SEKE 2013

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

Boston
June 27-29, 2013

PROCEEDINGS

SEKE 2013

The 25th International Conference on Software Engineering & Knowledge Engineering

Sponsored by

Knowledge Systems Institute Graduate School, USA

Technical Program

June 27 - 29, 2013

Hyatt Harborside at Logan Int'l Airport, Boston, USA

Organized by

Knowledge Systems Institute Graduate School

ISBN-10: 1-891706-33-0 (paper)
ISBN-13: 978-1-891706-33-2

ISSN: 2325-9000 (print)

Additional Copies can be ordered from:
Knowledge Systems Institute Graduate School
3420 Main Street
Skokie, IL 60076, USA
Tel:+1-847-679-3135
Fax:+1-847-679-3166
Email:office@ksi.edu
http://www.ksi.edu

Proceedings preparation, editing and printing are sponsored by
Knowledge Systems Institute Graduate School

Printed by Knowledge Systems Institute Graduate School

Foreword

This year marks the 25th anniversary for the International Conference on Software Engineering and Knowledge Engineering (SEKE). For a quarter of century, 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 covering the interplay 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 2013.

We received 259 submissions from 40 countries this year, marking an increase of approximately 15 percent in the number of papers, and 33 percent in the number of countries.. Through a rigorous review process where a majority (88 percent) of the submitted papers received three reviews, and the rest with two reviews, we were able to select 72 full papers for the general conference (27 percent), 5 full papers for four special tracks (2 percent), and 78 short papers (30 percent), for presentation in forty one sessions during the conference. In addition, the technical program includes excellent keynote speech and panel discussions, and four special tracks: Mining Social Network Data for Industrial Applications, Petri Nets for Emerging Technologies, Knowledge Management in Software Engineering and Slow Intelligence Systems.

The high quality of the SEKE 2013 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 speakers who graciously agreed to share their insight on important research issues, to the conference organizing committee members for their superb work, and to the external reviewers for their contribution.

Personally, I owe a debt of gratitude to a number of people whose help and support with the technical program and the conference organization are unfailing and indispensable. I am deeply indebted to Dr. S. K. Chang, Chair of the Steering Committee, for his constant guidance and support that was essential to pull off SEKE 2013. My heartfelt appreciation goes to Dr. Du Zhang, the Conference Chair, for his help and experience, and to the Program Committee Co-Chairs, Dr. Marek Reformat of the University of Alberta, Canada, and Dr. Haiping Xu of the University of Massachusetts, Dartmouth, USA for their outstanding team work. I am truly grateful to the special track organizers, Dr. Aldo Dagnino of ABB, USA, Dr. Swapna Gokhale of the University of Connecticut, USA, Dr. Dianxiang Xu of the Dakota State University, USA , Dr. Sivan Rapaport of the Columbia Business School, USA, Dr. Meira Levy of the Shenkar College of Engineering and Design, Israel, Dr. Shi-Kuo Chang of the University of Pittsburgh, USA, and Dr. Irit Hadar of the University of Haifa, Israel, for their excellent job in organizing the special sessions. I would like to express my great appreciation to the Publicity Chair Dr. Xiaoying Bai of Tsinghua University, China; to Asia, Europe, and South America liaisons Dr. Raul Garcia Castro of Universidad Politecnica de Madrid, Spain, Dr. Hironori Washizaki, Waseda University, Japan, and Jose Carlos Maldonado, ICMC-USP, Brazil and to the Poster/Demo session Co-Chairs, Dr. Farshad Samimi of Trilliant, USA and Dr. Ming Zhao of Florida International University, USA. Last but certainly not the least, I feel compelled to acknowledge all the timely help that I have always received from the following KSI staff members: David Huang, Alice Wang, and Dennis Chi. Their enthusiastic and round-the-clock support and assistance throughout the entire process has been truly remarkable. It has been a great pleasure to work with all of them.

Finally, I hope you will find your participation in the SEKE 2013 programs rewarding. Enjoy your stay in the Bay State and Beantown.

Swapna S. Gokhale
SEKE 2013 Program Chair

The 25th International Conference on Software Engineering & Knowledge Engineering (SEKE 2013)

June 27 - 29, 2013
Hyatt Harborside at Logan Int'l Airport, Boston, USA

Conference Organization

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, *Madrid Technological University, Spain*
Taghi Khoshgoftaar, *Florida Atlantic University, USA*
Guenther Ruhe, *University of Calgary, Canada*
S. Masoud Sadjadi, *Florida International University, USA*

Conference Chair

Du Zhang, *California State University, Sacramento, USA*

Program Chair

Swapna Gokhale, *University of Connecticut, USA*

Program Co-Chairs

Marek Reformat, *University of Alberta, Canada*
Haiping Xu, *University of Massachusetts Dartmouth, USA*

Program Committee

Silvia Teresita Acuna, *Universidad Autonoma de Madrid, Spain*

Taiseera Albalushi, *Sultan Qaboos University, Oman*

Edward Allen, *Mississippi State University, USA*

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*

Rami Bahsoon, *University of Birmingham, United Kingdom*

Xiaoying Bai, *Tsinghua University, China*

Purushotham Bangalore, *University of Alabama at Birmingham, USA*

Fevzi Belli, *Univ. Paderborn, Germany*

Ateet Bhalla, *Oriental Institute of Science & Technology, Bhopal, India.*

Swapan Bhattacharya, *Jadavpur University, India*

Alessandro Bianchi, *Department of Informatics - University of Bari, Italy*

Borzoo Bonakdarpour, *University of Waterloo, Canada*

Ivo Bukovsky, *Czech Technical University in Prague, Czech Republic*

Gerardo Canfora, *Universita del Sannio, Italy*

Jaelson Castro, *Universidade Federal de Pernambuco - UFPE, Brazil*

Raul Garcia Castro, *Universidad Politecnica de Madrid, Spain*

Peggy Cellier, *IRISA/INSA of Rennes, France*

Keith Chan, *The Hong Kong Polytechnic University, Hong Kong*

Kuang-nan Chang, *Eastern Kentucky University, USA*

Ned Chapin, *InfoSci Inc., USA*

Shu-Ching Chen, *Florida International University, USA*

Wen-Hui Chen, *National Taipei University of Technology, Taiwan*

Zhenyu Chen, *Nanjing University, China*

Stelvio Cimato, *The University of Milan, Italy*

Peter Clarke, *Florida International University, USA*

Esteban Clua, *Universidade Federal Fluminense, Brasil*

Nelly Condori-fernandez, *University of Twente, The Netherlands*

Fabio M. Costa, *Instituto de Informatica, Brasil*

Maria Francesca Costabile, *University of Bari, Italy*

Jose Luis Cuadrado, *University of Alcala, Spain*

Juan J. Cuadrado-gallego, *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 Ebnenasir, *Michigan Technological University, USA*

Raimund Ege, *NIU, USA*

Magdalini Eirinaki, *Computer Engineering Dept, San Jose State University, USA*

Davide Falessi, *University of Rome, TorVergata, Italy*

Behrouz Far, *University of Calgary, Canada*

Scott D. Fleming, O*regon State University, USA*

Liana Fong, *IBM, USA*

Renata Fortes, *Instituto de Ciencias Matematicas e de Computacao - USP, Brazil*

Ellen Francine Barbosa, *University of Sao Paulo, Brazil*

Fulvio Frati, *The 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*

Itana Gimenes, *Universidade Estadual de Maringa, Brazil*

Swapna Gokhale, *Univ. of Connecticut, USA*

Wolfgang Golubski, *Zwickau University of Applied Sciences, Germany*

Desmond Greer, *Queen's University Belfast, United Kingdom*

Eric Gregoire, *Universite d'Artois, France*

Christiane Gresse Von Wangenheim, *UFSC - Federal University of Santa Catarina, Brazil*

Katarina Grolinger, *University of Western Ontario, Canada*

Hao Han, *National Institute of Informatics, Japan*

Xudong He, *Florida International University, USA*

Miguel Herranz, *University of Alcala, Spain*

Clinton Jeffery, *University of Idaho, USA*

Jason Jung, *Yeungnam University, South Korea*

Natalia Juristo, *Universidad Politecnica de Madrid, Spain*

Selim Kalayci, *Florida International University, USA*

Eric Kasten, *Michigan State University, USA*

Taghi Khoshgoftaar, *Florida Atlantic University, USA*

Jun Kong, *North Dakota State University, USA*

Nicholas Kraft, *The University of Alabama, USA*

Aneesh Krishna, *Curtin University of Technology, Australia*

Vinay Kulkarni, *Tata Consultancy Services, India*

Gihwon Kwon, *Kyonggi University, South Korea*

Jeff Lei, *University of Texas at Arlington, USA*

Bixin Li, *School of Computer Science and Engineering, Southeast University, China*

Ming Li, *Nanjing University, China*

Tao Li, *Florida International University, USA*

Yuan-Fang Li, *Monash University, Australia*

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*

Hakim Lounis, *UQAM, Canada*

Joan Lu, *University of Huddersfield, United Kingdom*

Marcelo de Almeida Maia, *Federal University of Uberlândia, Brazil*

Antonio Mana, *University of Malaga, Spain*

Vijay Mann, *IBM, India*

Riccardo Martoglia, *University of Modena and Reggio Emilia, Italy*

Hong Mei, *Peking University, China*

Hsing Mei, *Fu Jen Catholic Unicersity, Taiwan*

Ali Mili, *NJIT, USA*

Alok Mishra, *Atilim University, Turkey*

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*

Erick Passos, *IFPI, 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*

Rick Rabiser, *Johannes Kepler University, Austria*

Filip Radulovic, *Universidad Politécnica de Madrid, Spain*

Damith C. Rajapakse, *National University of Singapore, Singapore*

Rajeev Raje, *IUPUI, USA*

Jose Angel Ramos, *Universidad Politecnica de Madrid, Spain*

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*

Salvatore Alessandro Sarcia, *OnESE FORUM, Italy*

Andreas Schoenberger, *Distributed and Mobile Systems Group, University of Bamberg, Germany*

Tony Shan, *Keane Inc, USA*

Michael Shin, *Computer Science/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*

Giorgio Valle, *The University of Milan, Italy*

Sylvain Vauttier, *Ecole des mines d'Ales, France*

Silvia Vergilio, *Federal University of Parana (UFPR), Brazil*

Akshat Verma, *IBM, India*

Sergiy Vilkomir, *East Carolina University, USA*

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, USA*

Hironori Washizaki, *Waseda University, Japan*

Victor Winter, *University of Nebraska at Omaha, USA*

Guido Wirtz, *Distributed Systems Group, Bamberg University, Germany*

Eric Wong, *University of Texas, USA*

Franz Wotawa, *TU Graz, Austria*

Dianxiang Xu, *Dakota State University, USA*

Frank Xu, *Gannon University, USA*

Haiping Xu, *University of Massachusetts Dartmouth, USA*

Chi-lu Yang, *Taiwan Semiconductor Manufacturing Company Ltd., Taiwan*

Hongji Yang, *De Montfort University, United Kingdom*

Jijiang Yang, *Tsinghua University, China*

Huiqun Yu, *East China University of Science and Technology, China*

Cui Zhang, *California State University Sacramento, USA*

Du Zhang, *California State University, USA*

Hongyu Zhang, *Tsinghua University, China*

Yong Zhang, *TsingHua University in Beijing, China*

Zhenyu Zhang, *The University of Hong Kong, Hong Kong*

Hong Zhu, *Oxford Brookes University, United Kingdom*

Xingquan Zhu, *Florida Atlantic University, USA*

Eugenio Zimeo, *University of Sannio, Italy*

Poster/Demo Sessions Co-Chairs

Farshad Samimi, *Trilliant, USA*
Ming Zhao, *Florida Int'l University, USA*

Publicity Chairs

Xiaoying Bai, *Tsinghua University, China*

Asia Liaison

Hironori Washizaki, *Waseda University, Japan*

Europe Liaison

Raul Garcia Castro, *Universidad Politecnica de Madrid, Spain*

South America Liasion

Jose Carlos Maldonado, *University of Sao Paulo, Brazil*

Proceedings Cover Design

Gabriel Smith, *Knowledge Systems Institute Graduate School, USA*

Local Arrangements

Judy Pan, *Chair, Knowledge Systems Institute Graduate School, USA*
Gabriel Smith, *Knowledge Systems Institute Graduate School, USA*
Noorjhan Ali, *Knowledge Systems Institute Graduate School, USA*
Dennis Chi, *Knowledge Systems Institute Graduate School, USA*
David Huang, *Knowledge Systems Institute Graduate School, USA*
Alice Wang, *Knowledge Systems Institute Graduate School, USA*
Farida Begum, *Knowledge Systems Institute Graduate School, USA*

Table of Contents

Requirements Engineering

Cloud Computing

Software Engineering Decision Support

Social Media

Human Computer Interaction

Data and Knowledge Visualization

Software Engineering Tools and Environments

Structural Testing and Coverage

Software Product Lines

Software Domain and Meta-Modeling

Slow Intelligence and Intelligent Systems

Quality and Reliability

Recommender Systems

Web and Data Mining

Software Architecture

Software Maintenance

Security and Fault Tolerance

Petri Nets

Pervasive Computing

Software Architecture and Quality

Measurement and Empirical SE

Mobile Systems

Software Maintenance and Quality

Web-based Knowledge Management

Knowledge Management in Software Engineering

Testing and Fault Diagnosis

Programming Languages and Software Engineering

Artificial Intelligence Approaches to Software Engineering

Software Process and Quality

Semantic Analysis

Agents and Ontologies

Embedded and Ubiquitous Software Engineering

Embedded and Ubiquitous Software Engineering

Adaptive Systems

Software Maintenance and Evolution

Poster/Demo

Note: (S) indicates a short paper.
(P) indicates a poster or demo, which is not a refereed paper.

Keynote I
Surprising discoveries from emotion sensors

Professor Rosalind W. Picard
Founder and Director, Affective Computing Research Group
Media Lab, Massachusetts Institute of Technology, USA

Abstract

Emotion is much more vital to health and cognition than we ever thought - influencing pretty much every organ in our body, not just our brain and “how we feel.” We built a camera to read heart rate, and a wearable sensor to measure a key dimension of emotion and encountered some huge surprises during long-term measurement - from seizure detection to mysterious “storms” that happen mostly during non-REM sleep. We have also been learning how brain activity can map to different places on the surface of your skin. This talk will highlight new technologies and insights that have come out of our lab including Q sensor, CardioCam, and MoodMeter.

About the Speaker

Professor Rosalind W. Picard, Sc.D. is founder and director of the Affective Computing research group at the Massachusetts Institute of Technology (MIT) Media Lab. She is also co-founder of Affectiva, Inc., delivering technology to help measure and communicate emotion.

Picard holds a bachelor’s degree in electrical engineering with highest honors from the Georgia Institute of Technology, and master’s and doctoral degrees, both in electrical engineering and computer science, from MIT. Prior to completing her doctorate, she was a member of the technical staff at AT&T Bell Laboratories where she designed VLSI chips for digital signal processing and developed new methods of image compression and analysis. In 1991 she joined the MIT Media Lab faculty. She became internationally known for constructing mathematical texture models for content-based retrieval of images, for creating new tools such as the Photobook system, and for pioneering methods of automated search and annotation in digital video. She published the award-winning book/Affective Computing,/which was instrumental in starting a new field by that name. Picard has been awarded dozens of distinguished and named lectureships internationally, and in 2005 was honored as a Fellow of the IEEE.

The author of over two hundred scientific articles and chapters in multidimensional signal modeling, computer vision, pattern recognition, machine learning, human-computer interaction, and affective computing, Picard is an international leader in envisioning and creating innovative technology.

She holds multiple patents, having designed and developed a variety of new sensors, algorithms, and systems for sensing, recognizing, and responding respectfully to human affective information, with applications in autism, epilepsy, autonomic nervous system disorders, sleep, stress, human and machine learning, health behavior change, market research, customer service, and human-computer interaction.

Picard interacts regularly with industry and has consulted for companies such as Apple, AT&T, BT, HP, iRobot, and Motorola. She is a popular keynote speaker and has given over 100 keynote talks. Her group’s achievements have been featured in forums for the general public such as/The New York Times, The London Independent,/National Public Radio,/Scientific American Frontiers,/ABC’s/ Nightline/and/World News Tonight with Peter Jennings, Time, Vogue, Wired,/Voice of America Radio,/New Scientist,/and BBC’s “The Works” and “The Big Byte.”.

Keynote II
Environment-Aware Software Engineering

Professor Shi-Kuo Chang
Dept. of Computer Science
University of Pittsburgh, USA

Abstract

In this talk I will introduce the concept of slow intelligence as a new approach for environment-aware software engineering. Not all intelligent systems have fast intelligence. There are a surprisingly large number of intelligent systems, quasi-intelligent systems and semi-intelligent systems that have slow intelligence. Such slow intelligence systems are often neglected in mainstream research on intelligent systems, but they are really worthy of our attention and emulation. I will discuss the general characteristics of slow intelligence systems and then concentrate on personal health care as an example of artificial slow intelligence systems. Other applications to social network modelling, product customization and image processing will also be discussed.

