.2(2009), pp.72-79.
- [5] Weyns, D., Schmer, B., Grassi, V., Malek, S., Mirandola, R., Prehofer, C., et al.; On Patterns for Decentralized Control in Self-Adaptive Systems, Software Engineering for Self-Adaptive Systems II Lecture Notes in Computer Science Volume7475, pp. 76-107, 2013.
- [6] Weyns, D., Iffikhar, U. and Soderlund, A.: Do External Feedback Loops Improve the Design of Self-Adaptive Systems? A Controlled Experiment, In:Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS2013), 2013

# DiCoMEF: A Distributed Collaborative Model Editing Framework

Amanuel Koshima  
University of Namur  
PReCISE Research Center, Belgium  
amanuel.koshima@unamur.be

Vincent Englebert  
University of Namur  
PReCISE Research Center, Belgium  
vincent.englebert@unamur.be

## Abstract

*Domain specific modeling languages have matured over-time and are widely used, but facilities for collaborative modeling are still limited. Hence, there is a demand for tool supports that facilitates sharing of modeling artifacts (collaboration), conflict detection, reconciliation, and merging of concurrently edited models. This paper presents a collaborative model editing framework called DiCoMEF.*

## 1 Introduction

Domain Specific Modeling Languages (DSML) have matured overtime and widely used as an efficient alternative to General Purpose Modeling languages (e.g., UML, Petri Nets) for modeling complex systems [2]. DSML describes concepts at different levels of abstraction using models, meta-models and meta-meta-models. A model is an abstraction of a software system and a meta-model is a language that describes concepts and constraints of models. A meta-meta-model is a minimum set of concepts which defines languages (e.g. EMF/Ecore<sup>1</sup>).

Several metaCASE tools have been developed to support the use of models defined with DSML, but most of these tools do not support collaboration [3]. Nevertheless, whenever the complexity of a problem increases, the diversity of users in groups increases as well [4]. Hence, methods and tools to facilitate the cooperative work of these users are required. A model version control system, which facilitates sharing of modeling artifacts, conflict detection, merging, and versioning of models, is then required. More particularly, conflicts (textual, syntactic, or semantic [1]) that cause inconsistency should be identified and resolved.

Different approaches have been adopted to ensure collaboration, for instance, a central repository (e.g. SVN) with locking techniques and merging mechanism. However, the locking technique is not scalable [1] and this approach forces all users to be dependent on a central repository. This might introduce unnecessary access right bureaucracy that

would lead to dissatisfaction among members. There is another mode of collaborative work where each member has his/her local copy of a shared modeling artifact and works in isolation with other members. This approach provides users a better control of modeling artifacts and mitigates the problem of being dependent on a central server. However, it is challenging to keep all copies of concurrently edited models consistent. Most of available versioning tools detect conflicts in text or tree based documents, but they are not suitable for models that have a graph based nature [1].

## 2 DiCoMEF Framework

DiCoMEF is a distributed collaborative model editing framework based on EMF/Ecore meta-model and it lets each member of the group having his/her local copy of (meta)model (see Figure 1) [3]. In DiCoMEF, modifications are controlled by human agents (controllers). They manage the evolution of either models or meta-models depending of their role in the cooperative work. Model (resp. meta-model) controller roles are flexible meaning that they can be assigned (delegated) to other members of a group as long as there is one unique coordinator per group. Whenever members of a group modify (meta-)models locally, elementary change operations (*create*, *delete*, and *update*) are stored locally. DiCoMEF extends a history meta-model of EDAPT<sup>2</sup> to capture edit operations. These operations are used later as a means of communication among members of a group. A controller is used as a central hub of communication among a group, but members could still communicate with other colleagues directly. Peer-to-peer communication could hinder convergences of all copies of (meta)models.

DiCoMEF relies on two concepts (*main-line* and *branches*) to ensure the communication framework. The main-line stores different versions of a copy (meta-)model locally at each editors site. Editors cannot modify (meta-)models stored on the main-line; they can only adapt those stored on the branch and send then their local modifications

<sup>1</sup><http://www.eclipse.org/modeling/emf/>

<sup>2</sup><http://www.eclipse.org/edapt/>

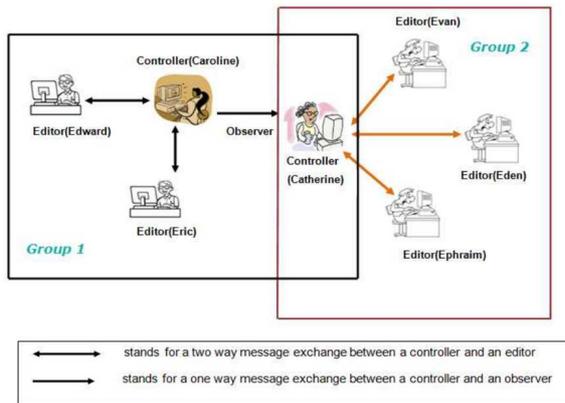


Figure 1. DiCoMEF architecture

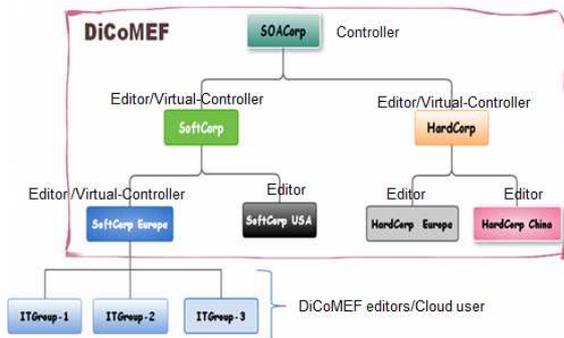


Figure 2. Extended architecture of DiCoMEF

to a controller as “change requests” so as to commit changes on the main-line. DiCoMEF detects structural conflicts (using edit operations) (see Figure 3) and semantic conflicts (with OCL constraints). Edit operations can be annotated with multimedia files to describe rationale of modifications. DiCoMEF framework could be extended to support a large community of users as shown in Fig. 2, where an editor acts as a *virtual controller* for other editors (*side editors*) working under her/his supervision. These new roles (Virtual controller and side editor) are transparent for the DiCoMEF controller. Side editors could also modify (meta-)models concurrently (e.g. by using the *Cloud*) but these modifications would be out of the scope of DiCoMEF.