About the Speaker

Dr. Chang received the B.S.E.E. degree from National Taiwan University in 1965. He received the M.S. and Ph.D. degrees from the University of California, Berkeley, in 1967 and 1969, respectively. He was a research scientist at IBM Watson Research Center from 1969 to 1975. From 1975 to 1982 he was Associate Professor and then Professor at the Department of Information Engineering, University of Illinois at Chicago. From 1982 to 1986 he was Professor and Chairman of the Department of Electrical and Computer Engineering, Illinois Institute of Technology. From 1986 to 1991 he was Professor and Chairman of the Department of Computer Science, University of Pittsburgh. He is currently Professor and Director of the Center for Parallel, Distributed and Intelligent Systems, University of Pittsburgh. Dr. Chang is a Life Fellow of IEEE. He published over 230 papers and 16 scientific books. He is the founder and co-editor-in-chief of the international journal, Visual Languages and Computing, published by Academic Press, the founder and editor-in-chief of the international journal, Software Engineering & Knowledge Engineering, published by World Scientific Press, and the co-editor-in-chief of the international journal on Distance Education Technologies. Dr. Chang pioneered the development of Chinese language computers, and was the first to develop a picture grammar for Chinese ideographs, and invented the phonetic phrase Chinese input method..

Keynote III
Overcoming Big Data Challenges

Professor Taghi M. Khoshgoftaar
Dept. of Computer & Electrical Engineering & Computer Science
Florida Atlantic University, USA

Abstract

Due to the influx of data across a wide variety of application domains, Big Data has become a central topic in data science research. Big Data provides many opportunities to learn key insights which can only be found from large collections of data, but also poses unique challenges when practitioners are faced with data characteristics which are much more difficult to address on large-scale data. For example, high-dimensionality (having a large number of independent attributes or features) can occur at multiple scales of data mining, but extremely large datasets are not amenable to some traditional approaches. In addition, data imbalance (having many more instances in one class than in other classes) can be especially challenging when large datasets make oversampling infeasible. Finally, one oft-overlooked challenge -- datasets which are inherently difficult to learn from -- is even more difficult to handle with extremely large quantities of noise. In this paper, we will discuss these problems in the context of one important Big Data application domain -- bioinformatics -- and present our work on addressing these challenges, though using techniques designed to solve these problems while operating efficiently and returning meaningful results even when faced with Big Data.

About the Speaker

Dr. Taghi M. Khoshgoftaar is a professor of the Department of Computer and Electrical Engineering and Computer Science, Florida Atlantic University and the Director of the Data Mining and Machine Learning Laboratory, and Empirical Software Engineering Laboratory. His research interests are in big data analytics, data mining and machine learning, health informatics and bioinformatics, and software engineering. He has published more than 500 refereed journal and conference papers in these areas. He was the conference chair of the IEEE International Conference on Machine Learning and Applications (ICMLA 2012). He is the workshop chair of the IEEE IRI Health Informatics workshop (2013). He is the Editor-in Chief of the Big Data journal. He has served on organizing and technical program committees of various international conferences, symposia, and workshops. Also, he has served as North American Editor of the Software Quality Journal, and was on the editorial boards of the journals Multimedia Tools and Applications, Knowledge and Information Systems, and Empirical Software Engineering and is on the editorial boards of the journals Software Quality, Software Engineering and Knowledge Engineering, Fuzzy Systems, and Social Network Analysis and Mining.

Runtime Values Driven by Access Control Policies

Statically Enforced at the Level of Relational Business Tiers

Óscar Mortágua Pereira[1], Rui L. Aguiar[2]
Instituto de Telecomunicações
DETI, University of Aveiro
Aveiro, Portugal
{omp[1], ruilaa[2]}@ua.pt

Maribel Yasmina Santos
Centro Algoritmi
DSI, University of Minho
Guimarães, Portugal
maribel@dsi.uminho.pt

***Abstract*—Access control is a key challenge in software engineering, especially in relational database applications. Current access control techniques are based on additional security layers designed by security experts. These additional security layers do not take into account the necessary business logic leading to a separation between business tiers and access control mechanisms. Moreover, business tiers are built from commercial tools (ex: Hibernate, JDBC, ODBC, LINQ), which are not tailored to deal with security aspects. To overcome this situation several proposals have been presented. In spite of their relevance, they do not support the enforcement of access control policies at the level of the runtime values that are used to interact with protected data. Runtime values are critical entities because they play a key role in the process of defining which data is accessed. In this paper, we present a general technique for static checking, at the business tier level, the runtime values that are used to interact with databases and in accordance with the established access control policies. The technique is applicable to CRUD (create, read, update and delete) expressions and also to actions (update and insert) that are executed on data retrieved by Select expressions. A proof of concept is also presented. It uses an access control platform previously developed, which lacks the key issue of this paper. The collected results show that the presented approach is an effective solution to enforce access control policies at the level of runtime values that are used to interact with data residing in relational databases.**

Keywords-security; access control; database, business tiers; software architecture.

I. INTRODUCTION

Sensitive data is growing every day as an immediate consequence of the increasing usage of software systems. The data is related not only to personal information, as it happens for example in social networks, but it is also related to other important and critical areas such as commercial, institutional and security organizations. To prevent any security violation, several security measures are taken such as user authentication, data encryption and secure connections. Another relevant security concern is access control. There are two main approaches to enforce access control policies: the one provided by vendors of database management systems and XACML [1] (eXtensible Access Control Markup Language). Both approaches rely on additional security layers built by security experts leading to a clear separation between the security mechanisms and business tiers. Moreover, current commercial tools that are used to develop business tiers do not support access control policies, this way hampering the process of bridging the gap between access control mechanisms and business tiers built from those tools. To overcome this situation, several access control techniques have been proposed [2-13] but none of them effectively models the values that are defined at runtime. The runtime values are critical because they are dynamically defined by users at runtime, this way enabling users to request the access to different data in each execution cycle. We present three examples to justify our claims. The first one is based on a native Select expression, the second one is based on a native Update expression and, finally, the third one is based on modifying the contents of a record set containing data retrieved by a Select expression (in these cases the modifications are also committed to the host database). The following example is a simple Select expression.

```
Select t1.* from table1 t1, table2 t2
      where t1.id = t2.t1_id and
            t1.value > pValue
```

The parameter (runtime value) *pValue* plays a key role to decide which data are retrieved from *table1*. In each individual execution cycle, the parameter may have a different value, this way retrieving a different set of records from *table1*. To overcome this source of possible security gaps, two approaches are used to implement the access control mechanisms: centralized approach and distributed approach. Regarding the centralized approach, the most common technique is the use of views (with [10] or without query rewriting techniques). This technique conveys several drawbacks among which the lack of scalability is emphasized [14, 15]. Regarding the distributed approach, two techniques were proposed: in [4] is proposed a new predicate, identified as *known*, to model which information users already know, this way covering the points here under discussion but only superficially; in [2] the policies are statically enforced at the table columns level and not at the CRUD (Create, Read, Update and Delete) expressions level, leading to lack of flexibility.

The following example is the second example, which is a simple Update expression:

```
Update table1 t1 set t1.value=pValue
       Where t1.id=pId
```

Similar to the Select expression, this Update expression also uses parameters. The parameter *pValue* updates the attribute *value* of *table1* of a record identified by another parameter *pId*. Once again, parameters are user defined and play a key role on Update expressions to decide the data to be updated. The current techniques and their limitations, previously described for Select expressions, are also applied to Update expressions. The remaining types of CRUD expressions, Insert and Delete, convey similar limitations.

The last example is a very common situation on current tools that are used to develop business tiers, such as JDBC [16], Hibernate [17], ADO.NET [18] and LINQ [19]. The example shows that beyond the use of CRUD expressions, databases are also modifiable by executing protocols on data retrieved by Select expressions. The example shows that after retrieving data from a database, it is kept in record sets (*recordSet*) and then applications are allowed to update their content through an update protocol. In this case the attribute *attributeName* was updated to *value* and then the modification was committed. This case is different from the two previous ones because there is no evidence of any CRUD expression and users are modifying data they have been previously authorized to retrieve. Even so, we cannot despise the need to control the runtime values being used to modify the contents of those record sets and, therefore, used to modify the contents of databases. Beyond the update protocol, current tools also provide an insert protocol where users are also allowed to use runtime values.

```
recordSet=executeSelectExpression(sql)
recordSet.update("attributeName", value)
recordsSet.commit()
```

Currently, there isn´t any known access control technique to enforce policies at the business tier level and able to statically control the provenance of runtime values that are used on actions issued against databases. To overcome this situation we propose a technique where parameters are statically driven by access control policies enforced at the business tier level. Additionally, we present a proof of concept to validate the proposed technique. The proof of concept leverages an existent and internal access control platform, partially based on [13].

This paper is organized as follows. Section II presents the related work. Section III presents the required background to keep the paper self-contained. Section IV describes the conceptual architecture and, finally, section V presents the final conclusion.

II. RELATED WORK

Views have been widely used to restrict the access to protected data. In spite of their relevance, the use of views to implement access control conveys a key drawback: lack of scalability [14, 15]. Basically the number of views increases with the number of policies. Access control based on views is easily managed in database applications with a short number of policies. But access control in database applications with a large number of policies may become unmanageable as in cases where they depend, for example, on data stored on databases. Moreover, the problem is not restricted to the level of views. Users accessing the same table but with different authorization levels use different views and, therefore, different CRUD expressions. In order to minimize this scalability gap, Rizvi et al. [10] present a query rewriting technique to determine at runtime if a CRUD expression is authorized, without the need of creating different versions of views. It uses security views to filter contents of tables and simultaneously to infer and check at runtime the appropriate authorization to execute any CRUD expression issued against the unfiltered table. The user is responsible for formulating the CRUD expression properly. They call this approach the Non-Truman model. Non-Truman models, unlike Truman models, do not change the original CRUD expressions. The process is transparent for users, and CRUD expressions are rejected if they do not have the appropriate authorization. This approach has some disadvantages: 1) performance - the inference rules to check the appropriate authorization at runtime are complex and time consuming; 2) productivity - authorizations are checked against security views and not against original data in a transparent way, hampering the debugging process when any syntax error or security violation occurs; 3) awareness - programmers cannot statically check the correctness of CRUD expressions because the policies and the mechanisms are centralized in a server; 4) incompleteness - the inference rules are complex and their completeness is not assured by the authors.

In [4], Chlipala et al. present a tool, *Ur/Web*, that allows programmers to write statically-checkable access control policies as CRUD expressions. Basically, each policy determines which data is accessible. Then, programs are written and checked to assure that data involved in queries is accessible through some policy. To allow policies to vary from one user to another, their CRUD expressions use actual data and a new extension to the standard SQL to capture '*which secrets the user knows'*. This extension is based on a predicate referred to as *´known*' used to model which information users are already aware of to decide upon the information to be disclosed. Ur/Web is a promising solution, but beyond introducing a new programming technique, it presents two key drawbacks: 1) it does not check the use of runtime values of *where* clauses, allowing queries to implicitly leak protected data; 2) authors say that their implementation "…*only handles a subset of the common SQL features.*".

Caires et al. [2] introduces a new programming language, name as λ, to define and enforce access control policies by static typing. The security model comprises tables, their attributes and the access control policies associated to each attribute. Authors show that runtime values are checked against the policies before being used. Beyond introducing a new programming language, policies are enforced at the attribute level of tables, this way hindering or even preventing the use of multiple policies on each attribute.

The paper [13] presents an access control-driven architecture with dynamic adaptation (ACADA). Business tiers are automatically built from a business architectural model, enforcing access control policies defined by a security expert. Access control mechanisms are statically

implemented by typed objects driven by security policies at the business tier level. ACADA effectively controls which CRUD expressions are authorized to be used but does not control the runtime values being used.

III. BACKGROUND

To ease the development process of business tiers, system architects use tools specially designed to that end. Two main groups of tools are considered: Call Level Interfaces (CLI) [20] and Object-to-Relational Mapping (O/RM) tools. ODBC [21], JDBC [16] and ADO.NET [22] are three examples of CLI and Hibernate [17], LINQ [23] and JPA [24] are three examples of O/RM tools. These tools provide services to allow applications to interact with databases. These services need to be understood before advancing to any security solution implemented at the business tier level. In spite of the diversity of tools and the difference between the paradigms of the two groups, there is a common basis between them. This is very important to promote the use of a single technique in all tools and mainly on both groups. The common basis is centered on their two main access modes to stored data: direct access mode and the indirect access mode. The direct access mode allows the execution of CRUD expressions written in the native SQL language and the indirect access mode allows applications to interact with data returned by Select expressions. While the direct access mode is widely used and easily understood, the indirect access mode needs a more detailed explanation. When a Select expression is executed, it returns a relation containing the retrieved data. These relations are locally managed by local memory structures (LMS). There are four protocols to interact with the data managed by LMS: read protocol (to read data from LMS), update protocol (to update data contained in LMS), insert protocol (to insert new data in LMS) and delete protocol (to delete data contained in LMS). Any modification on the contents of LMS is replicated on the host database. Figure 1 and Figure 2 depict a simple example based on JDBC and LINQ, respectively. The method *updateStudentMobilePhone* updates numbers of mobile phones of every student whose *id* is contained in the first argument (*sId*). The Select expression is built with two parameters (line 29-31, 116-118) and executed (line 32, 119-120) through the direct access mode (*rs.executeQuery* and *jpa.ExecuteQuery*). Then LMS (*rs* (ResultSet [25]) for JDBC and *ord* (typed object) for LINQ) are iterated row by row (line 33, 121). *mobilePhone* is updated (line 36-37, 126-127) if the student *id* (*rs.getInt* and *s.id*) is contained in the list *sId* (line 34-35, 123-124) through the indirect access mode. This update on the LMS is equivalent to the following Update expression

```
Update Student s
       Set s.mobilePhone=mobilePhone
       Where s.id=sId(idx)
```

and, therefore, *sId* and *modiblePhone* in Figure 1 and Figure 2 behave as runtime values for the two parameters of the equivalent Update expression. From this example it is also easily inferred the equivalency between the insert and delete protocols and the correspondent Insert and Delete

```
26  void updateStudentMobilePhone(List<Integer> sId,
27                   List<String> mobilePhone)
28                   throws SQLException {
29    String sql="Select * from dbo.Student " +
30               "where id between "+
31                  max(sId)+" and " + min(sId);
32    rs=st.executeQuery(sql);
33    while (rs.next()) {
34      int idx=sId.indexOf(rs.getInt(1));
35      if (idx!=-1) {
36          rs.updateString(4,mobilePhone.get(idx));
37          rs.updateRow();
38      }
39    }
40  }
```

Figure 1. Example based on JDBC.

```
113 void updateStudentMobilePhone(Collection<int> sId,
114                   Collection<string> mobilePhone)
115 {
116   string sql="Select * from dbo.Student " +
117              "where id between "+
118                    sId.Max()+" and "+sId.Min();
119   IEnumerable<Student> student =
120                   jpa.ExecuteQuery<Student>(sql);
121   foreach (Student s in student)
122   {
123     int idx = sId.IndexOf(s.id);
124     if (idx != -1)
125     {
126        s.mobilePhone = mobilePhone.ElementAt(idx);
127        jpa.SubmitChanges();
128     }
129   }
130 }
```

Figure 2. Example based on LINQ.

expressions. These two simple examples have shown the usage of the two access modes that are provided by current tools and also the usage of runtime values. Additionally, the examples also show that JDBC and LINQ, akin to the remaining tools, are not driven by access control policies. Their access modes allow programmers to write any CRUD expression (using the direct access mode) and also allow the use of any protocol on LMS. These latter two issues have been addressed in [13].

IV. ARCHITECTURE PRESENTATION

In this section we present an access control technique which enforces policies at the business tier level which is able to statically control the provenance of runtime values that are used on actions issued against databases. The technique supervises the runtime values that are used on both access modes of current tools that are used for developing business tiers. Nevertheless, access control policies can only be effectively enforced if other complementary aspects are also considered. Among them the authorized CRUD expressions and the actions on LMS are emphasized. Those aspects are not addressed in this paper because they were already addressed in [13]. From [13], a platform has been designed and developed. The platform will be used and modified to present the proof of concept. This section is organized as follows: the sub-section *A* presents the proposed technique; sub-section *B* briefly presents the used platform;

sub-section *C* presents the proof of concept and, finally, subsection *D* presents a use case.

A. Proposed Technique

We start by introducing the concept of Business Access Point (BAP). A BAP is an entity responsible for managing the runtime values of the two access modes in accordance with the established access control policies. Each access mode type has its own particular characteristics. As such, their conceptual architecture is presented separately.

Direct Access Mode

The direct access mode allows the execution of CRUD expressions based on the native SQL language. In a general context, each CRUD expression comprises a hard coded part and eventually one or more parameters of which the values are defined at runtime. The values for these parameters are not mandatory to be driven by any access control policy. It is up to the security expert to decide for each CRUD expression which parameters are driven by access control policies and which parameters are not driven by any access control policy. Thus, the direct access mode (DAM) is formalized by the next triplet:

$$DAM(\ RTV, RTV_{acp}, execute)$$

where *RTV* is a set of RunTime Values for parameters not driven by any access control policy, RTV_{acp} is a set of RunTime Values for the parameters driven by access control policies and, finally, *execute* is a method responsible for setting the runtime values for parameters and also for the execution of CRUD expressions. As initially announced, RTV_{acp} are statically enforced and, therefore, their implementation will have this in consideration. Eventually, each runtime value may be encapsulated as an interface that provides a service aimed at returning values driven by access control policies.

Indirect Access Mode

The indirect access mode provides four protocols for the interaction with the data contained by LMS that is returned by native Select expressions. A first solution has been proposed to provide the four protocols driven by access control policies [13]. Basically, it includes two aspects: 1) the availability of each protocol is individually configured and 2) each protocol that is made available provides methods to access only the attributes that are authorized by the established policies. This approach is not complete because it does not support parameters driven access control policies. Next follows the proposed approach to overcome this security gap. The indirect access mode (IAM) is formalized as follows:

$$IAM(readP, insertP, updateP, deleteP)$$

where *readP* is the read protocol, *insertP* is the insert protocol, *updateP* is the update protocol and, finally, *deleteP* is the delete protocol. Only the insert and the update protocols use runtime values. The read protocol does not modify the contents of LMS and the delete protocol is executed as an atomic operation on all attributes of the selected row. Thus, each individual method of the insert and update protocol that is used to modify each attribute of the returned relation (contained in LMS) needs to be configured to be or not to be driven by access control policies. They are formalized as:

$$method(RTV) \text{ or } method(RTV_{acp})$$

where *method* is the method's name, *RTV* and RTV_{acp} have the meaning previously presented for the direct access mode. The only difference is that either *RTV* or RTV_{acp} represent a single runtime value. The indirect access mode is only available after a Select expression is executed through the direct access mode. The remaining CRUD expressions do not create LMS. This leads to the need of defining two facets for the BAP: one for the Select expressions (BAP_s) and another for the remaining expressions (BAP_{iud}). BAP_{iud} supports the direct access mode only and is formalized as follows:

$$BAP_{iud}(DAM)$$

BAP_s supports both modes and is formalized as follows:

$$BAP_s(DAM,IAM)$$

B. Used Platform

The proof of concept here presented leverages the work previously presented in [13]. The work has been used to design a new architecture known as DACA (Dynamic Access Control Architecture). Figure 3 presents a simplified block diagram of DACA. DACA is able to dynamically, at runtime, build and keep updated business logic of relational database applications in accordance with the established access control policies. It comprises 2 main components: a client side component for the application and business tiers and a server side component where metadata of access control mechanisms are kept. The basic operation of DACA is as follows (see Figure 3): 1- application tier instantiates a Dynamic Access Control Component (DACC); 2- DACC, through the Business Manager, establishes a connection with the Policy Server; 3- The Policy Server transfers and keeps security metadata and CRUD expressions continuously updated on DACC, in accordance with the established access control policies; 4- DACC, through the Business Manager, dynamically builds and keeps business logic updated; 5- application tiers ask Business Manager to execute authorized CRUD expressions; 6- Business Manager delegates the execution of CRUD expressions on the implemented

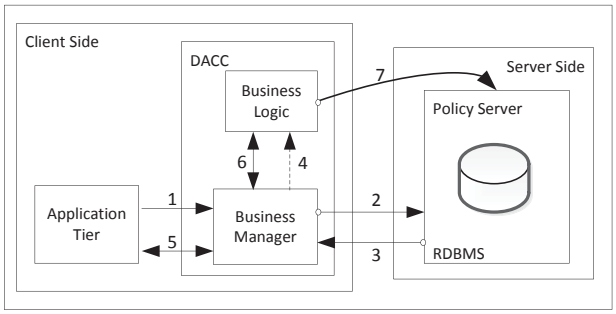


Figure 3. Simplified block diagram of DACA.

Business Logic; 7- CRUD expressions are executed (the RDBMS server may or may not be the same as the one responsible for the Policy Server).

C. Proof of Concept

The initial version of DACA was redesigned to address the issues of this research and it is hereafter known as RDACA (Redesigned-DACA). We have decided that the policy to be followed for RTV_{acp} requires that the values can only come from data previously retrieved by authorized Select expressions. To address this new security requirement the original DACA security access control mechanisms were redesigned. To give a complete view of the implemented solution, class diagrams of BAP will be provided.

The client-side of RDACA was implemented in Java and JDBC and, therefore, all examples are based on those tools. In the RDACA each RTV_{acp} is defined as an interface comprising a unique method which is responsible for retrieving the authorized value. The proposed approach, as it will be shown, allows a static validation for all RTV_{acp} at development time.

Figure 4, Figure 5 and Figure 6 present simplified class diagrams for the approach followed for the BAP to enforce access control policies. In a first step, one interface is defined for each individual RTV_{acp} as shown in Figure 4: IRTV_a, IRTV_b, …, IRTV_n. Each interface is related to a unique RTV_{acp} and it comprises also one unique method responsible for ensuring that the values are effectively authorized. *rA*, *rB*, …, *rN* are the defined methods and *DT_a*, *DT_b* and *DT_n* are the data types of the RTV_{acp} in the host programming language. The concrete implementation of each method depends on the adopted security strategy. In case of the RDACA, these methods retrieve data from data previously retrieved by authorized Select expressions and also managed by BAP_s.

Figure 5 presents a simplified class diagram for one BAP_s. The constructor of the base class, BAPs, receives a connection to the database and the CRUD *id* to be executed. Programmers do not write CRUD expressions anymore. They are only allowed to select, though the CRUD *id*, which CRUD expression is necessary. In case she is not authorized to use the requested CRUD expression, an exception will be raised. Other important aspects are the *IExecute* and the *ILMS* interfaces. *IExecute* is associated with the direct access mode and ILMS is associated with the indirect access mode. *IExecute* comprises one unique method (*execute)*. It accepts as arguments RTV and RTV_{acp} for the runtime values of the clause conditions for the Select expression to be executed. In this particular case, it accepts an RTV of type DT_a and an RTV_{acp} of type IRTV_b. Thus, to execute the requested Select expression it is necessary to be a holder of an BAP_s providing an IRTV_b. Regarding the *ILMS* interface, it comprises several interfaces being *IRead* and *IUpdate* presented with some detail. They are enough to convey a complete understanding about the followed approach. *IRead* implements the read protocol on LMS providing all the necessary methods to that end. Each method retrieves the value of one attribute of the returned relation. There are two types of methods: one type retrieves values that can only be used as RTV and the other type retrieves values that can be used as RTV_{acp}. Methods retrieving RTV are directly defined in the *IRead* interface, such as *rB* and *rC* as shown in Figure 5. Methods retrieving RTV_{acp} are defined by extending *IRead* with the interfaces that provide RTV_{acp}, see Figure 4. The shown IRead interface provides two methods for RTV (rB and rC) and one interface for one RTV_{acp} (IRTV_a). This distinction allows Business Manager (see Figure 3), by analyzing the schema, to be able to distinguish between RTV from RTV_{acp} and, therefore, to provide, during the automatic building process of Business Logics, different implementations for the two types of methods. Regarding the *IUpdate* interface it is associated with the update protocol. In this particular case it comprises two methods: a) *uA* updates the attribute *a* and it accepts an RTV_{acp} (IRTV_a); b) *uB* updates the attribute *b* and it accepts any RTV of type *DT_b*.

Figure 6 presents a simplified class diagram for one BAP_{iud}. The description for the base class and also for the *IExecute* interface is identical to the previous BAP_s. Regarding *ISet*, it comprises one unique method (*set*), which accepts as arguments RTV and RTV_{acp} for the runtime values of the column list of the Update expressions. In t his case it accepts two RTV_{acp} and one RTV. Thus, to be able to use

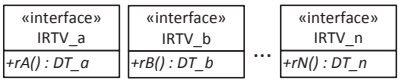


Figure 4. Set of RTV_{acp}.

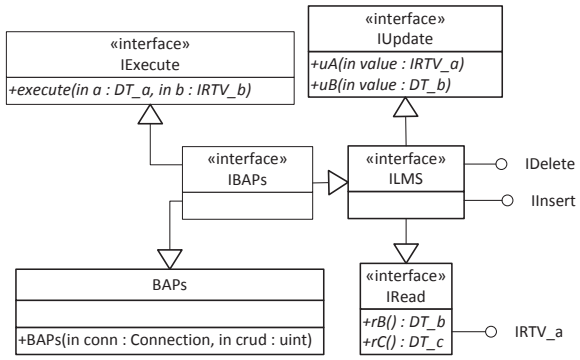


Figure 5. Simplified class diagram for a concrete BAP_s.

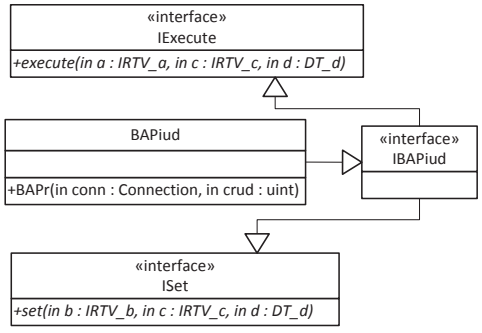


Figure 6. Simplified class diagram for a BAP_{iud}.

this BAP_{iud} it is required to be authorized to execute the required CRUD expressions and to hold three RTV_{acp} (IRTV_a, IRTV_B and IRTV_c) provided by one or more BAP_s.

D. Use Case

We are now prepared to present a real use case implemented with Java, JDBC and Microsoft Northwind database[1]. The use case is based on an actor responsible for managing orders coming from customers in the USA only. The actor is authorized to execute the two following CRUD expressions:

```
Select * from Customers
        where customerId=?          // (RTV)
              and Country='USA'
Select * from Orders
         Where CustomerId=?         // (RTVacp)
               and ShipCountry=?    // (RTV)
```

The first Select expression allows the access to information about the customers residing in the USA and the second Select expression allows the access to orders only from customers the user is authorized to know (RTV_{acp} – in this case residing in USA) and whose ship county is user defined (RTV). The BAP_s associated with the latter Select expression is updatable and one particularity is that the attribute *employeeId* requires an RTV_{acp} when using the indirect access mode.

To address this case, two BAP_s are needed, one for each CRUD expression. We have used the table names to identify each BAP_s, *Customers* and *Orders*. From the two Select expressions we see that, when using the direct access mode, the second one requires an RTV_{acp} for the first parameter - *CustomerId*. Figure 7 shows an example of how the two BAP_s (Customers and Orders) may be used. A new instance of Customers is created (line 30) and the CRUD expression is executed (line 31) to select data about the customer identified by the RTV of *customerId*. Then, the first and only row of the LMS (rs) is selected (line 32). Some attributes are read (line 33-34). Then an instance of Orders is created (line 35) and the CRUD is executed (line 36). The CRUD has two parameters, the first one is an RTV_{acp} and the second one is an RTV. The RTV_{acp} is for *customerId* and it is passed as the instance of Customers, which implements the required interface for the RTV_{acp}. The ship country is an RTV and, therefore, it is user defined. Some attributes are read (line 37-38) and the programmer tries to update *employeeId* but the NetBeans indicates an error because the correct data type cannot be an *integer* (line 39). To update *employeeId* through the indirect access mode the programmer needs an RTV_{acp} of the required type. To convey a deeper understanding some additional details are provided for the two BAP_s. Figure 8 shows the interface herein named as *ICustomerId* for the RTV_{acp} *customerId*. This interface is used not only to be implemented by BAP_s but also used whenever identifications of customers need to be used as RTV_{acp} for arguments of BAP methods, as shown in Figure 7. The implementation of

[1] http://www.microsoft.com/en-us/download/details.aspx?id=23654

```
28  private void go(Connection conn, int customerId,
29                  String shipCountry)throws Exception{
30      Customers c=new Customers(conn,crudCustomersId);
31      c.execute(customerId);
32      if (c.moveNext()) {
33          companyName=c.rCompanyName();
34          //... read more attributes
35          Orders o=new Orders(conn,crudOrdersId);
36          o.execute(c,shipCountry);
37          orderId=o.rOrderId();
38          // ... read more attributes
            o.uEmployeeId(5);
40          //... more code
```

Figure 7. Example to show the use of the two BAP_s: Customers and Orders.

```
    public interface ICustomerId {
        int rCustomerId() throws SQLException;
15  }
```

Figure 8. Interface for the RTV_{acp} to be used for the parameter CustomerId.

this interface should comprise some validation procedures to prevent its misuse. As previously explained, BusinessManager automatically generates the required source code for Business Logic. In this particular case, it creates the required source code for *rCustomerId()* in accordance with the established security requirements.

The IRead interface for *Customers* is presented in Figure 9. It provides a set of methods to read the attributes of the returned relation. *CustomerId* is the only attribute with the ability to be used as an RTV_{acp} and, therefore, the IRead interface extends the *ICustomerId* interface.

Figure 10 shows the *IExecute* interface for the BAP_s *Orders*. It comprises two arguments. The first argument is an RTV_{acp} for *customerId* and, therefore, it requires the correspondent interface (*ICustomerId*). The second argument is an RTV for the ship country. Figure 11 presents its implementation in which a main aspect is emphasized. The RTV_{acp} (*customerId*) is passed as an interface (line 28) and

```
    public interface IRead extends ICustomerId {
        String rCompanyName() throws SQLException;
        String rContactName() throws SQLException;
16      //... other attributes
```

Figure 9. IRead interface for Customers.

```
public interface IExecute {
    void execute(ICustomerId customerId,
16                 String shipCountry ) throws SQLException;
17  }
```

Figure 10. IExecute interface for Orders.

```
    @Override
28  public void execute(ICustomerId customerId,
29                      String shipCountry)
30                      throws SQLException {
31      ps=conn.prepareStatement(crud);
32      ps.setInt(1,customerId.rCustomerId());
33      ps.setString(2,shipCountry);
34      rs=ps.executeQuery();
35  }
```

Figure 11. *execute* method implementation of Orders.

the run time value (line 32) is obtained from the method specifically created for the effect and defined in the *ICustomerId* interface.

There is a runnable demo available at https://dl.dropboxusercontent.com/u/71192544/Work/Confers/SEKE/SEKE_2013/Example.7z .

V. CONCLUSION

This paper presents a technique aimed at enforcing access control policies statically at the level of the runtime values that are used on business tiers to interact with data stored on relational database management systems. The technique is applicable to commercial tools geared up to develop business tiers, such as JDBC, ODBC, Hibernate and LINQ, and supports their two most common access modes: the direct and the indirect access mode. Security experts are able to decide the policies to be used, which runtime values are driven by those policies and which are not. Runtime values driven by access control policies are managed at the business tier level to ensure the use of authorized values only. The presented proof of concept is based on an existent platform that has been redesigned to support a new security requirement. The new security requirement says that only previously retrieved values from the database are allowed to be used for the runtime values driven by access control policies. The implemented technique is based on interfaces comprising a unique method of which the implementation ensures the new security requirement. Beyond the presented proof of concept a runnable demo is also available.

It is expected that the outcome of this research will have impact on future proposals addressing access control on relational databases, mainly when policies are enforced at the level of client business tiers.

As future work, we intend to apply the techniques used in [26, 27] to design a thread-safe version of DACC. These techniques have proved to be not only simple to implement but above all conveying a significant performance improvement.

REFERENCES

[1] OASIS. "XACML - eXtensible Access Control Markup Language," Feb, 2012; http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml.

[2] L. Caires, J. A. Pérez, J. C. Seco et al., "Type-based access control in data-centric systems," in 20th European conference on Programming Languages and Systems: part of the joint European conferences on theory and practice of software, Saarbrucken, Germany, 2011, pp. 136-155.

[3] S. Chaudhuri, T. Dutta, and S. Sudarshan, "Fine Grained Authorization Through Predicated Grants," in IEEE 23rd ICDE - Int. Conf. on Data Engineering, Istanbul, Turkey, 2007, pp. 1174-1183.

[4] A. Chlipala, "Static checking of dynamically-varying security policies in database-backed applications," in 9th USENIX Conf. on Operating Systems Design and Implementation, Vancouver, BC, Canada, 2010, pp. 1-14.

[5] B. J. Corcoran, N. Swamy, and M. Hicks, "Cross-tier, Label-based Security Enforcement for Web Applications," in 35th SIGMOD Int. Conf. on Management of Data, Providence, Rhode Island, USA, 2009, pp. 269-282.