### 3 Status and Evaluation

DiCoMEF framework is an eclipse plugin (54K LOC) based on EMF/Ecore meta-model and it provides a fully operational support for collaborative meta-modeling. The DiCoMED plugin, along with screencasts and tutorials are available online on the DiCoMEF Web site<sup>3</sup>. We have conducted a preliminary evaluation of DiCoMEF framework

<sup>3</sup><https://sites.google.com/site/dicomef/home>

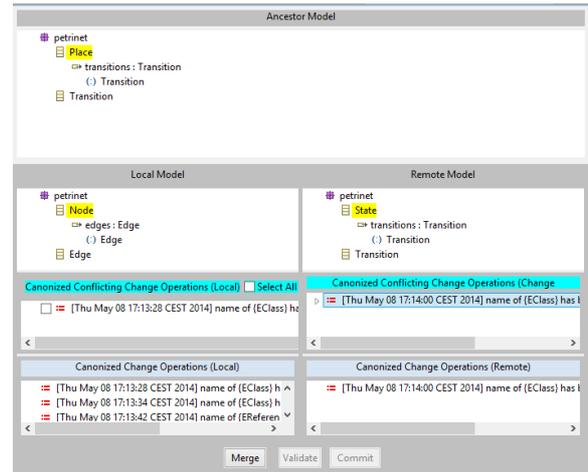


Figure 3. Merge tool

with master students to assess: (1) the feasibility of collaborative methods and processes with DiCoMEF, (2) the correctness of conflict detection mechanisms (recall and precision), (3) the usability of the merge tool and DiCoMEF framework (4) measuring user efforts (time) needed to merge concurrently edit meta-models manually and using DiCoMEF merge tool. Overall result of the evaluation was positive and it showed that DiCoMEF can be used for collaborative meta-modeling.

### 4 Future Work

The reconciliation process of DiCoMEF will be improved by letting users work with concrete syntax editors and meta-model semantics (i.e. OCL rules for instance). Besides, more advanced collaborative workflows will also be investigated and defined on top of DiCoMEF. In addition, we plan to conduct more experiments and evaluations in the future.

### References

- [1] K. Altmanninger, M. Seidl, and M. Wimmer. A Survey on Model Versioning Approaches. Technical report, Johannes Kepler University Linz, 2009.
- [2] S. Kelly. Case tool support for co-operative work in information system design. In *Info. Sys. in the WWW Environment*. Chapman & Hall, 1998.
- [3] A. Koshima and V. Englebert. Collaborative editing of emf/ecore metamodels and models: Conflict detection, reconciliation, and merging in dicomef. In *Proc. of the MODEL-SWARD 2014*, Lisbon, Portugal, 2014.
- [4] T. Zimmermann and C. Bird. Collaborative software development in ten years: Diversity, tools, and remix culture. In *Proc. of the CSCW Workshop on the Future of Collaborative Software Development*, 2012.

# Model-based time-distorted Contexts for efficient temporal Reasoning

Thomas Hartmann\*, Francois Fouquet\*, Gregory Nain\*, Brice Morin<sup>‡</sup>, Jacques Klein\* and Yves Le Traon\*

\*Interdisciplinary Centre for Security, Reliability and Trust (SnT), University of Luxembourg, Luxembourg, first.last@uni.lu

<sup>‡</sup>SINTEF ICT Norway, Norway, first.last@sintef.no

## I. INTRODUCTION

Intelligent systems continuously analyze their context to autonomously take actions. Building a proper knowledge representation of the context is key to take adequate actions. This requires context models, *e.g.* formalized as ontologies or meta-models. As these systems evolve in dynamic contexts, reasoning processes typically need to analyze and compare the current context with its history. A common approach consists in a temporal discretization, which regularly samples the context at specific timestamps (snapshots) to keep track of history. Fig. 1 shows a context sampled at three different timestamps. Reasoning processes would then need to mine a huge amount of data, extract a relevant view, and finally analyze it. This would require lot of computational power and be time-consuming, conflicting with the near real-time response time requirements of intelligent systems. To address these issues, we define *time-distorted* contexts as time-aware context models. Fig. 2 shows a context representation, where the context variables belong to different timestamps. Our approach considers temporal information as first-class property crosscutting any context element, and enables building time-distorted views of a context composed by elements from different times rather than a mere stack of snapshots. We claim that these time-distorted views can efficiently empower continuous reasoning processes and outperform traditional full sampling approaches by far.

## II. BACKGROUND

Over time different formalisms to represent the context of intelligent systems have been developed [1], [2], [3] for different purposes. Entity-relationship models [4], as a general modeling concept for describing entities and the relationships between them, are widely used for building context representations. Most of these approaches describe a context using a set of concepts (classes, types, elements), attributes (properties), and the relations between them. We refer to the representation of a context (set of elements) as a *context model* and to a single concept as *model element*. An emerging paradigm called *models@run.time* [5], [6], [7] proposes to use models both at design and runtime in order to support intelligent systems. Models support the design and implementation of the system, and are then embedded at runtime to support the reasoning processes of intelligent systems, as models offer a *simpler, safer and cheaper* [8] means to reason. Our implementation

and the provided API are build on a models@run.time-based context representation approach and are integrated into an open source modeling framework, called Kevoree Modeling Framework [9] (KMF<sup>1</sup>). KMF is an EMF [10] alternative specifically designed to support the models@run.time paradigm.

## III. TIME-DISTORTED CONTEXTS

We consider temporal knowledge as part of a domain itself (*e.g.* electric load or wave propagation prediction, medical recommender systems, financial applications) and think that defining and navigating temporal data directly within domain contexts is far more efficient and convenient than regularly sampling a context and independently querying each model element with the appropriate time. Here, we define concepts to navigate into the time dimension of contexts and seamlessly combine elements from different points of time.

**Temporal validity:** Instead of relying on context snapshots, we define a context as a continuous structure, where each element can evolve independently. We define an implicit *validity* for model elements by associating a timestamp to each of them that defines a *version*  $v_{m_e}(t)$  of a model element  $m_e$  at a time  $t$ . If a model element evolves, an additional version of the same element is created and associated to a new timestamp. Timestamps can be compared and thus form a chronological sequence. Although timestamps are discrete values, they logically define intervals in which a model element can be considered as *valid* (*i.e.* from the time it is captured until a new version is captured). New versions are only created if the model element changes. Because of this temporal validity, a relationship  $r$  from a model element  $m_{e1}$  to  $m_{e2}$  is no longer uniquely defined. Instead, the timestamps of model elements have to be taken into account for resolving relationships.

**Navigating in time:** Based on the assumption that intelligent systems need to consider not only the current context but also historical data, we provide means to enable an efficient navigation into time. We define three operations that can be called on each model element. The *shift* operator takes a timestamp as parameter, looks for the valid version of the element at the required timestamp, loads it from storage and returns the loaded version. *previous* and *next* are shortcuts to respectively retrieve the direct predecessor and successor of the current model element. These operations allow to shift model elements independently from each other through time, and make it possible to create context models combining model elements from different points of time.

The research leading to this publication is supported by the National Research Fund Luxembourg (grant 6816126) and Creos Luxembourg S.A. under the SnT-Creos partnership program.

<sup>1</sup><http://kevoree.org/kmf>

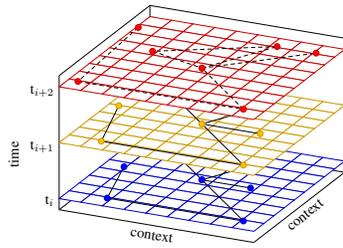


Fig. 1. Linear sampled context

**Time-relative navigation:** Navigating temporal data is complex, since a relationship  $r$  from an element  $m_{e1}$  to an element  $m_{e2}$  is not uniquely defined. Indeed,  $r$  can link different versions of  $m_{e2}$  depending on the timestamps  $t_1$  and  $t_2$  of  $m_{e1}$  and  $m_{e2}$ , and on the set of versions of  $m_{e1}$  and  $m_{e2}$ . This means that the version of  $m_{e2}$  linked to  $m_{e1}$  by  $r$  depends on the timestamp  $t$  of  $m_{e1}$ . Navigating in the model, while considering this time-relative navigation manually is complicated and error-prone. We therefore provide a navigation mechanism, hidden in the navigation methods of the context model, that automatically resolves the relationships transparently for the user. Hereby, a context time can be defined (the curve in fig. 2) and each model element is then resolved accordingly to this definition while traversing the model. For example, the context time can be defined as the current time of a model element minus one day. When navigating from model element  $m_{e1}$  at timestamp  $t_i$  to element  $m_{e2}$ , the version of  $m_{e2}$ , which is valid at timestamp  $t_i - 1 \text{ day}$  is resolved. In case at timestamp  $t_i$  object  $m_{e2}$  does not exist the *prior existing version* of  $m_{e2}$  is returned. Considering model elements in the context of a specific time interval creates a navigable time dimension for model elements. This time relative data resolution is one of the novel concepts of this contribution. Indeed unlike in previous approaches (e.g. relationships in MOF [11] or predicates in RDF [12]), the navigation function is not constant but yields different results depending on the navigation context (i.e. the current observation date). This distortion in terms of navigable relations finally enables what we call a time-distorted context.

#### IV. IMPLEMENTATION AND EVALUATION

This approach has been integrated in an open source modeling framework: KMF. The integration relies on two properties: i) each model element must be uniquely identifiable, and ii) it must be possible to get a serialized representation of all attributes and relationships of each model element, with no relativity to a time. KMF offers a *path* mechanism to support the i) property, and *traces* to address the ii) requirement. Thanks to this integration, we ran an experimentation on a smart grid example to evaluate our approach by comparison with a full sampling strategy. To this end, we implement a reasoning engine, which aim is to predict if the electric load in a certain region will likely exceed or surpass a critical value. Our validation focuses on two key indicators: (1) performance of the reasoning process and (2) insertion time. In order to cover several use cases, our experimentation involves 4 different set-ups varying a) the size of the area used in the prediction (number of meters) and b) the size of the history. Our experimentation shows that the insertion time, compared to full sampling, is improved by a factor of 17 and that the

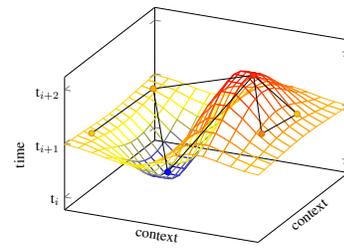


Fig. 2. Time-distorted context

time required by the algorithms to finish have been improved by factors from 598 to 1361.

#### V. CONCLUSION

Considering time as a crosscutting concern of data modeling has been discussed since more than two decades. However, recent data modeling approaches mostly still rely on a discrete time representation, which can hardly consider model elements coming from different points of time. We presented an approach, which considers time as a first-class property crosscutting any model element, allowing to organize context representations as time-distorted views dedicated for reasoning processes, rather than a mere stack of snapshots. By introducing a temporal validity for model elements we allow them to evolve independently and at different paces, making the full sampling of a context model unnecessary. Instead of introducing a dedicated querying language we provided operations to move model elements independently through time. Finally, we added a time-relative navigation, which makes an efficient navigation between model elements, coming from different timestamps, possible. Our approach has been implemented and integrated into KMF.

#### REFERENCES

- [1] M. Perttunen, J. Riekkilä, and O. Lassila, "Context representation and reasoning in pervasive computing: a review," *Int. Journal of Multimedia and Ubiquitous Engineering*, pp. 1–28, 2009.
- [2] M. Baldauf, S. Dustdar, and F. Rosenberg, "A survey on context-aware systems," *Int. J. Ad Hoc Ubiquitous Comput.*, vol. 2, no. 4, 2007.
- [3] T. Strang and C. L. Popien, "A context modeling survey," in *UbiComp 1st Int. Workshop on Advanced Context Modelling, Reasoning and Management*, 2004, pp. 31–41.
- [4] P. P. shan Chen, "The entity-relationship model: Toward a unified view of data," *ACM Trans. Database Syst.*, vol. 1, pp. 9–36, 1976.
- [5] G. Blair, N. Bencomo, and R. France, "Models@ run.time," *Computer*, vol. 42, no. 10, pp. 22–27, 2009.
- [6] B. Morin, O. Barais, J. Jezequel, F. Fleurey, and A. Solberg, "Models@ run.time to support dynamic adaptation," *Computer*, vol. 42, 2009.
- [7] S. Kent, "Model driven engineering," in *IFM*, 2002.
- [8] J. Rothenberg, L. E. Widman, K. A. Loparo, and N. R. Nielsen, "The nature of modeling," in *Artificial Intelligence, Simulation and Modeling*, 1989, pp. 75–92.
- [9] F. Fouquet, G. Nain, B. Morin, E. Daubert, O. Barais, N. Plouzeau, and J. Jézéquel, "An eclipse modelling framework alternative to meet the models@runtime requirements," in *MoDELS*, 2012.
- [10] F. Budinsky, D. Steinberg, and R. Ellersick, *Eclipse Modeling Framework : A Developer's Guide*, 2003.
- [11] OMG, *OMG Meta Object Facility (MOF) Core Specification, Version 2.4.1*, Object Management Group Std., Rev. 2.4.1, 2011.
- [12] O. Lassila and R. R. Swick, "Resource Description Framework (RDF) Model and Syntax Specification," W3C, W3C Recommendation, 1999.