[6] J. Fischer, D. Marino, R. Majumdar et al., "Fine-Grained Access Control with Object-Sensitive Roles," in 23rd ECOOP - European Conference on Object-Oriented Programming, Italy, 2009, pp. 173-194.

[7] Q. Wang, T. Yu, N. Li et al., "On the correctness criteria of fine-grained access control in relational databases," in 33rd Int. Conf. on Very Large Data Bases, Vienna, Austria, 2007, pp. 555-566.

[8] W. Gary, G. Carl, S. Zhendong et al., "Static checking of dynamically generated queries in database applications," ACM Transansactions on Software Eng. Methodology, vol. 16, no. 4, pp. 14:01-14:27, 2007, doi: http://doi.acm.org/10.1145/1276933.1276935.

[9] B. Hicks, S. Rueda, D. King et al., "An architecture for enforcing end-to-end access control over web applications," in 15th ACM symposium on Access Control Models and Technologies, Pittsburgh, Pennsylvania, USA, 2010, pp. 163-172.

[10] S. Rizvi, A. Mendelzon, S. Sudarshan et al., "Extending Query Rewriting Techniques for Fine-grained Access Control," in ACM SIGMOD Int. Conf. on Management of Data, Paris, France, 2004, pp. 551-562.

[11] K. LeFevre, R. Agrawal, V. Ercegovac et al., "Limiting disclosure in hippocratic databases," in 30th Int. Conf. on Very Large Databases, Toronto, Canada, 2004, pp. 108-119.

[12] J. Yang, K. Yessenov, and A. Solar-Lezama, "A language for automatically enforcing privacy policies," SIGPLAN Not., vol. 47, no. 1, pp. 85-96, 2012, doi: 10.1145/2103621.2103669.

[13] Ó. M. Pereira, R. L. Aguiar, and M. Y. Santos, " ACADA - Access Control-driven Architecture with Dynamic Adaptation," in SEKE - 24th Intl. Conf. on Software Engineering and Knowledge Engineering, San Francisco, CA, USA, 2012, pp. 387-393.

[14] M. I. Y. d. Valle, A. Mana, J. Lopez et al., "Secure Content Distribution for Digital Libraries," in Proceedings of the 5th International Conference on Asian Digital Libraries: Digital Libraries: People, Knowledge, and Technology, 2002, pp. 483-494.

[15] J. Lopez, A. Mana, and M. I. Y. d. Valle, "XML-Based Distributed Access Control System," in Proceedings of the Third International Conference on E-Commerce and Web Technologies, 2002, pp. 203-213.

[16] M. Parsian, JDBC Recipes: A Problem-Solution Approach, NY, USA: Apress, 2005.

[17] B. Christian, and K. Gavin, Hibernate in Action: Manning Publications Co., 2004.

[18] C. Pablo, M. Sergey, and A. Atul, "ADO.NET entity framework: raising the level of abstraction in data programming," in ACM SIGMOD International Conference on Management of Data, Beijing,China, 2007, pp. 1070-1072.

[19] M. Erik, B. Brian, and B. Gavin, "LINQ: Reconciling Object, Relations and XML in the .NET framework," in ACM SIGMOD Intl Conf on Management of Data, Chicago,IL,USA, 2006, pp. 706-706.

[20] ISO. "ISO/IEC 9075-3:2003," [2011 May; http://www.iso.org/iso/catalogue_detail.htm?csnumber=34134.

[21] Microsoft. "Microsoft Open Database Connectivity," Jul, 2012; http://msdn.microsoft.com/en-us/library/ms710252(VS.85).aspx.

[22] G. Mead, and A. Boehm, ADO.NET 4 Database Programming with C# 2010, USA: Mike Murach & Associates, Inc., 2011.

[23] D. Kulkarni, L. Bolognese, M. Warren et al., "LINQ to SQL: .NET Language-Integrated Query for Relational Data," Microsoft.

[24] D. Yang, Java Persistence with JPA, pp. 390: Outskirts Press, 2010.

[25] Oracle. "ResultSet," Jul, 2012; http://docs.oracle.com/javase/6/docs/api/java/sql/ResultSet.html.

[26] Ó. M. Pereira, R. L. Aguiar, and M. Y. Santos, "A Concurrent Tuple Set Architecture for Call Level Interfaces," in ICIS - 12th IEEE/ACIS International Conference on Computer and Information Science, Niigata,Japan, 2013, pp. (accepted).

[27] O. M. Pereira, R. L. Aguiar, and M. Y. Santos, "Assessment of a Enhanced ResultSet Component for Accessing Relational Databases," in ICSTE-Int. Conf. on Software Technology and Engineering, Puerto Rico, 2010, pp. V1:194-201.

Exploring Architectural Design Decision Management Paradigms for Global Software Development

Meiru Che, Dewayne E. Perry
Department of Electrical & Computer Engineering
The University of Texas at Austin
Austin, Texas, USA
meiruche@utexas.edu, perry@mail.utexas.edu

***Abstract*—Global software development (GSD) is an increasing trend in the field of software engineering. It can be considered as coordinated activities of software development that are geographically and temporally distributed. The management of architectural knowledge, specifically, architectural design decisions (ADDs), becomes important in GSD due to the geographical, temporal, and cultural challenges in global environment. However, little work has be done on capturing, sharing, and evolving ADDs in a GSD context. Based on our previous work on ADD management in localized software development (LSD), we extend our study to explore ADD management paradigms for GSD in this paper. We propose three ADD management strategies for the distributed development environment, and according to global software project structures, we explore and analyze three typical ADD management paradigms that can be widely adopted in GSD. We aim to provide a fundamental framework on managing ADD documentation and evolution in GSD, and offer good insights into sharing and coordinating ADDs in a global setting.**

***Keywords*-architectural design decisions; global software development; architectural knowledge; documentation; evolution**

I. Introduction

Global software development (GSD) is an increasing focus in the field of software engineering. It can be considered as the coordinated activities of software development that are not localized and centralized but geographically and temporally distributed [14]. In GSD, software teams work together at geographically separated locations to accomplish software projects. Thus, global teams face challenges associated with the coordination of their work due to different locations, time zones, languages, and cultures. In order to cope with different challenges in the globalization of software development, communication, as well as coordination, a number of approaches have been proposed in different domains of GSD [1]. However, little attention has been paid to software architecting processes and software architectural knowledge management in the context of GSD. Similar to localized software projects, software architecting and architectural knowledge are important to support designing, developing, testing, and evolving software. We note, however, that in the global development of large complex systems, architecture plays an even more critical role in the structure of the project [13]. Therefore, managing and coordinating architectural knowledge such as architectural design decisions (ADDs) is a significant and also relatively new research problem in the context of GSD.

Perry and Wolf considered the selection of elements and their form to be ADDs, and the justification for these decisions to be found in the rationale [20]. It was not until 2004, with Boschs paper [5] at the European Workshop on Software Architecture, that software architecture has generally come to be considered as a set of ADDs. This specific focus on ADDs led to a broader focus on architectural knowledge [19]. Capturing and representing ADDs helps to organize architectural knowledge and reduce its evaporation, thus providing a better control on many fundamental architectural drift and erosion problems [20] in the software life cycle. In a globally distributed software environment, the documenting and sharing of ADDs can serve to support the complex collaboration and coordination needs of software projects. With the increasing trends of further globalization of software development, managing ADDs in GSD becomes a much more critical task than in a localized environment.

In our previous work on ADD management, we had an overall goal of providing a systematic approach that supports ADD documentation and evolution in a localized software development (LSD) context. Based on this, we intend to focus on involving ADD management in a global development environment in this paper. Since little work has been done on ADD documentation and evolution in GSD, we are going to discuss several ADD management strategies for multi-site software projects, and then explore the typical paradigms for ADD management in global software projects. We aim to provide a fundamental framework for managing ADDs in the context of GSD, and offer insights into architectural knowledge documentation and evolution for researchers and practitioners in the field of software architecture.

II. Localized ADD Management Approach

This section briefly introduces our previous work on ADD documentation and evolution in a localized software project context. We give an overview of the basic approach to

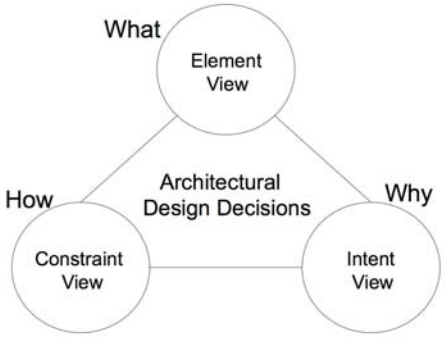


Figure 1. Triple View Model Framework

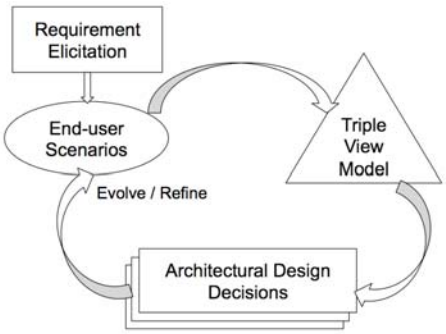


Figure 3. The Process of the Scenario-Based Method

managing ADDs, which provides a foundation for exploring ADD paradigms in GSD.

A. Triple View Model

To capture and document the ADD set in a software project, we propose the Triple View Model (TVM) to clarify the notion of ADDs and to cover key features in an architecting process [7].

The TVM is defined by three views: the element view, the constraint view, and the intent view. This is analogous to Perry/Wolf models elements, form, and rationale but with expanded content and specific representations [20]. Each view in the TVM is a subset of ADDs, and the three views together constitute an entire ADD set. Specifically, the three views mean three different aspects when creating an architecture, i.e., "what", "how", and "why", as shown in Fig. 1. The three aspects aim to cover design decisions on "what" elements should be selected in an architecture, "how" these elements combine and interact with each other, and "why" a certain decision is made. The detailed contents of each view in the TVM are illustrated in Fig. 2.

In the element view, the ADDs describe "what" elements should be selected in an architecting process. We define computation elements, data elements, and connector elements in this view. Computation elements represent processes, services, and interfaces in a software system. Data elements indicate data accessed by computation elements. Both computation elements and data elements are regarded as components in software architecture, and connector elements are (at minimum) communication channels (that is, mechanisms to capture interactions) between those components in the architecture.

In the constraint view, the ADDs are defined as behaviors, properties, and relationships. They describe constraints on system operations and are typically derived from requirement specifications. Specifically, behaviors illustrate what a system should do and what it should not do in general. It specifies prescriptions and proscriptions based on requirement specifications and other system drivers. Properties are defined as constraints on a single element in the element view, and relationships are constraints on interactions and configurations among different elements.

The ADDs in the intent view are composed of rationale and best-practices in an architecting process. Rationale, which includes alternatives, motivations, trade-offs, justifications and reasons, is generated when analyzing and justifying every decision that is made. Best-practices are styles and patterns we choose for system architecture and design. The architectural decisions in the intent view mainly exist as tacit knowledge [24].

B. Scenario-based ADD Documentation and Evolution

The TVM is the foundation of ADD documentation and evolution. Based on the TVM, we define the scenario-based ADD documentation and evolution method (SceMethod) [7].

In the SceMethod, we aim to obtain and specify the element view, constraint view, and intent view through end-user scenarios, which are represented by Message Sequence Charts (MSCs) [21]. Figure 3 illustrates the SceMethod process. At the beginning of the architectural design process, we obtain initial ADD results. Later on, as the requirements change, the ADDs are evolved and refined according to the new or updated requirements. By documenting all the possible ADDs and evolving these decisions with changing requirements, the SceMethod effectively makes ADDs explicit and reduces architectural knowledge evaporation.

Basically, we have the following four steps in the SceMethod to derive ADDs in a software project. For the sake of brevity, we will not discuss the detailed process of each step, but just give a brief introduction. We have the full description in [8].

1) Initialization: Before applying the TVM to end-user scenarios, the requirements of the software system are elicited, then we use MSCs to describe both the positive and negative scenarios. An MSC is composed of agent instances, interaction messages, and the timelines of the agents.

2) From MSC Syntax to Element View: We derive the element view directly from the syntax of MSCs. Specifically, each agent instance is taken as a computation element, and from the interaction messages between the source and target agent instances, we can extract data elements accessed by computation elements. Connector elements serve as communication channels between computation elements. Therefore, the element view is derived as follows:

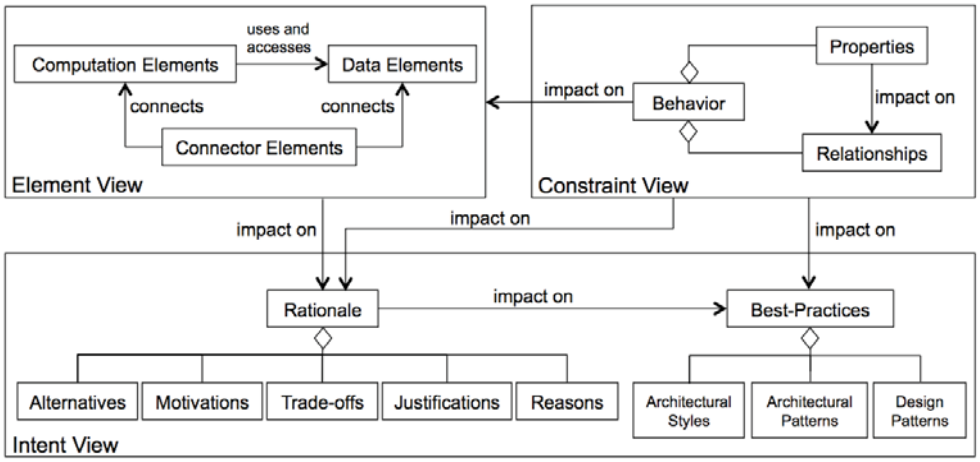


Figure 2. Triple View Model for Architectural Design Decisions

Computation Elements = {*Agent Instances*}
Data Elements = {*Interaction Messages*}
Connector Elements = {*Channels between Agents*}

3) From MSC Semantics to Constraint View: Based on the semantics of MSCs, we analyze behavior, properties, and relationships of the goal system to document ADDs in the constraint view. The ADDs on the behavior of the system are documented as:

Behavior = {*Prescriptions; Proscriptions*}
Prescriptions = {*Positive Scenarios*}
Proscriptions = {*Negative Scenarios; Exceptions*}

In addition, we use three factors to define properties, and we adopt simple path expressions to illustrate the interacted events in the MSCs to specify relationships:

Properties = {*Receive; Issue; Check*}
Relationships = {*Event Traces by Path Expressions*}

4) Intent View Documentation: Since decision making strategies are usually behind stakeholders' thoughts, the intent view cannot be derived directly from MSCs, which make it difficult to define a formal specification for documenting the intent view. The best way to make the intent explicit is to record decision making strategies as the architecting process moves forward. Specifically, answering each question that occurs to the stakeholders in the architecting and designing phase is helpful to constitute the ADDs in the intent view. Besides, architectural styles, architectural patterns and design patterns that we apply as best-practices should also be recorded as design decisions in the intent view.

Rationale = {*Answers to The Intent-Related Questions*}
Best-Practices = {*Architectural/Design Styles and Patterns*}

III. Multi-site ADD Management Strategy

As mentioned previously, the TVM and the SceMethod are the foundation of architectural knowledge management in GSD. However, managing ADDs in GSD becomes more difficult and complex than in LSD. On the one hand, the capturing and the documenting on ADDs are not just within a centralized environment, but considered for multi-site teams distributed geographically and temporally. On the other hand, the communication and the exchange of ADDs have a significant impact on the coordination of the distributed teams, and further, influence the subsequent analysis, design and implementation of global projects.

Table I
Multi-site ADD Management Strategies

Strategy	ADD Management Mechanism
Client-Server Strategy	Centralized ADD documentation on the headquarters site; Centralized ADD evolution on the headquarters site; Central repository is set up to store ADD knowledge information from the headquarters site; Central repository is accessed by all the sites.
Hybrid Strategy	Individual ADD documentation on each local site; Individual ADD evolution on each local site; Central repository is set up to store ADD knowledge information from each local site; Central repository is accessed by all the sites.
Incremental Strategy	Individual ADD documentation on site 1; Individual ADD evolution on site 1; ADD knowledge on site 1 is transferred to site 2; Individual ADD documentation on site 2; Individual ADD evolution on site 2; ADD knowledge on site 2 is transferred to site 3; . . ADD knowledge on site n-1 is transferred to site n; Individual ADD documentation on site n; Individual ADD evolution on site n.

In order to support ADD management in GSD projects, we propose three different strategies for managing the documentation and the evolution of ADDs in a distributed context, and discuss how distributed sites coordinate with each other to share and maintain consistent architectural knowledge. The three strategies for multi-site ADD management are client-server strategy, hybrid strategy, and incremental strategy respectively. Table I describes the detailed ADD management mechanism for each strategy.