# Agent-based Stochastic Simulation of Schema Matching

Hicham Assoudi

LATECE, Département d'Informatique  
 Université du Québec à Montréal  
 Montréal, Canada  
 assoudi.hicham@courrier.uqam.ca

Hakim Lounis

LATECE, Département d'Informatique  
 Université du Québec à Montréal  
 Montréal, Canada  
 lounis.hakim@uqam.ca

**Abstract**—In this demo, we present the implementation of a novel Agent-based Modelling and Simulation approach for the Schema Matching problem called “Schema Matching Agent-based Simulation” (SMAS). Our solution aims at generating high quality schema matchings with minimum uncertainty. As far as we know, there is no previous literature describing a solution approaching the Automatic Schema Matching and Mapping problem under the angle of Agent-Based Modelling and Simulation.

**Keywords**-schema matching; multi-agent systems; agent-based modelling and simulation

## I. INTRODUCTION

Schema matching is to find a pairing of attributes (or groups of attributes) from the source schema and attributes of the target schema such that pairs are likely to be semantically related. In many systems finding such a schema matching is an early step in building a schema mapping. Although these tools comprise a significant step towards fulfilling the vision of automated schema matching, it has become obvious that the user must accept a degree of imperfection in this process [1].

Agent based modelling and simulation (ABMS) is a new modelling approach that has gained increasing attention over the last decade. In ABMS, active components or decision makers are conceptualized as agents, being modeled and implemented using agent-related concepts and technologies [2].

In a this demo, we propose a novel Agent-based Modelling and Simulation approach for the Schema Matching problem called “Schema Matching Agent-based Simulation” (SMAS). We believe that our solution can be an effective approach to reduce to uncertainty inherent to the process of automatic schema matching.

## II. SCHEMA MATCHING AGENT-BASED SIMULATION (SMAS)

In a nutshell our idea is to model the Schema Matching process as interactions, within a self-organized environment, between agents called “Schema Attribute Agent”. In the rest of the paper we are going to refer to the “Schema Attribute Agent” simply as agent. Each schema attribute is modelled as an agent belonging to one of two groups: source or target

schema group. Furthermore, the schema matching process is modelled as the interaction between the two group's agents.

The figure bellow illustrates how the schema source and target attributes could be represented as agents in a simulation environment.

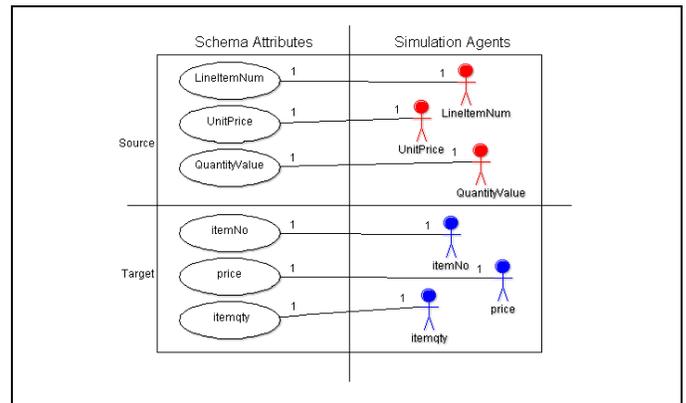


Figure 1. Representation of Schema Attributes as Simulation Agents

In our model the agents have as a main goal to find the best matching agent within the other group of agents. They execute behaviors based on rules with some randomness (stochasticity). The main randomness elements influencing the simulation are as follows:

1. Similarity calculation based on a similarity measures selected randomly from a similarity measures list.
2. Similarity scores aggregation based on aggregation functions selected randomly from an aggregation function list (MAX, AVERAGE, WEIGHTED).
3. Similarity score validation based on generated random threshold value (within interval)

The simulation ends when each agent has reached a consensus, about its candidate matching, with another agent (both agents are referring to each-other as candidate match).

As opposed to deterministic solutions, for schema matching, the nondeterministic and stochastic nature of our agent-based simulation increases the confidence in the quality

of the matching results. Despite the fact, that the agent's behaviors are based on randomness (e.g. during the similarity calculation), our model can often produce the right matchings at the end of each simulation run.

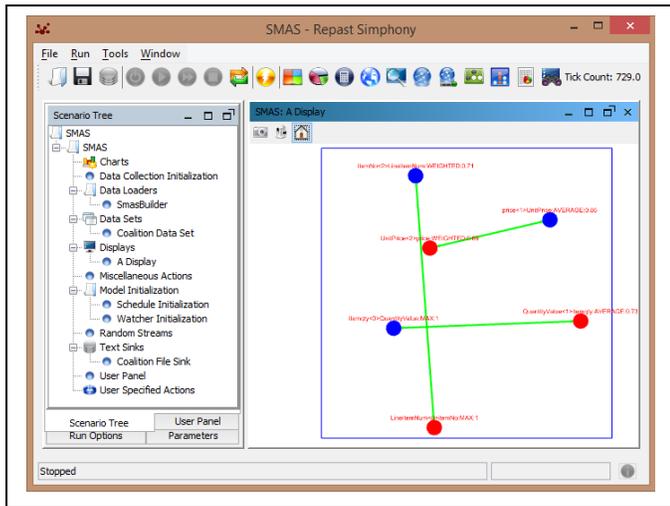


Figure 2. Screenshot of SMAS prototype

The foundation of the rules governing the agents behaviors, in our model, is stochasticity (randomness). In fact, a certain degree of randomness is present in each step executed by each agent during the simulation. Below are the steps executed during each tick of the simulation run:

- Step1 & Step2 - Calculation of the name and comment similarity (similarity measure is selected randomly from a similarity measures list and the score is compared to a random, within interval, threshold value)
- Step3 - Compare the best matches: If the best match for name and the best match for comment are converging to the same Agent (in the other group) then the scores are aggregated and the agent's status is changed from "NO\_MATCH" to "CANDIDATE\_MATCH" (the aggregation function is selected randomly from an aggregation function list: MAX, AVERAGE, WEIGHTED).
- Step4 - Check for consensus:
  - Consensus watching: check if a consensus was reached with another agent (both agents are referring to each-other as candidate match). If so, then the agent's status is changed to "CONSENSUAL\_MATCH"
  - Consensus timeout: If after a certain number of ticks (random value within interval) the agent has not yet reached a consensus with another agent then the agent beliefs about the candidate match are reset to null and the status is changed to "NO\_MATCH"
- Step 5 - Ending the simulation: If all agents have reached consensus about their matchings then their

status is changed to "STABLE\_MATCH" and the simulation is ended.

We developed a prototype for SAMS model (based on Repast Symphony (2.1) [3] and DKPro Similarity (2.1.0) [4]) and performed preliminary tests for which the results<sup>1</sup> obtained were compared to the expected matchings (provided by the human expert). For the sample schemas, we performed the experiments with, SMAS was able to successfully resolve all the expected matchings. We also performed statistical analysis on multiple simulation runs to develop confidence that what we are seeing is not a fluke of one particular run but instead is some regularity coming repeatedly and robustly out of our model.

### III. DISCUSSION AND CONCLUSION

Even though, the experiments we conducted so far are still preliminary, nevertheless the results we obtained are very encouraging and reinforced our initial idea about the fact that modelling schema matching as agent-based simulation could be not only a novel way for resolving the problem but also can contribute to the goal of coping with uncertainty inherent to the automatic schema matching process. We believe that many intrinsic properties of our model, derived from ABMS paradigm, contribute efficiently to the increase of the matching quality and thus the decrease of the matching uncertainty. In fact, we can summarize those intrinsic properties as follows:

- Emergence: The emergence of the macro solution (schema matching) from local behaviors, rules and interactions between agents (micro solutions).
- Cooperation: The cooperation of source and target schema attributes (represented as agents) to reach a consensus about their best matchings.
- Stochasticity (randomness): The randomness on which the model is based makes it possible the ability to statistically analyze the output from multiple simulation runs.

The current release of SAMS, uses only the lexical text similarity measures, we are planning, for the next release, to add semantic similarity measures such as Vector Space Model (VSM) and Latent semantic analysis (LSA).

### REFERENCES

- [1] A. Gal, « Uncertain schema matching », *Synth. Lect. Data Manag.*, vol. 3, n° 1, p. 1–97, 2011.
- [2] F. Klügl et A. L. Bazzan, « Agent-based modeling and simulation », *AI Mag.*, vol. 33, n° 3, p. 29, 2012.
- [3] M. J. North, E. Tatara, N. T. Collier, et J. Ozik, « Visual agent-based model development with repast symphony », Tech. rep., Argonne National Laboratory, 2007.
- [4] D. Bär, T. Zesch, et I. Gurevych, « DKPro Similarity: An Open Source Framework for Text Similarity », in *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, 2013, p. 121–126.

<sup>1</sup> SMAS experiments animations as well as output files: [http://www.researchgate.net/publication/262181441\\_Schema\\_Matching\\_Agent-based\\_Simulation\\_\(SMAS\)\\_Test\\_animation](http://www.researchgate.net/publication/262181441_Schema_Matching_Agent-based_Simulation_(SMAS)_Test_animation)

## Authors' Index

A. Campbell, John	262
Aceituna, Daniel	397
Acuña, Silvia T.	250
Adeodato, Paulo	286
Adjoyan, Seza	1
Adornes, Daniel	25
Aguiar, Rui	463
Akoka, Jacky	684
Al-Msie'Deen, Ra'Fat	138
Alemerien, Khalid	13
Alencar, Antonio Juarez	381
Alvaro de Lima Silva, Luís	262
Anderlin Neto, Amadeu	256
Anvaari, Mohsen	181
Anvik, John	470
Aranha, Eduardo	250
Arnatovich, Yauhen	205, 705
Assoudi, Hicham	334, 748
Assunção, Joaquim	534
Asuncion, Hazeline	292
Audy, Jorge L. N.	412
Augustin, Iara	453
Averbakh, Anna	350, 355
Azevedo, Ryan	82
Badri, Mourad	115
Bae, Doo-Hwan	688
Bagheri, Ebrahim	671
Baik, Doo-Kwon	659
Baioco, Gisele	695
Baker, Chase	187
Barbosa Santos, Adriana	516
Barbosa, Ellen	459
Barbosa, Ellen Francine	678
Bari, Moncef	76
Barnes, Eric	422
Barnes, Eric Christopher	156
Barreiros, Jorge	67
Barreto, Raimundo	603
Barros, Rodolfo	43
Beier, Georg	566
Ben-Abdallah, Hanêne	431
Bender, Lisa	478
Biffli, Stefan	552

Blake, M. Brian	592
Bouassida, Nadia	431
Braga, Rosana	71, 459
Brito Batista, Gleison	522
Brito, Patrick	35
Brooks, Marshall	470
Burton, Henry	470
Cagnin, Maria Istela	459
Campos, Edmilson	250
Canada, Justin	470
Cao, Bei	314
Cassimiro, Dimas	82
Castro Degrossi, Livia	570
Cavalli, Ana	90, 393
Chanda, Nagaprashanth	156
Chang, Hung-Fu	274
Chang, Shi-Kuo	511, 653
Chang, Xuling	314
Che, Xiaoping	90
Chen, Dejian	528
Chen, Deng	19, 121
Chen, Guanling	217
Chen, Jianxun	132
Chen, Jinfu	121
Chen, Lei	507
Chen, Lin	144
Chen, T.Y.	94
Chen, Tsong Yueh	126
Chen, Wen-Hui	511
Chen, Xiangping	53
Chen, Xiangqun	507
Chen, Yin	443
Chen, Zhenyu	620, 723
Chourabi Tan Tan, Olfa	684
Clua, Esteban	495, 522
Colaço, Methanias	474
Connor, Andy	546
Conte, Tayana	235, 256, 603
Correa, Alexandre	381
Costa, Catarina	268
Costa, Evandro	35
Costa, Valeria	318
Cox, David	86
Cristina Silveira dos Santos, Djenane	729
Da Silva Júnior, Jose Ricardo	495
Da Silva, Luciano	459

Dagnino, Aldo	86, 642
Dai, Yafei	636
Dallilo, Felipe	387
Daubal, Mohammed	292
Davis, Delmar	292
De Souza, Cleidson	235
Deb, Debzani	49
Delamaro, Marcio	199
Ding, Junhua	626
Ding, Zuohua	94
Diniz Da Costa, Andrew	330
Diniz De Souza, Adler	729
Dony, Christophe	426, 447
Doran, Derek	642
Dos Anjos Borges, Vanessa	516
Dos Santos, Marco Aurélio	603
Du, Weichang	671
Duarte, Marcos P.	82
Duncan, Nathan	292
Ekaputra, Fajar J.	552
El-Kharboutly, Rehab	615
Englebert, Vincent	744
Eyal-Salman, Hamzeh	426, 447
Fagerholm, Fabian	478
Falvo Junior, Venilton	678
Fan, Ming	740
Fang, Chunrong	723
Fantinato, Marcelo	387
Far, Behrouz	338, 365
Farias, Mário André De Freitas	474
Fava, Maria Clara	570
Feng, Qi	377
Feng, Yang	620
Fernandes, Luiz	25
Fernandes, Paulo	534
Figueiredo, Jose J. C.	268
Finlay, Jacqui	546
Fiondella, Lance	615
Fouquet, Francois	586, 746
Freire, Marilia	250
Freitas Duarte Filho, Nemésio	678
Frez, Liliane	39
Fuad, Mohammad Muztaba	49
Fukazawa, Yoshiaki	711
Gang Wang,	361

Gao, Kehan	280
Gao, Pan-Pan	371
Garcia-Nunes, Pedro I.	695
García, Miguel	150
Gayed, Tamer	76
Gil, Victor N.	199
Gokhale, Swapna	615, 642
Gomede, Everton	43
Gomez, Silvio	534
Goswami, Anurag	735
Greer, Des	324, 576
Griebler, Dalvan	25
Gu, Junzhong	501
Guan, Xiaohong	223
Guessi, Milena	162
Guimaraes, Denilson	381
Guo, Yao	507
Gupta, Munmun	397
Gursimran, Walia	478
Gómez, Marta	250
Habibi, Jafar	168
Hanna, Philip	324
Hardt, Wolfram	566
Harrer, Simon	31
Hartmann, Thomas	586, 746
Haubold, Tobias	566
He, Keqing	371
Hegde, Reshma	229
Heydarnoori, Abbas	168
Hirasaki, Yasuhiro	193
Hong, Gwangui	688
Hong, Jang-Eui	717
Hou, Qiuju	132
Hu, Jianpeng	314
Huang, Linpeng	314
Huang, Rubing	19, 121
Huchard, Marianne	138
Hung, Wen-Ping	511
Hwa, Jimin	688
Hyunsook, Do	397
Jali, Nurfaeza	324
Jarrett, Julian	592
Jedlitschka, Andreas	250
Jiang, Jing	636
Jiang, Mingyue	94
Jiang, Sheng	19

Jing, Nan	274
Kalinowski, Marcos	552
Kamiya, Tomoyuki	711
Kaur, Charanjeet	365
Kawai, Yukiko	665
Khoshgoftaar, Taghi	280, 540
Kiesling, Stephan	355
Kim, Jeong-Dong	659
Klein, Jacques	586, 746
Knauss, Eric	355
Kojima, Hideharu	193
Koshima, Amanuel	744
Krishnamurthy, Diwakar	365
Kroll, Josiane	412
Kulesza, Uirá	250
Kuo, Fei-Ching	94
Lamontagne, Luc	115
Le Traon, Yves	586, 746
Lee, Euijong	659
Lee, Sang-Ho	717
Leung, Hareton K. N.	582
Li, Arturo	648
Li, Liping	598
Li, Xiangyang	223
Li, Xuandong	632
Li, Yongchao	217
Li, You	217
Lian, Xiaoli	63
Liang, Peng	174, 308
Lima, Priscila	381
Lin, Jessica	490
Lin, Zhitian	609
Ling, Jimin	377
Liu, Huai	126
Liu, Jia	723
Liu, Kaiping	705
Liu, Kaping	205
Liu, Ting	223, 740
Liu, Xiaoqing	156, 422
Lopes, Lucelene	534
Lopez, Jorge	90
Lounis, Hakim	76, 334, 748
Lu, Stephen	274
Lui, Kim Man	582
M. Alves, Ana Raquel	82

Ma, Yutao	132
Maag, Stephane	90
Macedo Rodrigues, Elder	402
Macedo, Autran	484
Magel, Kenneth	13
Maia, Marcelo	484
Manzoni Fontoura, Lisandra	262
Mao, Xinjun	443
Maran, Vinfcius	453
Marín, Beatriz	408
Matalonga, Santiago	106
Maâzoun, Jihen	431
Meira, Silvio	82
Memon, Atif	199
Mendiondo, Eduardo Mario	570
Mendonça, Manoel	244, 474
Moaven, Shahrouz	168
Monteiro, Rodrigo	318
Moreira, Ana	67
Morin, Brice	586, 746
Murta, Leonardo	268, 318, 495
Nain, Grégory	586, 746
Nair, Anil	7
Nakagawa, Elisa Yumi	162, 344
Nakagawa, Hiroyuki	742
Napolitano, Amri	280, 540
Nassar, Muhammad	592
Neto, Baldoino	35
Neves Esteca, Antonio Marcos	516
Neves, Glauco	100
Nguyen, Huu Nghia	393
Niklas, Kai	350
Noorian, Mahdi	671
Novais, Ranato	474
Nunes, Fatima L.S.	199
Nygard, Kendall	478
Odzaly, Edzreena Edza	576
Ohsuga, Akihiko	742
Oinuma, Morihide	110
Oliveira Dos Santos, Elanne Cristina	522
Oliveira Junior, Edson	678
Oliveira Neto, Rosalvo	286
Oliveira, Bruno	298
Oliveira, Flavio	402
Oliveira, Lucas Bueno Ruas	344
Oliveira, Rafael A.P.	199