In the *client-server strategy*, one site in the global software teams is considered as the headquarters, and it is responsible for the entire process of ADD documentation and evolution

in the global software project. Therefore, all the tasks on ADD documentation and evolution are conducted in the headquarters, which is similar to a localized software project context. In addition, a central repository is set up in the headquarters site to record and store the up-to-date ADDs, so that all the other sites can access the repository to share and reuse the ADDs through the global context. We term this strategy the client-server strategy because architectural knowledge resides in a central repository (as the server) and is accessed by all the distributed sites (as the clients).

In the *hybrid strategy*, every individual site manages architectural knowledge in a localized context, i.e., each site in the GSD project documents and evolves its own ADDs that are derived from its local architecting process. However, a central repository is also set up in one of the GSD sites for storing and sharing architectural knowledge throughout all the global teams. The repository is accessed by all the sites in the global project, and by this means, different sites can share and reuse ADD knowledge, or even reapply ADD knowledge in a different context. We term this strategy the hybrid strategy because it combines both the local ADD management and the global architectural knowledge sharing and reusing.

In the *incremental strategy*, the ADD management mechanism is analogous to the incremental development, i.e., on site 1, it manages the local ADD documentation and evolution process, and stores all the architectural knowledge in its local site. When site 1 derives all the ADD knowledge, it transfers the ADDs to site 2. Site 2 manages its local ADD documentation and evolution as well, and moreover, it combines the ADD information from site 1 into a larger ADD set. Similarly, when site 2 derives all the ADD knowledge, it transfers the ADDs to site 3, which follows the same way as site 1 and 2. In this strategy, each site captures ADDs in a certain context of the global project, and the final goal is that we are able to have a fully complete ADD set through the incremental documenting and evolving process.

IV. ADD Management Paradigms in GSD

In this section, we aim to explore the typical ADD management paradigms in GSD. First, we introduce three main software project structures adopted in a global development context, then we discuss the corresponding paradigm that is specific to each project structure.

A. Global Software Project Structures

Since global software projects very often have to deal with large and complex software systems, and development activities are performed by geographically different teams, the structure of a global software project plays a significant role in GSD. A good structure provides an effective way to organize the GSD project across multiple development sites, and in the meantime, it also offers a platform for managing the resources on both the development and the organizational activities.

A large number of possible ways to structure the GSD projects have been adopted. The main structures widely used are product-based structure, process-based structure, and release-based structure [3]. In addition, platform-based structure, competence-based structure, and open source structure are also often considered in GSD [3]. In this paper, the focus is on the first three typical structures to address ADD management in GSD.

In a *product-based structure*, a global system is decomposed into different components based on its requirement specification. Different components are then allocated as work items to different global teams. In a *process-based structure*, work items are allocated across different teams in accordance with the development phases of a software project. Specifically, we may allocate requirement, design, development, and test to different sites, and each site focuses on the tasks in the specific phase. As for the *release-based structure*, each site is responsible for a different release of the project, i.e., the first product release is developed on site 1, the second release is developed on site 2, and the third on site 3. In most cases, the releases are overlapped on different sites due to the timing requirement from customers.

B. ADD Management Paradigms

Given the foregoing discussion, we are going to explore and discuss three different paradigms for managing ADDs in GSD, which are specific to the three widely used software project structures.

1) Product-based Paradigm (Product-based Structure / Hybrid Strategy): For product-based structures in GSD, the system is decomposed into components and the components are allocated to distributed sites, thus each site conducts its own architecting process locally focusing on the functionality of the allocated components. During the architecting process in each local site, ADDs can be captured and documented by using the TVM and the SceMethod. We adopt the hybrid strategy to manage ADDs in the GSD projects with product-based structures. Figure 4 illustrates this paradigm in details.

As shown in Fig. 4, each site manages ADD documentation and ADD evolution locally according to the SceMethod and the TVM that we discussed in localized software projects. In addition, one of the global sites is selected as the headquarters and needs to set up a central repository for recording and storing the architectural decisions, which enables the geographically distributed sites to share ADDs in the global context. Each site can access the central repository, check in their local ADDs to the repository, and even read and reuse the ADDs from other sites when necessary. The headquarters site with the central repository coordinates the architectural knowledge in the repository and keep them consistent without conflicts. During the evolutionary process,

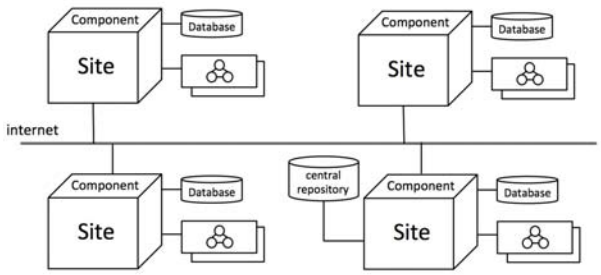


Figure 4. Product-based Paradigm in GSD

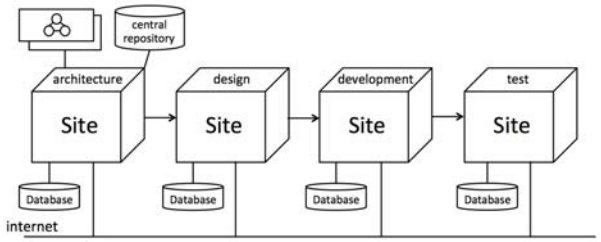


Figure 5. Process-based Paradigm in GSD

the evolved ADDs from each site are also transferred to the central repository. As we discussed for the hybrid strategy, the multiple sites in GSD manage architectural decisions not only within their local sites, but also with a central coordination to share and reuse architectural knowledge.

2) Process-based Paradigm (Process-based Structure / Client-Server Strategy): For the process-based structures in GSD, it is appropriate to use client-server strategy for managing ADDs. The reason is that the architecting process mainly occurs in the architecture phase, and all the other subsequent development phases, i.e., the design, development, and testing phases, are largely considered as the clients who access the ADDs that are derived in the architecture phase. Therefore, the client-server strategy provides us suitable support for GSD projects with process-based structures. We describe this paradigm in Fig. 5.

In Fig. 5, we note that the architecting process is conducted in the site with architecture phase, relying on our TVM and SceMethod to derive the entire ADD set. Moreover, a repository is set up in the same site to manage architectural knowledge documentation and evolution. This repository is also regarded as a central repository among the global teams, and all the other sites access the repository for sharing and reusing ADDs in the specific development phases. In some cases, the subsequent development phases, such as design phase, may also come up with new ADDs as the process proceeds. However, we do not deal with this kind of exceptions for now, but only explore the general paradigms that are normally used in GSD. More implications and exceptions will be addressed in our future work.

3) Release-based Paradigm (Release-based Structure / Incremental Strategy): The third paradigm that we are going to explore and analyze is for GSD projects with the release-based structures. In release-based structures, different product releases are allocated to different sites, so that each site handles all the development activities for the assigned release. It is obvious that in this paradigm each site derives its ADD set locally, and maintain ADD documentation and evolution in its local repository. Note that we do not need to create a central repository as the previous two paradigms, but only use each local repository for the architectural knowledge management.

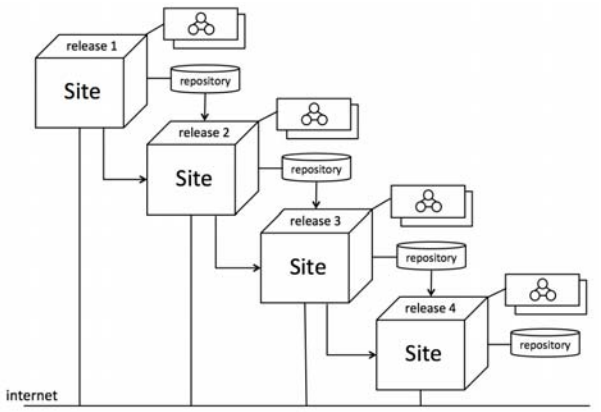


Figure 6. Release-based Paradigm in GSD

As illustrated in Fig. 6, each repository plays an important role in establishing a bridge to transfer architectural knowledge, which complies with the mechanism in our incremental strategy. Typically, the ADDs from the first site are transferred to the second site, so that the architectural knowledge from the first product release can be reused efficiently in the next release. In the release-based structure, the multiple releases contain similar or even the same functionalities and product features, which implies that the ADDs derived from different releases may have similarities as well. By adopting the incremental strategy in this paradigm, each repository can serve as a reused ADD pool, and the latter site is easy to combine, reuse, or even modify the ADDs derived from the former site.

V. Related Work

The key concepts of the traditional view on software architecture are components and connectors [4], [20]. Nowadays, software architecture has been seen as a set of ADDs [16], [23]. The architectural decisions in the software architecting process are increasingly focused by researchers and practitioners [12], [18], and ADDs are also considered to be a part of architectural knowledge [19].

Guidelines for documenting software architecture has been provided in [9], [15], however, those documentation approaches do not explicitly capture ADDs in the architecting process. Recently, many models and tools have been proposed for capturing, managing, and sharing ADDs, most of which are discussed and used within a localized software development context. Tyrees template [25] provides a simple

document describing key architectural decisions, which establishes a concrete direction for design and implementation, and also clarifies the rationale for different stakeholders. In [19], an ontology of ADDs and their relationships have been described. This ontology then can be used to construct architectural knowledge of a software system. ADDSS [6] is a web-based tool for documenting ADDs. It establishes the backward and forward traceability between requirements, decisions, and architectures. Other models and tools such as Archium [17] and AREL [22] are also proposed for managing ADDs

With the increasing attention paid to GSD, ADD management should be able to effectively applied in a GSD setting as well. However, little work has be done on ADD management in the GSD environment. A few of general architectural knowledge management practices for GSD have been proposed in [10], and the usefulness of these practices are evaluated in [11]. Furthermore, a literature review has been done [2] to explore architectural knowledge in a GSD context, and to synthesize architectural knowledge concepts, practices, tools and challenges that are important in GSD. In [26], six architectural viewpoints are defined to model GSD systems, which are based on a metamodel that has been derived after a thorough domain analysis of GSD literature.

Notably, architectural knowledge, specifically, ADDs, has not been widely discussed and supported in GSD, and the aforementioned approaches do not address in detail how to capture, share, and evolve ADDs in a global software project. In this paper, our goal is to provide a fundamental framework on managing ADDs in the GSD context.

VI. Conclusions and Future Work

With the increasing trend of GSD, the management of ADDs becomes more significant and critical due to the geographical, temporal, and cultural challenges innate to GSD. In this paper, we propose three different strategies for managing ADDs within multiple distributed development sites. Based on this, we explore three typical ADD management paradigms that can be widely used in GSD, and provide a high-level methodology on how to manage the documentation and the evolution of ADDs in the GSD context. In our future work, we plan to perform field studies to evaluate the ADD management paradigms in GSD projects. We also intend to investigate problems and implications for ADD management to provide insights into architectural knowledge management in GSD.

References

[1] First Workshop on Architecture in Global Software Engineering. Helsinki, Finland, August 2011. Http://www.cs.bilkent.edu.tr/AGSE-2011/.

[2] N. Ali, S. Beecham, and I. Mistrik. Architectural knowledge management in global software development: A review. In *ICGSE'10*, pages 347–352, 2010.

[3] A. Avritzer, D. Paulish, and Y. Cai. Coordination implications of software architecture in a global software development project. In *WICSA '08*, pages 107–116, 2008.

[4] L. Bass, P. Clements, and R. Kazman. *Software architecture in practice*. Addison-Wesley, Boston, MA, USA, 1998.

[5] J. Bosch. Software architecture: The next step. In *EWSA'04*, pages 194–199, 2004.

[6] R. Capilla, F. Nava, S. Pérez, and J. C. Dueñas. A web-based tool for managing architectural design decisions. *SIGSOFT Softw. Eng. Notes*, 31, September 2006.

[7] M. Che and D. E. Perry. Scenario-based architectural design decisions documentation and evolution. In *ECBS'11*, pages 216–225, 2011.

[8] M. Che and D. E. Perry. Managing architectural design decisions documentation and evolution. *International Journal Of Computers*, 6:137–148, 2012.

[9] P. Clements, D. Garlan, L. Bass, J. Stafford, R. Nord, J. Ivers, and R. Little. *Documenting Software Architectures: Views and Beyond*. Pearson Education, 2002.

[10] V. Clerc. Towards architectural knowledge management practices for global software development. In *SHARK'08*, pages 23–28, 2008.

[11] V. Clerc, P. Lago, and H. v. Vliet. The usefulness of architectural knowledge management practices in gsd. In *ICGSE'09*, pages 73–82, 2009.

[12] J. C. Dueñas and R. Capilla. The decision view of software architecture. In *EWSA'05*, pages 222–230, 2005.

[13] R. E. Grinter, J. D. Herbsleb, and D. E. Perry. The geography of coordination: dealing with distance in r&d work. In *GROUP '99*, pages 306–315, 1999.

[14] J. D. Herbsleb. Global software engineering: The future of socio-technical coordination. In *FOSE*, pages 188–198, 2007.

[15] C. Hofmeister, R. Nord, and D. Soni. *Applied software architecture*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.

[16] A. Jansen and J. Bosch. Software architecture as a set of architectural design decisions. In *WICSA*, pages 109–120, 2005.

[17] A. Jansen, J. van der Ven, P. Avgeriou, and D. K. Hammer. Tool support for architectural decisions. In *WICSA*, page 4, 2007.

[18] P. Kruchten, R. Capilla, and J. C. Dueñas. The decision view's role in software architecture practice. *IEEE Softw.*, 26:36–42, March 2009.

[19] P. Kruchten, P. Lago, and H. V. Vliet. Building up and reasoning about architectural knowledge. In *Quality of Software Architectures*, pages 43–58, 2006.

[20] D. E. Perry and A. L. Wolf. Foundations for the study of software architecture. *SIGSOFT Softw. Eng. Notes*, 17:40–52, October 1992.

[21] D. M. A. Reniers. Message sequence chart: Syntax and semantics. Technical report, Faculty of Mathematics and Computing, 1998.

[22] A. Tang, Y. Jin, and J. Han. A rationale-based architecture model for design traceability and reasoning. *J. Syst. Softw.*, 80:918–934, June 2007.

[23] R. N. Taylor, N. Medvidovic, and E. M. Dashofy. *Software Architecture: Foundations, Theory, and Practice*. Wiley, 2009.

[24] D. Tofan. Tacit architectural knowledge. In *ECSA'10*, pages 9–11, 2010.

[25] J. Tyree and A. Akerman. Architecture decisions: Demystifying architecture. *IEEE Softw.*, 22:19–27, March 2005.

[26] B. M. Yildiz and B. Tekinerdogan. Architectural viewpoints for global software development. In *ICGSE*, pages 9–16, 2011.

A Semantic-based Semi-automated Role Mapping Mechanism

Lijuan Diao
East China Normal University, China
Lijuan_diao@126.com

Wei She, I-Ling Yen
Univ. of Texas at Dallas
{wxs061000, ilyen}@utdallas.edu

Junzhong Gu
East China Normal University, China
jzgu@ica.stc.sh.cn

Abstract. Role based access control (RBAC) has been widely adopted in industrial and government. However RBAC is only suitable for closed enterprise environment. With modern Internet based application, collaboration and sharing among multiple organizations become essential and RBAC is no longer sufficient. Role mapping has been the solution to deal with multiple domains, where the roles in the hierarchy of one organization are mapped to the roles in the hierarchy of another organization. But role mapping can be a tedious task for the security officers if it is done fully manually. Yet, performing role mapping automatically incur security risks.

In this paper, we introduce a semi-automated role mapping process, where promising role mappings are generated automatically and recommended to the security officer(s). The security officers then approve or modify the recommended role mappings. We present a method for automatically generate role mappings based on the similarities of the roles in two role hierarchies. We use an example to illustrate our approach and show its feasibility.

Keywords: role-based access control, role mapping, concept extraction, role similarity.

I. INTRODUCTION

With the rapid proliferation of Internet technologies, sharing and collaboration have become the common paradigm in the new, globalized cyber world. For example, cloud provides large-scale sharing of hardware, software, and services and facilitates collaboration among them. Federated data warehousing offers information sharing. Applications, such as globalized supply chains, emergency response, distributed design, etc., require a wide range of sharing and collaboration. These sharing and collaborations raise security concerns. Proper access control and security defenses should be in place to assure that certain resources are only viewed, used, or modified by the entities who are intended to be allowed to view, use, or modify those resources.

There have been significant advances in access control technologies over the last two decades. In the early era, basic access control schemes, such as access control matrix and capability lists, have been used. From late 90s, role-based access control (RBAC) becomes the major paradigm, especially for large enterprises and organizations. Role hierarchy semantically reflects the structure of authorities and responsibilities of the personnel in an organization and, hence, the access rights can be defined accordingly. Also, compared to other access control models, RBAC can greatly cut down the cost for access control policy specifications.