Oquendo, Flavio	162, 344
Ortin, Francisco	150
Ortins, Paulo	474
Osório, Fernando Santos	344
Otunba, Rasaq	490
Pagels, Max	478
Paikari, Elahe	338
Paiva, Débora	459
Palazzo M. de Oliveira, José	453
Panda, Aditi	304
Park, Jae-Jin	717
Park, Jihun	688
Parra Jiménez, Merlin	330
Parvizi-Mosaed, Alireza	168
Passos, Leonardo	402
Pears, Russel	546
Pereira de Lucena, Carlos José	330
Pereira, Oscar	463
Peters, Lawrence	241
Porto De Albuquerque, Joao	387
Porto de Albuquerque, João	570
Qu, Binbin	19
Queiroz, Paulo	71
Quinteros, José	408
Quiroga, Jose	150
Rabelo, Jacilane	603
Rath, Santanu Kumar	304
Rechia Machado, Nielsen Luiz	262
Regateiro, Diogo	463
Richardson, Ita	412
Rocha Machado, Genival	286
Rocha, Ana Regina	560, 729
Rocha, Hemilis	35
Rocha, Rodrigo	82
Rodrigues de Carvalho Filho, Dailton	286
Ruhe, Guenther	437
Röck, Cedric	31
Saad, Rodrigo	402
Saito, Hibiki	110
Sakamoto, Kazunori	711
Saleh, Iman	592
Salgado, Ana Carolina	286
Santos, Gleison	560
Santos, José A. M.	244

Sarma, Anita	495
Satapathy, Shashank Mouli	304
Schmitz, Eber	381
Schneider, Kurt	350, 355
Schots, Natália Chaves Lessa	560
Sei, Yuichi	742
Seo, Dongwon	688
Seriai, Abdelhak	138, 426, 447
Seriai, Abdelhak-Djamel	1
Serral, Estefania	552
Shahnewaz, S. M.	437
Shankar, Mani	7
Shao, Weizhong	507
Shao, Yuxiang	626
Shar, Lwin Khin	205
Sharma, Ashish	701
Shatnawi, Anas	1
Shen, Ning	211
Shen, Yuanhong	416
Shi, Qingkai	620
Shin, Donghwan	688
Shin, Michael	187
Silva de Souza, Laudson	59
Silva, Ana E.	695
Silva, Italo	35
Singh, Deepali	701
Soares de Aquino Jr., Gibeon	59
Solari, Martín	106
Souza, Rogéria C. G.	516
Souza, Simone	298
Stewart, Darryl	576
Sumiya, Kazutoshi	665
Sun, Ding	205
Sun, Yanchun	528
Tahara, Yasuyuki	742
Takada, Shingo	110
Tan, Hee Beng Kuan	205, 705
Tang, Shan	598
Tanno, Haruto	110
Teixeira, Leopoldo	402
Thiry, Marcello	39
Tian, Gang	371
Tian, Zhenzhou	740
Toure, Fadel	115
Tsuchiya, Tatsuhiro	193
Urtado, Christelle	138

Valêncio, Carlos Roberto	516
Vauttier, Sylvain	138
Viana, Davi	235
Vieira de Sousa, Victor Hugo	522
Vilain, Patrícia	100
Walia, Gursimran	229, 397, 735
Wan, Zhiyuan	416
Wang, Chen	620
Wang, Haijun	223, 740
Wang, Huanjing	540
Wang, Jian	371
Wang, Linzhang	217, 632
Wang, Rongcun	121
Wang, Xiao	636
Wang, Ye	416
Wang, Yingze	653
Wang, Yongming	501
Wang, Yuanyuan	665
Washizaki, Hironori	711
Winkler, Dietmar	552
Wirtz, Guido	31
Wu, Di	144
Wu, Qing	723
Xavier, Joicy	484
Xu, Baowen	144
Xu, Dianxiang	211
Xu, Haiping	648
Xu, Xiaoran	723
Xu, Yongrui	174
Xu, Youwei	132
Xu, Zhensheng	53
Xue, Jianxin	598
Yamagata, Satoru	742
Yang, Chen	308
Yang, Haoyu	620
Yang, Linchao	711
Yang, Wenjing	598
Yang, Zijiang	223
Yu, Lechen	223
Zambon, Antonio C.	695
Zapico, Daniel	150
Zhang, Dongmei	626
Zhang, Li	63, 377, 636

Zhang, Lin	507
Zhang, Xiaofang	126
Zhao, Jianhua	632
Zhao, Weidong	371
Zheng, Qinghua	223, 740
Zheng, Tao	609
Zhou, Bo	416
Zhou, Peng	582
Zhou, Xiaoli	632
Zhou, Yuming	144
Zhou, Zhiquan	94
Zhou, Zili	501
Zhuang, Eryue	740
Zhuang, Yi	609
Zimmermann, Olaf	181
Zorzo, Avelino F.	402
Zoucas, Alessandra	39

## Program Committee Reviewers' Index

Silvia Acuna Shadi  
Alawneh Taiseera  
Albalushi Omar  
Ariss  
Doo-Hwan Bae  
Ebrahim Bagheri  
Hamid Bagheri  
Xiaoying Bai  
Purushotham Bangalore  
Fevzi Belli  
Ateet Bhalla  
Swapan Bhattacharya  
Alessandro Bianchi  
Ivo Bukovsky  
Jaelson Castro  
Keith Chan  
Kuang Nan Chang  
Shikuo Chang  
Ned Chapin  
Meiru Che  
Shu-Ching Chen  
Wen-Hui Chen  
Zhenyu Chen  
Stelvio Cimato  
Peter Clarke  
Esteban Clua  
Nelly Condori-Fernandez  
Fabio Costa  
Maria Costabile  
Jose Cuadrado  
Aldo Dagnino  
Jose De La Vara  
Massimiliano Di Penta  
Scott Dick  
Junhua Ding  
Jing Dong  
Weichang Du  
Philippe Dugerdil  
Christof Ebert Ali  
Ebneansir  
Raimund Ege  
Magdalini Eirinaki  
Omar El Ariss  
Davide Falessi  
Behrouz Far  
Liana Fong