Most of the traditional access control models, including RBAC, are developed on the basis of a closed system where the users, roles, activities, and the protected resources are well defined. These access control models cannot be directly applied to multi-domain systems, where cross domain accesses cannot always be properly defined in advance.

Attribute-based access control (ABAC) is another access control model that has been extensively investigated in recent years. ABAC are suitable for the open systems where requesters (users and processes) are rarely pre-known to the access control module. Attribute-based access control can be regarded as a natural extension of many conventional access control models (e.g. multi-level security model, role-based model, etc.), and is highly expressive. However, the cost of policy specification and decision making in ABAC greatly depends on the set of attributes selected for the involved domains (e.g. its size). Moreover, there is no well formed standard for attribute-based models yet, and, thus, are hard to be put to use in practice.

Cross domain role mapping is another potential solution to achieve access control in multi-domain systems. It extends the RBAC model and maps the roles of foreign collaborative domains into the local roles of each domain [1-5]. There are two approaches in role mapping. First, a trusted mediator can be used to integrate the role hierarchies of two interacting domains [1,3]. Since the access rights defined in the global role hierarchy may have conflicts with the access control requirements of the individual domains, these works focus on conflict resolution and optimization of role mappings. Generally, the mediator-based approaches have scalability problems and require a fully trusted mediator to perform the integration. In [3-5], mediator-free solutions are proposed to secure cross-domain interactions. They do not perform the integration of the hierarchies. They record all the inter-domain mappings that are activated to enable the access, and deny the access when there is a conflict.

In existing role mapping approaches [1-5], the association of roles from different domains is done solely by security officers. Such manual process can be tedious and lack of agility. In some applications, the role mapping may be required on demand. For example, in an emergency response scenario, new parties may come to the aid due to special needs in special circumstances, and they will need immediate information and resources sharing with all assisting teams. In order to assure proper access control, role mappings among the new and the existing organizations need to be done dynamically in real-time. Thus, a certain degree of automation is needed in order to achieve timely role mappings. However, security is a serious issue and automation is definitely not a full solution. Thus a rigorous

process should be defined such that automated analysis can be done to come up with "recommendations" of role mappings to cut the cost and time for role mapping. Security officers of the involved domains should verify and validate the recommendations to assure proper access control policy definition and enforcement.

The automated role mapping generation process is based on similarity of roles. It extracts key concepts for each role. Based on the key concepts, the basic similarity between two roles is computed. Conceptually, the position of the role in the role hierarchy, i.e., the parents and children of the role, also provides plenty of information about what the role is. Thus, we also consider the role hierarchy in the similarity metric for the roles. Based on an aggregated similarity measures between roles, role mappings are generated and recommended. Such recommendation of role mappings can be altered individually or system-wide to improve its flexibility. Also, the recommendation can help security officers and cut down the time and efforts for role mappings.

The rest of this paper is organized as follows. Section II provides a running example to illustrate the role mapping process. Section III discusses the semi-automated role mapping process and the corresponding system architecture. Section IV presents the role mapping recommendations for the example role hierarchies. Section V states the conclusion of the paper.

II. A RUNNING EXAMPLE

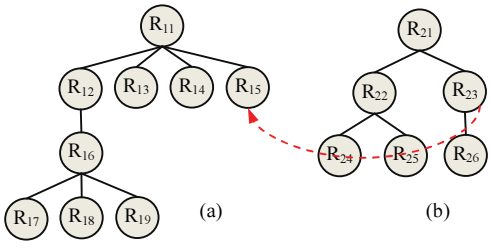


Figure 1. The role hierarchies of two health care systems.

Health care systems contain numerous private data and are frequently considered in security research. We take example role hierarchies of two health care systems as the running examples (Figure 1). In Figure 1 (a), the role hierarchy of a hospital (only showing a partial set of roles) and the names and descriptions of the roles are given in Table 1. Figure 1 (b) shows the role hierarchy of a long-term care nursing home (LCNH). The roles, names and descriptions of the roles in the LCNH are given in Table 2.

Role	Role Name	Part of role description
R_{11}	Hospital President	The president of a hospital. In charge of doctors, nurses, pharmacists, administration of the hospital
R_{12}	Surgical Dept Head	The head of the surgical department. In charge of surgeons who perform surgeries, nurses, surgical facilities
R_{13}	OB/Gyn Dept Head	The head of the obstetricians and gynecologists. In charge of the doctors, nurses, and facilities
R_{14}	Pharmacist Dept Head	Head of the pharmacists In charge of pharmacists, prescriptions, drugs
R_{15}	Head Nurse	Head of the nurses In charge of nurses, nurse hospitalized patients, patient care, assist doctors
R_{16}	Surgeon	Doctor who performs surgical operations
R_{17}	Medical Intern	Medical_intern is an advanced student or graduate in medicine gaining supervised practical experience (`houseman' is a British term)
R_{18}	Surgical Nurse	Nurse, specialized to take care of patients after surgery, knowledgeable in cleaning wound, removing surgical threads, IV and other injection, simple medical checks, such as blood pressure, body temperature.
R_{19}	Medical Technicians	Operate special medical devices, perform medical tests, and interpret the results. Examples include radiology tech, cardiology tech

Table1. The roles and their descriptions in the hospital case.

Role	Role Name	Part of role description
R_{21}	LCNH Director	The director of the long-term care nursing home. In charge of nurses and facilities in the nursing home, administrations, correspondence with medical director for health and care advices.
R_{22}	Medical Director	The doctor provides long-term care advices, is involved at all levels of care and supervision for the individual patients in the LCNH.
R_{23}	Nursing Director	Head of the nurses. Manage nurses for all long-term care patients.
R_{24}	Visiting Doctor	The doctor that provides regular visits to the LCNH, assessing health conditions of patients.
R_{25}	Physical Therapist	Develop physical therapy programs for individual patients, such as exercises, equipment assisted exercises, massage, evaluate patient progress in physical conditions.
R_{26}	Nurse	Take care of patients, perform needed care, injections, simple medical checks, such as blood pressure, body temperature.

Table2. The roles and their descriptions in the NCNH case.

Our goal is to perform semi-automated role mappings for the two role hierarchies. More specifically, we plan to recommend role mappings based on similarities between pairs of roles. For example, role R_{23}, which is a nursing director role in NCNH can very likely be mapped to R_{15}, the nursing director in the hospital role hierarchy.

In RBAC, permissions are assigned to roles to grant their accesses to various resources. There may be a large number of permissions in a real system. In Table 3, we list a few example permissions for the hospital, including the permission IDs and the functional descriptions of the permissions.

Permission ID	Function description
P_1	Record the nursing care information of patients
P_2	View medicine information of patients
P_3	Need to prepare and dispense drugs
P_4	View diagnoses of patient
P_5	Responsibility for curing the patient in the ear
P_6	Check the operation information of the patient

Table 3. The example permissions list for the hospital.

The example permissions given in Table 3 are assigned to the roles in the hospital role hierarchy. R_{11} has permission $\{P_1\}$. Also, R_{14}, R_{15} and R_{18} have permissions $\{P_2\}$, $\{P_3, P_4, P_5, \}$, and $\{P_6\}$, respectively.

III. SEMI-AUTOMATED ROLE MAPPING PROCESS

A. The Role Mapping Manager

We consider having a role mapping manager (RMM) in each domain to manage role mapping related tasks. RMM includes a role mapping recommender (RMR), a role mapping approver (RMA), a role mapping management unit (RMMU), a role mapping approval interface (RMAI), and a role mapping activation manager (RMAM). The architecture of RMM is shown in Figure 2.

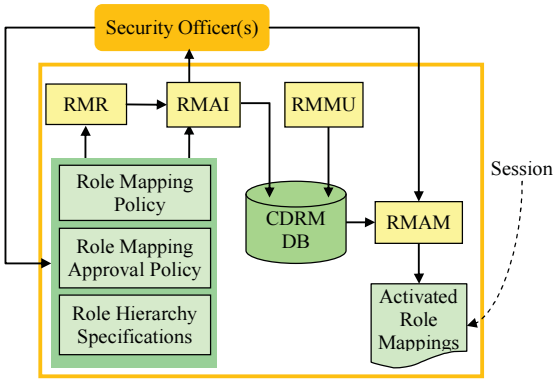


Figure 2. Architecture of the role mapping manager.

RMR. In a collaborative task, the roles of domain A that need to access some resources in domain B need to be mapped to domain B. To ease the task, the role mapping recommender (RMR) automatically generates potential cross-domain mappings for the involved roles and prepares them for approval. Since role mappings involve multiple domains, the RMRs of the involved domains need to work together to generate the role mappings. In Section IV, we discuss the techniques for generating role mapping recommendations.

RMAI. To assure security, each role mapping generated by RMR, say (r_x^A, r_y^B), which maps a role in a domain A to a role in domain B, needs to be manually approved by the SOs of domain B. Some mappings may only require the approval by one SO and more critical mappings may require the approval by additional SOs. The approval policy specifies such approval requirements. Once a mapping (r_x^A, r_y^B) is approved, it is sent to the RMMU and stored in the cross-domain role mapping database (CDRMDB) with an approval signature.

RMMU. RMMU manages the cross-domain role mappings in its database. The role mappings approved manually are stored in the regular database while the role mappings approved automatically are stored in a tagged database. Other entities in the system can retrieve the role mappings through RMMU. RMMU also provides tools and interfaces for the security officers to define role mappings manually and to modify and/or remove cross-domain role mappings in the database.

RMAM. The cross-domain role mappings stored in the regular database of the RMMU are deactivated by default. When a specific collaborative task (e.g., the execution of a workflow, etc.) occurs, RMAM activates a set of cross-domain role mappings and resolve the potential conflicts among them to facilitate the execution and proper data accesses of the task.

B. The Role Mapping Process

With the RMM architecture, the semi-automated role mapping process for generating cross-domain role mappings from domain A to domain B include:

1) The RMR in domain A identifies the roles in domain A that may need to access some data and other resources in domain B in potential collaborative tasks. It then sends a role mapping request with all the identified roles to the RMR in domain B.
2) Upon receiving the role mapping request, for each role r_i^A in the request, the RMR in domain B searches its role hierarchy for a similar role to it, say r_j^B. In this step, RMR may perform semantic similarity analysis between r_i^A and r_j^B and trust analysis of domain A and the involved entities in domain A.
3) The RMR in domain B, after generated the list of recommended cross-domain role mappings, passes them to the RMAI for approval. RMAI notify the security officers regarding the approval request. The SO (or SOs) of domain B goes through the role mappings using RMAI's GUI tools to view these mappings and makes the approval decisions.
4) The list of approved mappings is returned to domain A.

C. Automated role mapping Analysis

We use semantic technologies to find similar roles in different domains and use the information to recommend potential role mappings between two domains.

We use the information in the role hierarchy, role description, and role responsibility as input to perform concept extraction, similarity derivation, etc.

1) Concept Extraction

We build the concept set of each role from the role description, role responsibility, and property in the OWL based role model [6]. Various concept extraction methods have been introduced in the literature [7]. Most of these methods are statistical based and require a significant amount of input data in order to extract special features such as paragraph features, thematic word features, uppercase word features, etc.

We develop a simple matrix-based analysis method for concept extraction from various descriptions related to each role in the OWL-based role model. The detailed approach can be found in [8]. We apply the approach to derive the concept sets for the roles in the hospital and the NCNH role hierarchies, and parts of the results are given in Tables 4 and 5.

Role	Concept
R_{11}	Hospital President, administrator, doctors, nurses
R_{12}	Surgical Dept Head, surgeons, doctor, director
R_{15}	Head nurse, nurse, patient care
…	……

Table 4. The concept list in the hospital case.

Role	Concept
R_{21}	Long-term care nursing home, director, administrator
R_{22}	Medical Director, doctor
R_{23}	Nursing Director, nurse, patient assistance, elderly care
…	…

Table 5. The concept list in the NCNH case.

2) Similarity between Roles

We use two independent metrics to define the similarities between roles. The first metric defines the similarity between the concept sets of the roles. Concept is a group of semantically equivalent or very similar words. The concept of the roles provides the general level of agreement in the use of words for describing the roles and detects equivalent words that are likely to have been used to refer to similar roles in different domains. Similarities in the concept of two roles can be used as a basic similarity assessment approach to detect equivalent or synonymous roles, which may be an inconclusive form of similarity assessment because roles in two domains may be very different in terms of their role names, descriptions and responsibilities. Note that the role hierarchies also carry important information which can be used to help with similarity assessment. Thus, our second similarity metric attempts to incorporate semantics into the similarity measure by using distinguishing features as another indicator of how similar roles are. Due to space limit, the detailed similarity definitions are omitted here and readers can find the information in [8].

IV. Role Mapping Recommendation for the Example Role Hierarchies

We derive the similarities between the roles from the NCNH role hierarchy to the hospital role hierarchy. For each role, the pair with the maximal similarity is chosen to be the recommended role mapping. The result mapping recommendations are illustrated in Figure 3.

Similarity between roles may not be the only factor to be considered for role mapping. Security officers can specify various mapping policies to guild the automated mapping recommendation process. For example, consider mapping roles from domain A to domain B. The trust level of each role from domain A can be matched against the criticality of the roles in the local domain B following the mapping policy. Also, specific permissions required for the collaborative tasks can be analyzed and specified in advance to guild the mapping with least privileges. Since the methods for evaluating the additional metrics that may be used in the role mapping recommendation process is not in the scope of this paper, we do not consider it here.

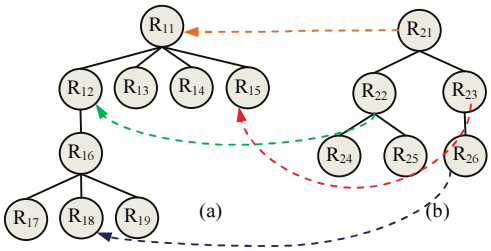


Figure 3. The role mapping recommendations for the example case.

V. Conclusions

In this paper, we introduce an automated role mapping recommendation process to help generate recommended role mappings between two domains to ease the role mapping task that is to be perform by the security officers manually. The approach in this paper focuses on using the semantic similarities between roles to determine the role mappings. We plan to investigate additional metrics that are frequently considered by security officers in determining role mappings and develop techniques to evaluate these metrics to guide the automated role mapping recommendation process.

VI. Acknowledgment

This research is supported in part by the NSF IUCRC on Net-Centric Systems and its industrial membership, by the Shanghai Scientific Development Foundation with Grant No. 11530700300, and by the China Scholarship Council. We thank our sponsors for their support.

VII. References

[1] B. Shafiq, J.B.D. Joshi, E. Bertino, A. Ghafoor, "Secure interoperation in a multidomain environment employing RBAC policies," IEEE TKDE, vol. 17, no. 11, pp. 1557-1577, 2005.

[2] P.A. Bonatti, M.L. Sapino, V.S. Subrahmanian, "Merging heterogeneous security orderings," European Symposium on Research in Computer Security, pp. 183-197, 1996.

[3] M. Shehab, E. Bertino, A. Ghafoor, "Secure collaboration in mediator-free environments," ACM CCS, pp. 58-67, 2005.

[4] S. Dawson, S. Qian, P. Samarati, "Providing security and interoperation of heterogeneous systems," Distributed and Parallel Databases, vol. 8, pp. 119-145, 2000.

[5] W. She, I-L. Yen, F.B. Bastani, B. Tran, B. Thuraisingham, "Role-based integrated access control and data provenance for SOA based net-centric systems," IEEE SOSE, pp. 225-234, 2011.

[6] J. Kupiec, J. Pedersen, F. Chen, "A trainable document summarizer," ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 68-73 , 1995.

[7] Tadashi Nomoto and Yuji Matsumoto, "A new approach to unsupervised text summarization," ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 26-34, 2001.

[8] L. Diao, W. She, I-L Yen, J. Gu, "A semantic-based Semi-automated role mapping mechanism," Tech. Rep. UTDCS-10-13.

Detecting Portability Issues in Model-Driven BPEL Mappings

Jörg Lenhard and Guido Wirtz

Distributed Systems Group, University of Bamberg
An der Weberei 5, 96047 Bamberg, Germany
E-mail: {joerg.lenhard | guido.wirtz}@uni-bamberg.de

Abstract

Service orchestration languages, like the Web Services Business Process Execution Language (BPEL), have been frequently used to provide an implementation platform for model-driven development approaches. As avoidance of vendor lock-in and portability of process definitions are central aims of BPEL, most approaches claim to support a large set of different runtime environments. But, even though today various runtimes for BPEL are available, every runtime implements a different language subset, thus hampering portability. Our idea is to improve this situation by using techniques, the Web Services Interoperability Organization (WS-I) has used to improve services interoperability. We describe a portability profile for BPEL that can detect portability issues in process definitions. Using this profile, we evaluate the portability of BPEL mappings used in several model-driven development approaches.