Ellen Francine  
Fulvio Frati  
Jerry Gao  
Kehan Gao  
Felix Garcia  
Raul Garcia  
Ignacio Garcia Rodriguez De  
Guzman  
Vahid Garousi  
Swapna Gokhale  
Wolfgang Golubski  
Desmond Greer  
Christiane Gresse Von Wangen-  
heim  
Katarina Grolinger  
Irit Hadar  
Hassan Haghighi  
Hao Han  
Xudong He  
Miguel Herranz  
Dan Hoffman  
Rubing Huang  
Bassef Isong  
Clinton Jeffery  
Jason Jung  
Taghi Khoshgoftaar  
Claudiu Kifor  
Jun Kong  
Aneesh Krishna  
Vinay Kulkarni  
Jeff Lei  
Meira Levy  
Bixin Li  
Ming Li  
Zhi Li  
Chien Hung Liu  
Shaoying Liu  
Shih Hsi Liu  
Xiaodong Liu  
Yi Liu  
Luanna Lopes Lobato  
Hakim Lounis  
Baojun Ma  
Marcelo De Almeida Maia  
Jose Maldonado  
Vijay Mann  
Riccardo Martoglia  
Santiago Matalonga

Hong Mei  
Hsing Mei  
Andre Menolli  
Michele  
Ali Mili  
Alok Mishra  
Manuel Mora  
Henry Muccini  
Kia Ng  
Allen Nikora  
Amjad Nusayr  
Edson Oliveira Junior  
Amit Paradkar  
Xin Peng Oscar  
Pereira Dragutin  
Petkovic Antonio  
Piccinno Alfonso  
Pierantonio Daniel  
Plante Huseyin  
Polat  
Rick Rabiser Filip  
Radulovic Damith  
Rajapakse Rajeev  
Raje  
Jose Ramos Sivan  
Rapaport Kattur  
Ravichandran  
Henrique Rebelo  
Marek Reformat  
Robert Reynolds  
Ivan Rodero Daniel  
Rodriguez Samira  
Sadaoui Masoud  
Sadjadi Ahmed  
Salem Farshad  
Samimi Claudio  
Sant'Anna  
Andreas Schoenberger  
Stefan Schulte  
Ashish Sharma  
Michael Shin  
Qinbao Song  
Wei Song  
George Spanoudakis  
Rajesh Subramanyan  
Jing Sun  
Yanchun Sun  
Gerson Sunye

Chuanqi Tao  
Jeff Tian  
Genny Tortora  
Mark Trakhtenbrot  
Peter Troeger  
T. H. Tse  
Burak Turham  
Hasan Ural  
Giorgio Valle  
Sylvain Vauttier  
Silvia Vergilio  
Marlon Vieira  
Sergiy Vilkomir  
Aaron Visaggio  
Arndt Von Staa  
Gurisimran Walia  
Huanjing Wang  
Jiacun Wang  
Jiacun Wang  
Linzhang Wang  
Xiaoyin Wang  
Ye Wang  
Hironori Washizaki  
Claudia Werner  
Victor Winter  
Guido Wirtz  
Eric Wong  
Franz Wotawa  
Dianxiang Xu  
Frank Xu  
Haiping Xu  
Weifeng Xu  
Chi-Lu Yang  
Hongji Yang  
Huiqun Yu  
Jiang Yue  
Du Zhang  
Hongyu Zhang  
Yong Zhang  
Zhenyu Zhang  
Lei Zhao  
Jiang Zheng  
Hong Zhu  
Jianlin Zhu  
Xingquan Zhu  
Eugenio Zimeo

## External Reviewers' Index

Allison, Mark  
Alsmadi, Izzat  
Aranda López King, Alejandrina Marfa  
Assunção, Wesley K. G.

Baral, Lal Mohan Ben-  
Assuli, Ofir  
Biokaghazadeh, Saman  
Borges, Vanessa  
Buono, Paolo

Castro Llanos, John Wilmar  
Chang, S.K.  
Chen, Hongjie

Desolda, Giuseppe

Fang, Chunrong  
Fazelpour, Alireza  
Fleites, Fausto  
Fonseca C., Efrain R.

Gallege, Lahiru  
Gamage, Dimuthu  
Ghandehari, Laleh  
Ghassemi, Fatemeh  
Goldstein, Anat  
Gonzalez, Rafael  
Guedes, Gabriela

Hao, Dan Harrer,  
Simon Hendijani,  
Fatemeh  
Heydarnoori, Abbas

Ibrahim, Hamdy

Kohwalter, Troy Kolb,  
Stefan Krishnamurthy,  
Diwakar

Lampe, Ulrich  
Lenhard, Jörg  
Li, Xuelin  
Li, Yihao  
Li, Zhongwei

Liu, Zicong

Mahbub, Khaled  
Mohamed, Mostafa  
Mohamed, Tamer  
Morris, Karl  
Mottu, Jean-Marie  
Mousavi, Sayed  
Moussavi, Mahmood

Oconnor, Rory  
Orrù, Matteo  
Ortu, Marco

Penzo, Wilma  
Petkov, Doncho  
Phillips-Wren, Gloria  
Pigatto, Daniel F.

Raisinghani, Mahesh  
Rehman, Zobia Risi,  
Michele Rodríguez,  
Francy D. Rybarczyk,  
Ryan

Sadaoui, Samira  
Shi, Qingkai  
Soares, Monique  
Souza, Draylson M.

Tarihi, Ali  
Tibermacine, Chouki  
Torkashvan, Milad

Urtado, Christelle  
Uslar, Mathias

Vessio, Gennaro

Wang, Fen

Xu, Jing

Yabin, Wang  
Yang, Yimin  
Yingfei, Xiong

Zhang, Wei



# SEKE

2014

Vancouver  
July 1-3, 2014

**Proceedings of the Twenty-Sixth  
International Conference on  
Software Engineering  
& Knowledge Engineering**

Copyright © 2014

Printed by

Knowledge Systems Institute

Graduate School

3420 Main Street

Skokie, Illinois 60076

(847) 679-3135

office@ksi.edu

www.ksi.edu

Printed in USA, 2014

ISBN 1-891706-35-7 (paper)

ISSN 2325-9000 (print)