Keywords: SOA, BPEL, portability, profile, mapping

1. Introduction

Software portability is the ability to move software from one runtime platform to another without having to rewrite it fully or in part. It is a central characteristic of software quality [5]. Next to interoperability, it is also one of the core aims of service-oriented processes [6]. Being an OASIS standard that describes an open and platform-independent XML language for programming executable processes based on services, BPEL [8] is a main driver of services portability. For this reason, it is used as execution language in various model-driven mappings, such as [10, 12]. In these mappings, higher level and generally more abstract process or domain models, such as business process models, are transformed into executable BPEL code in the form of one or more process definitions. Through their property of being portable, BPEL process definitions enable these approaches to work on many different systems.

Shared standards are the basis for interoperability and portability of applications that run on heterogeneous platforms. However, these characteristics are difficult to achieve. Various case studies [7, 13] show that interoperability of heterogeneous systems still is limited. In the Web Services ecosystem, the WS-I is established as the main driver of interoperability. Its working mode is to define profiles which are standard documents that describe restrictions and assertions on existing standards, such as the *basic profile 2.0* [14]. Such assertions delimit the expressiveness of a standard or clarify the interpretation of it with the aim of making implementations more likely to interoperate. Although this approach does not guarantee interoperability, it relieves the problem and is accepted in practice.

Whereas the WS-I profiles deal with enforcing interoperability, portability has been neglected so far. The idea we present in this paper is to apply the concept of profiles from the area of *services interoperability* to the area of *services portability*. We address BPEL, as it is designed to build portable and executable programs. In previous work [4], we could show that current runtimes for BPEL (i.e., BPEL *engines*) implement different parts of the specification and the portability of process definitions among them is problematic. Here, we present a portability profile for BPEL that works similar to WS-I profiles [14] and addresses common portability issues. The issues are identified through an extensive benchmark of current engines. Using the profile, we can detect portability limitations in approaches that use a model-driven mapping to BPEL [9, 10, 12].

In the next section, we outline related work and in the following present the *BPEL Portability Profile*, focusing on the test assertions defined by it. Thereafter we discuss issues in model-driven approaches using BPEL. Finally, we summarize the paper and present areas of future work.

2. Related Work

Related work separates in approaches that try to mitigate portability issues in BPEL and work on BPEL mappings.

[1] also identifies the problem of portability among different BPEL engines which is ascribed to the informality of the specification and resulting ambiguities. The authors address the problem by defining the language B*lite* which en-

hances BPEL with a formal definition and refines the behavior of problematic constructs. B*lite* programs can be compiled to executable BPEL code for a specific runtime [1]. Such an approach can preempt portability errors, but requires the usage and understanding of another language ontop of BPEL, which in the case of a formal notation can be hard to learn. We do not define a new language, but provide assertions based on empirical data that can be used to detect portability issues in existing code.

Being an open international standard, BPEL has been used as target language in numerous model-driven mappings. A well-known one is part of the Business Process Model and Notation (BPMN) [9]. This is a standard developed by the OMG for modeling and visualizing several perspectives of business processes. It defines a notation for process models and a mapping of these process models to executable BPEL code. Since BPMN 2.0, this mapping is updated to the most recent revision of BPEL [9, pp. 445–474]. Before that, academic approaches tried to map BPMN 1.0 to BPEL 2.0 [10].

In service-oriented computing, a notable distinction of process models is made between orchestration and choreography models [11]. Choreography models describe a global view on a distributed process between multiple autonomous parties. In a model-driven setting, a local executable process for each of the different parties can automatically be derived from the global choreography. Such a local service-oriented process is called an orchestration. An example of a choreography to BPEL mapping is given in [12] based on the ebXML Business Process Specification Schema (ebXML BPSS).

3. The BPEL Portability Profile

The BPEL portability profile follows the scheme of the WS-I profiles [14]. It defines test assertions that can be seen as invariants of the standard specification [8]. Each test assertion defines a normative requirement of the profile that should be adhered to, if the goal of the profile (in this case portability of the code) is to be reached.

The assertions for the profile are based on data of an analysis of the BPEL conformance of a large set of BPEL engines [4]. The conformance assessment was performed using the tool *betsy*[1] [3]. The benchmark in this paper comprises seven engines: ActiveBPEL, bpel-g, Apache ODE, OpenESB BPEL Service Engine, Easy BPEL, Orchestra, and the engine of a leading middleware vendor whose name we cannot disclose for licensing reasons. The conformance test produces a data set indicating the support for every feature of the language specification by each engine. Based on this, it is possible to calculate the relative proportion to which each feature of the language is supported by today's runtimes. For each feature that is not fully supported by all engines and the usage of which might consequently result in a portability issue, a test assertion can be derived. These test assertions in turn enable the identification of portability issues in process definitions.

Test Assertions: Test assertions follow a predefined structure and contain several decisive elements. For the WS-I profiles and tools, not all test assertions are testable, because for several assertions the required information cannot be collected with the tools. The assertions of this profile base on the structure of process definitions only and as a consequence all assertions are testable. We defined and implemented a test assertion for every feature of the specification that was not supported by all engines under test. In total, this amounts to almost 70 assertions, each checking for the usage of specific BPEL elements or activities, their combination, or their configuration. Assertions that check for similar aspects, such as the usage of `toParts` and `fromParts`, are grouped together. The testing is based on XPath 2.0 expressions, similar to the mechanism used by WS-I profiles. Each assertion defines such an expression which selects all elements in the code that violate the criterion the assertion is checking. Based on the amount of engines that do not support a given feature, a test assertion can be classified according to a level of severity. The lower the amount of engines that support a feature, the more of a barrier to portability this feature will be.

The assertions are specified in the XML format for assertions defined by the WS-I. Crucial elements are the *target* and *predicate* which are both XPath 2.0 expressions. The target selects all elements in a process definition that violate the test assertion. This is necessary for being able to produce a list of all violations of an assertion when using the profile. The predicate determines whether on evaluation the assertion as a whole is counted as passed, which is the case if there are no elements found by the target. The *diagnostic* part specifies the severity of the test assertion.

Portability Levels: Based on the severity of the test assertions violated, it is possible to classify a process definition into different levels. This classification can be used to discriminate high-quality process definitions in terms of their portability from low-quality ones. We define the portability levels i) *portable*, ii) *widely portable*, iii) *partially portable*, iv) *limited portability*, and v) *nonportable*.

The severity, $Sev(ta)$, of an assertion ta depends on the degree of support of the feature, $S(ta)$, tested by the assertion. If all engines support a feature, it is fully *portable*. If at least 80 % of all engines support the feature, which can be considered an acceptable level of portability [2], it is classified as *widely portable*. If less than 80 %, but more than 50 % support the feature, it is classified as *partially*

[1] Betsy is a conformance testing tool for BPEL. For more information, see the project page: `https://github.com/uniba-dsg/betsy`. Betsy is also used in [4], but here we consider a larger number of engines.

portable. If less than 50 %, but more than one engine support the feature, which here amounts to at least 16 %, it is classified as being of *limited portability*. Finally, if only a single or no engine supports the feature, it is classified as *nonportable*.

$$Sev(ta) = \begin{cases} Portable, & \text{if } S(ta) = 100\% \\ Widely\ Portable, & \text{else if } S(ta) \geq 80\% \\ Partially\ Portable, & \text{else if } S(ta) \geq 50\% \\ Limited\ Portability, & \text{else if } S(ta) \geq 16\% \\ Nonportable, & \text{otherwise} \end{cases}$$

The classification, $Level(p)$, of a mapping or process definition p depends on the severity of the test assertions that it violates. The set V is the set of assertions violated by p and N_V is its size. Effectively, a process definition is assigned to the portability level of the most severe test assertion it violates. It is classified as portable only if no issues could be detected. This means, if all violations that are found concern test assertions of the severity *widely portable*, then the complete process definition is assigned to this level. If there is a single violation of the level *nonportable*, then the complete process definition is classified as *nonportable*.

$$Level(p) = \max_{i=1...N_V} (Sev(ta_i)) \ where \ ta_i \in V$$

The Bpp Tool: We have implemented the assertions and classification in the *bpp* tool[2]. The tool is written in Java and takes fragments of BPEL code, being complete process definitions or not, as input. The test assertions are encoded in it, and can be printed in the schema of a WS-I profile. At runtime, bpp checks each test assertion for the input code and records violations. Finally, the tool produces reports that conform to the WS-I report schema.

4. Evaluation of BPEL Mappings

In the following, we discuss issues in mappings from BPMN 1.0 [10], BPMN 2.0 [9, pp. 445–474], and ebXML BPSS [12] to BPEL.

4.1. BPMN 1.0

A notable contribution in the area of BPMN to BPEL mappings is [10] which focuses on revision 1.0 of BPMN, but claims to map to the up-to-date revision of BPEL. BPMN 1.0 on the other hand maps to an outdated version of BPEL. The mapping in [10] takes the form of BPMN constructs that are translated to fragments of BPEL code and two examples of complete mappings.

[2]For more information and a description on how to use the tool, see the project page: `https://github.com/uniba-dsg/bpp`. The BPEL fragments that underpin the discussion in section 4 are included as well.

Issues: The approach focuses on a control-flow oriented mapping to BPEL and omits several details. Whereas missing namespaces can be easily fixed, a missing declaration on how data handling works (i.e., by referencing variables or using the `parts` syntax) is problematic when it comes to the portability of the mapping. The `parts` syntax for dealing with data in BPEL is rarely supported, so the underspecification of data handling results in a portability issue here. A more severe problem is that the mapping uses elements unknown to BPEL 2.0 to direct the flow of control, as for example the `if-case` elements in the order fulfillment process mapping. These elements seem to be a variation of the BPEL 1.0 `switch-case` activity and their usage renders the mapping nonportable. Issues of minor severity in the mapping are the usage of `links` in the `flow` activity, as for example in the order fulfillment process mapping, which are only partially supported. A minority of engines does not support the `flow` activity, `onAlarm` timeout handlers and `onMessage` handlers when used in a `pick`.

Discussion: The severe issues in this mapping can be fixed by using the syntax defined by BPEL. By enhancing the mapping with a definition of how data items are handled in `invoke` activities, also this issue can be tackled. The usage of `links` is a crucial aspect of the mapping, so replacing it is likely to be a non-goal, even if it limits its portability.

4.2. BPMN 2.0

In revision 2.0 of BPMN, the mapping to BPEL is updated to the most recent revision of the specification [9, pp. 445–474]. The mapping describes elements of BPMN process diagrams and presents a fragment of BPEL code for each element. We analyzed each code fragment separately and, barring spelling errors in the BPMN specification, could detect portability issues in half of these code fragments. None of them are classified as nonportable, but 25 % of the issues are of limited portability.

Issues: One issue that reduces the overall mapping to limited portability are the fragments for the send and service tasks [9, pp. 448/449] and message end events [9, p. 457]. Similar to the mapping in [10], theses mappings use an `invoke` activity in BPEL, but do omit data flow and, therefore, fail to specify for instance an `inputVariable`. Depending on the Web Service operation that is invoked, this is legal in BPEL, but omiting variables is not supported by a majority of the engines. The data associations mapping [9, pp. 467/468] readdresses this issue, but uses the `from-` and `toParts` syntax of BPEL to assign variables instead of `assign` activities, which is also rarely supported. Areas of partial portability in the mapping are compensation intermediate events and compensation end events [9, pp. 457/458] which both use the `compensate` or the `compensateScope` activity. Error end events [9, pp.

Table 1. Summary of Portability Issues

Mapping	Classification	Major Issues	Minor Issues
BPMN 1.0 [10]	nonportable	non-BPEL elements, data handling	timeout handling, links
BPMN 2.0 [9]	limited portability	data handling, usage of parts-syntax	compensation, links, timeout handling
ebXML BPSS [12]	widely portable	–	timeout handling

457/458] are mapped to a `throw`. The interpretation of this activity, in case it is used to terminate a process instance, varies among engines. Another problematic part is the mapping of message handlers [9, p. 452] and message boundary events [9, pp. 458/459] which map to `onMessage` event handlers of a BPEL `scope`. While such event handlers are widely supported when used in `pick` activities, this is not the case when they are attached to a `scope`. Finally, the mapping of error boundary events [9, p. 459], multiple boundary events [9, pp. 460/461], and the inclusive decision pattern [9, p. 463] use `links`, partly in combination with `transitionConditions`, to direct the flow of control which is only of partial portability. Minor issues in the mapping are timeout handlers in event sub-processes [9, p. 452] and the exclusive event-based decision pattern [9, p. 462] which use `onAlarm` event handlers and timer intermediate events [9, p. 456] which rely on the `wait` activity. These timing-related activities are unsupported in a minority of engines. Also, the `forEach` activity, used in the multi-instances mapping [9, p. 455], is not ubiquitous.

Discussion: The main portability problems in the mapping result from data handling, either from not specifying it or from using a syntax that is only of limited portability. The more critical issues could be resolved by using `assign` activities instead of the `parts` syntax in BPEL.

4.3. ebXML BPSS

[12] defines a mapping from the ebXML Business Process Specification Schema to BPEL. The models translated are state-machine-based choreography definitions and BPEL fragments for translating states are presented.

Disussion: Only few issues could be detected in the mapping. No nonportable, limited, or partially portable elements were found, so the overall classification of the approach is *widely portable*. The only issues relate to timeout handling, implemented through `onAlarm` event handlers which are not supported by all, but by a majority of engines.

4.4. Summary

Table 1 summarizes the results from the previous sections. Portability issues could be detected in all mappings and two out of three produce BPEL code that will only be executable on a minority of today's BPEL engines. The most severe issues in the mappings relate to data handling which can be expressed in several different ways in BPEL, of which only a subset is widely supported.

5. Outlook and Future Work

In this paper, we proposed a mechanism for detecting portability issues in BPEL code, applied it to three model-driven development approaches that target the language, and made recommendations on how to fix detected issues.

Future work centers on two areas: Improving the portability profile by gathering more data on BPEL support and complementing the qualitative assessment here with a quantitative one based on formally defined portability metrics.

References

[1] L. Cesari, A. Lapadula, R. Pugliese, and F. Tiezzi. A tool for rapid development of ws-bpel applications. In *ACM SAC, Sierre, Switzerland*, March 22-26 2010.

[2] M. Glinz. A Risk-Based, Value-Oriented Approach to Quality Requirements. *IEEE Computer*, 25(8):34–41, 2008.

[3] S. Harrer and J. Lenhard. Betsy – A BPEL Engine Test System. Bamberger Beiträge zur WI und AI, no. 90, University of Bamberg, July 2012.

[4] S. Harrer, J. Lenhard, and G. Wirtz. BPEL Conformance in Open Source Engines. In *IEEE SOCA*, Taipei, Taiwan, December 17-19 2012. IEEE.

[5] ISO/IEC. *Systems and software engineering – System and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models*, 2011. 25010:2011.

[6] R. Khalaf, A. Keller, and F. Leymann. Business processes for Web Services: Principles and applications. *IBM Systems Journal*, 45(2):425–446, 2006.

[7] S. Kolb, J. Lenhard, and G. Wirtz. Bridging the Heterogeneity of Orchestrations - A Petri Net-based Integration of BPEL and Windows Workflow. In *IEEE SOCA*, Taipei, Taiwan, December 17-19 2012. IEEE.

[8] OASIS. *Web Services Business Process Execution Language*, April 2007. v2.0.

[9] OMG. *Business Process Model and Notation (BPMN) Version 2.0*, January 2011.

[10] C. Ouyang, M. Dumas, W. M. P. van der Aalst, A. H. M. ter Hofstede, and J. Mendling. From Business Process Models to Process-Oriented Software Systems. *ACM Transactions on Software Engineering and Methodology*, 19(2), 2009.

[11] C. Peltz. Web Services Orchestration and Choreography. *IEEE Computer*, 36(10):46–52, October 2003.

[12] A. Schönberger, C. Pflügler, and G. Wirtz. Translating Shared State Based ebXML BPSS models to WS-BPEL. *IJBIDM*, 5(4), 2010.

[13] A. Schönberger, J. Schwalb, and G. Wirtz. Interoperability and Functionality of WS-* Implementations. *International Journal of Web Services Research*, 9(3):1–22, 2012.

[14] WS-I. *Basic Profile Version 2.0*, November 2010.

Introducing Software Process Specification to Task Context

Ivens da S. Portugal, Toacy C. Oliveira
COPPE/UFRJ – Federal University of Rio de Janeiro (UFRJ)
Rio de Janeiro – RJ - Brazil
ivens@cos.ufrj.br, toacy@cos.ufrj.br

***Abstract*—A Software Development Process is used as a guideline to the development of a Software Product, from its conception to its delivery and maintenance. Depending on the complexity of the Software Process, a large number of artifacts must be handled by the Software Engineer. However, despite the sheer number of available artifacts, only a subset of such artifacts is used to complete a particular activity within the software process. As a result, typical Process-Centered Software Engineering Environments must restrict the access to unnecessary artifacts based on the current process context thus avoiding errors and information overflow. The Degree of Interest (DOI) function is a mechanism that is used to infer the importance of an element related to the ambient it is inserted. Currently, the Mylyn plugin uses the DOI function to infer the artifact's relevance based on how the user manipulates files in the Eclipse Environment but overlooking the Software Process in which such artifacts are inserted. This work presents an extension to the Mylyn's DOI function to incorporate software process' context information to discover the relevance of a given artifact.**

software development process, software process, software process execution, Degree of Interest, task context

I. INTRODUCTION

A Software Development Process (SDP) is a structure used by Software Engineers during the development of a software product. Basically, a SDP helps them to define what activities should be done, as well as the order, and provide an easy and simple way of communication between all stakeholders [1]..

Depending on the complexity of the SDP and the Software Product under development, the software process execution can contain a large number of interrelated activities and artifacts [4]. For example, the Rational Unified Process (RUP[1]) specifies more than a hundred activities and artifacts that can easily overwhelm the developers with unneeded information. Due to such great amount of activities and artifacts, a Software Engineer must search, within several artifacts, those of interest, to perform a particular activity. The task of searching those artifacts can be tiring, error-prone and time-consuming. Some effort will be put into the act of looking for artifacts rather than to the act of completing the activity being executed. Once the Software Engineer has the relevant artifacts on his workspace, he can perform the activity.

Besides, the execution of an activity can be interrupted by the execution of another activity with higher priority. It means that the Software Engineer should close the artifacts related to the first activity while performing another search to retrieve the artifacts related to the new interrupting activity. When that new activity is finished, the Software Engineer then closes the artifacts related to this activity and search the artifacts for the first activity. The process of changing activities demands attention and especially time. Software Engineer spends a considerable amount of time searching artifacts, which reduces the time he/she effectively spends working. This leads to a decrease in their productivity [10].

A solution for these two problems, searching for proper artifacts and switching activities, is the utilization of a Degree-of-Interest (DOI) function to create what is called a context for the activity. An activity context relates each activity with the most relevant artifacts required in its execution. The relationships are established by calculating a value of interest for each artifact used in the execution of a particular activity. The more selected and edited an artifact is, the more important it is for that activity. As a result, artifacts with high interest (more manipulated) are added to the related context. On the other hand, low interest artifacts can be omitted from Software Engineer's view. A current implementation of the DOI function can be found in Mylyn [9] where tasks are associated with Java files within the Eclipse Integrated Development Environment.

An important drawback to the current DOI function is its obliviousness to the SDP the activities should follow. As a result, establishing the initial context of an activity is difficult due to the lack of information on artifact manipulation when starting an activity. Moreover, without process knowledge, it is hard to use DOI for other types of artifacts than Java files since Java conforms to a precise structure from where some relations can be inferred (e.g. import and class extension relationships).

In this work we propose an extension to the Mylyn's DOI function and its associated environment to add SDP awareness. We do this by modifying the DOI function to take into account the activity-artifact relations that are present in most software processes specifications such as RUP and SCRUM. We have also adapted the Mylyn plugin to import a process specification. The final adaptation is called MylynSDP.

This work is organized as follows. Section 2 presents Related Work. In Section 3, we introduce the idea discussed in this paper as well as the MylynSDP's implementation. Section 4 draws some conclusion related to this work.

[1] http://www-01.ibm.com/software/awdtools/rup/ verified on May 10, 2013

II. RELATED WORK

A. Task Context

During the development of a Software product, a programmer may face an Integrated Development Environment (IDE) full of classes and methods that belong to the project. In order to perform a task, such as implementing a new functionality for the system under construction, he/she often must consult a set of classes or methods, so he can come up with ideas and strategies to the implementation. The problem is that this set of classes and methods relevant to the task being performed is too small compared to the set of classes and methods available for use. Due to this, the programmer may spend a great amount of time to gather the relevant classes and so perform the required task [10][7].

The problem of creating a task context, described above, has been under investigation by scientists of British Columbia University, in Canada, since 2005. These studies led to the construction of a tool for programmers to help them retrieve relevant classes to the task at hand, called Mylyn [9]. Mylyn is a plugin for the IDE Eclipse that associates a development task to the Java classes, its methods and variables. Mylyn has a DOI function, which can set an interest value for each class, method or variable based on the programmer's interaction with the code. The most used classes have higher interest, which means that should be in the context of the task being performed. After setting the interest value for some classes, Mylyn's DOI function can omit some classes from the programmer's view (those with lower interest value) and help him focus on the most important classes. Table I describes the types of programmer's interactions that are recognized.

TABLE I. MYLYN'S INTERACTION EVENT TYPES

Interaction Event Type	*Description*
Selection	Select an artifact via mouse or keyboard
Edition	Textual or graphical edits
Command	Operations like save, compile, build
Propagation	Interaction affects other classes, methods or variables
Prediction	Possible future interactions

Mylyn is aware of the fact that a programmer's task can be interrupted either by another task with a higher priority or because the programmer's need to close the IDE and continue the task later. In both situations, the context of the task would be lost if Mylyn did not persisted the context to disk. This action allows the programmer to interrupt a task's execution, either by stopping it or switching contexts, and consequently switching tasks, without the need of retrieving the activity's context again later, which increases his/her productivity

Unfortunately, Mylyn's DOI function and other features are aimed to programming activities only. This corresponds to just one part of the whole software development process. A good approach would be to expand these features and cover the whole software process, by dealing with all activities and all artifacts present in the software process.

[1] http://www.redmine.org verified on May 10, 2013

[2] http;//www.ibm.com/software/rational/products/rtc/ verified on May 10, 2013

B. Process-Centered Software Engineering Environment

A Process-Centered Software Engineering Environment (PSEE) is a software development environment that allows a Software Engineer to design and execute a software process. In other words, a PSEE supports software process enactment. A PSEE can provide ways of managing the Software Process by being aware of the duration of the activities, its end dates and what artifacts are being used and produced [2][8]. An extensive study of PSEE characterization has been made in [11].

The concept of associating Software Process' activities and artifacts, concerning their importance, in order to construct a context for the activity has been used in a project called WebAPSEE [3][5][6]. WebAPSEE is a PSEE aimed at providing automation and flexibility of Software Process management. To achieve this, WebAPSEE allows the Software Engineer to design and execute the Software Process. It also provides other features such as process reuse, artifact version control and business reports generation.

WebAPSEE maintains a repository of Artifacts that is consulted when constructing the context of an Activity. Unfortunately, task contexts in WebAPSEE are manually defined. If any new artifacts are created in the specification, task contexts need to be manually updated. The same happens when an activity is added to the software process specification.

Another relevant PSEE is called TABA Station [12]. TABA is a meta-environment able to generate software development environments based on a software process specification. By doing this, a Software Engineer can manage the execution of the Software Process within an environment suitable for his/her needs. When instantiating an environment, TABA creates a repository where the software process' artifacts are stored. Unfortunately, as in WebAPSEE, the Software Engineer must associate artifacts and activities to build the contexts manually although this context is already described in the software process specification. An important drawback of this PSEE is related to process flexibilization. Whenever a change is made in the software process specification, the environment generated must be recreated.

Some other PSEEs are RedMine[1] and Rational Team Concert (RTC)[2].

III. MYLYNSDP

A. Overview

Depending on the complexity of the Software Process, it can contain a large number of activities and artifacts. In addition to this, as discussed in previous sections, a Software Engineer can possibly switch from one activity to another, which leads to the time-consuming job of reconstructing the context of the activity. The objective of this study is to help Software Engineers to be more productive by reducing the time spent with the search for artifacts to construct the context of an activity. In order to achieve that, Mylyn project and features were extended, which led to the implementation of an application named MylynSDP.

In this paper, and throughout the code of MylynSDP, each job that should be executed by the Software Engineer is called

an Activity. In addition, the documents used in the process are named Artifacts. However, during the execution of the Software Process, every activity is referred to as a Task. Artifacts remain being called Artifacts. Also, formally, the set of Artifacts a Software Engineer needs to complete a Task is called a Task Context.

The application described in this paper is aimed to work on Eclipse IDE[1] (as Mylyn is). After importing the Software Process specification and creating the desired artifacts and tasks, the main interface will look like Figure 1. Tasks are shown in the right panel (Figure 1-a), artifacts are shown in the left panel (Figure 1-b) and the central panel is the working area (Figure 1-c). The main idea is to omit irrelevant artifacts when a Software Engineer activates a task, as shown in the second window of Figure 1.

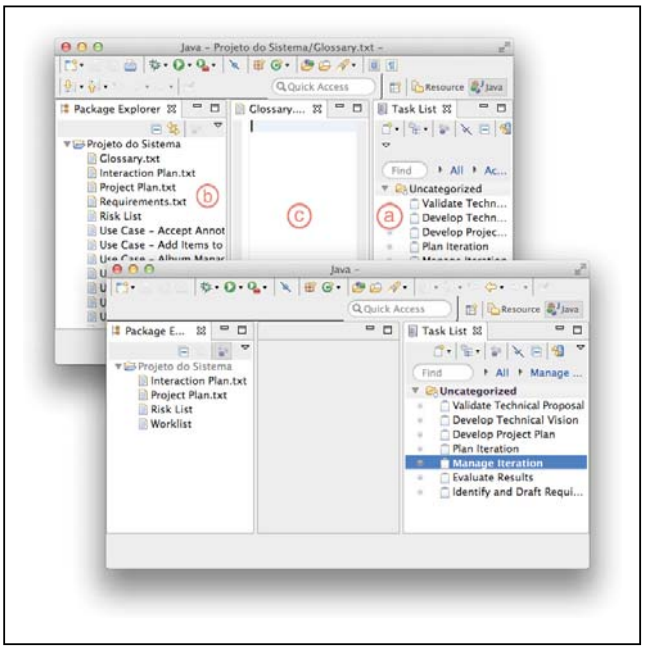


Figure 1. MylynSDP's Main Interface. When a task is selected, only a those artifacts related to that task are shown

MylynSDP, described in this paper, has 4 main parts: Software Process Specification Import Mechanism; Saving Mechanism; Restore Mechanism; and DOI function. Each of them is explained in the following sections.

B. *Software Process Specification Import Mechanism*

Initially, MylynSDP imports an XML file with the written specification of the Software Process. The Import Mechanism perform three actions: (1) it process the XML file to gather information about the activities and artifacts, (2) it copies Process specification to the workspace for later use and (3) it transform the XML file being imported to a format that Mylyn understands, so it can continue the import and can create the space needed for the management of tasks and artifacts.

During the execution of a Software Process, an activity can be performed more than once. To perform an activity, it should be instantiated into a task. Due to this, it is said that an activity is a type of a task. Artifacts present in the Software Process specification do not fully represent the set of artifacts that is actually being used in the execution. For example, while a system can have 30 different real Use Case documents, they will be represented as one single artifact element in the software process specification. Because of this, it is said that the artifacts in the software process specification document are a type for the artifacts used in the execution of the Software Process. From now on, this paper will treat activities and artifacts presented in the Software Process specification as types for Tasks and Artifacts of the Software Process execution, respectively.

C. *Restore Mechanism*

From this moment, the Software Engineer can create activities as well as artifacts. There is a new Wizard for each of the actions. Artifacts are simple text files but created in a different manner, so that a type of artifact can be associated and persisted to disk by the Restore Mechanism.

In order to create a task, the Software Engineer follows the local task creation process in Mylyn, but when setting the parameters for the task, such as its name, there is a section to fill in the type of the task, i.e. the associated activity. The Restore Mechanism saves both name and type of the new task to disk. It is important to note that both artifacts and activities types are based on the Software Process specification imported earlier.

As the Software Process specification already defines which artifacts are associated with an activity, the context of a particular task (instance of an activity) can be initially inferred. This is done by consulting the Software Process specification already imported whenever a task is created. A similar action happens when creating a new artifact, so it can be put on the suitable already created task contexts.

D. *Saving Mechanism*

The original Mylyn's Saving Mechanism's objective is to persist to disk information about the interest value associated with each file, along with the actions performed by the Mylyn's user. As explained on section Section II - A, Mylyn maps five types of interactions: Selection, Edition, Command, Propagation and Prediction. A new type of interaction is being introduced: the Specification interaction. It was created to handle the case where an interest value should be increased for a particular artifact because it initially belongs to a context.

Modifications made to the Saving Mechanism are small and mainly related to the ability of saving a new type of interaction in the interaction history.

E. *DOI function*

Each interaction event with a particular artifact increases that artifact's interest related to the task being executed. In addition, the interest value will decrease when interaction events are happening to other artifacts.

Mylyn do not store all interaction events being performed by the Software Engineer. Instead, it saves the ordinal number of the interaction event performed at the creation of the artifact.

This is done so that Mylyn can save disk storage space. When the next interaction event happens, the comparison between the two numbers, (1) the ordinal number associated with the new event and (2) the ordinal number related to the interaction event performed on the creation of that artifact, will allow the DOI function to calculate the decay value of the interest value.

The DOI algorithm, shown Figure 2, that calculates the interest value associated with the artifacts of a task context is divided into three parts: (1) the event registration, (2) the encoded value calculation and (3) the decay value calculation. When an interaction event happens to a particular artifact, "`updateEventState()`" method is called. It just increments the number of selections, edits or any other interaction event type that was performed on that artifact. Later, when needed, Mylyn, and so MylynSDP, calls "`getValue()`" method to retrieve the interest value for an artifact in a task context.

The first step is to calculate the number of 'selection', 'edition' and 'command' interaction events registered so far times a constant associated with each interaction event type. The "`specificationBias`" variable is a zero-or-one value that indicates if that artifact initially belongs to the current task context based on the Software Process specification. If so, the value of 200 is added to the interest value, which will certainly put this artifact into the current task's context. The value 200 is arbitrary and was reached by testing the code and calibrating it. The "`getDecayValue()`" method is then called to calculate the decrease of the interest based on how many interaction events were applied since the creation of that artifact. Finally, "`getValue()`" method returns the interest value for a particular artifact in a task context.

```
public float getValue() {
    float value = getEncodedValue();
    value += predictedBias;
    value += propagatedBias;
    return value;
}

public float getEncodedValue() {
    float value = 0;
    value += selections * contextScaling.get(InteractionEvent.Kind.SELECTION);
    value += edits * contextScaling.get(InteractionEvent.Kind.EDIT);
    value += commands * contextScaling.get(InteractionEvent.Kind.COMMAND);

    value += specificationBias * 200;

    value -= getDecayValue();
    return value;
}

public float getDecayValue() {
    if (context != null) {
        return (context.getUserEventCount() - eventCountOnCreation)
                * contextScaling.getDecay();
    } else {
        return 0;
    }
}
```

Figure 2. Degree of Interest function's code.

IV. CONCLUSION

In this paper, a new way of interacting with a Software Process Execution's artifacts was described. By selecting a task to be performed, the Software Engineer is presented with the most relevant artifacts related to the task activated instead of all artifacts. This smaller set of artifacts is called the task context. As the Software Engineer executes the task, he handles some of the artifacts of the task context. The more an artifact is used, the more it is relevant to the task execution and the more it is likely to remain on that task context. In addition, the less an artifact is used, the less it is important to the execution, and the less it is likely to be on that task context.

The initial context of any task is defined based on the Software Process specification. The idea of filtering the visualization of the Software Process Execution's artifacts by grouping them into task context is based in Mylyn's approach to help programmers that deals with a large number of classes in a project. The idea described has two main advantages, both related to the search of artifacts. The first is that a Software Engineer no longer needs to spend much time to build a context to complete a task's execution. The second advantage is related to the task change, and consequently the context change. The Software Engineer does not need to reconstruct the task context after every task switch. The framework helps him by persisting the task context to disk and retrieving it when needed.

All the ideas discussed in this paper were implemented as an Eclipse's Mylyn extension, which was then named MylynSDP. A validation project is about to be conducted to verify and validate all the concepts presented in this paper.

REFERENCES

[1] A. Fuggetta, "Software process: a roadmap," in Proceedings of the ICSE Conference on the Future of Software, pp. 25-34, 2000.

[2] A. Fuggetta and C. Ghezzi, "State of the art and open issues in process-centered software engineering environments," in Journal of Systems and Software, vol. 26, 1994, pp.53-60.

[3] A. Lima, A. Costa, B. França, C. A. L. Reis and R. Q. Reis, "Gerência flexível de processos de software com o ambiente WebAPSEE," in 20th Software Engineer Brazilian Symposium – Frameworks Session, Florianópolis – Brazil, 2006.

[4] B. Bruegge, A. D. Lucia, F. Fasano and G. Tortora, "Supporting distributed software development with fine-grained artefact management," in Proceedings of the IEEE International Conference on Global Software Engineering, pp. 213-222, 2006.

[5] C. A. L. Reis and R. Q. Reis, "Laboratório de engenharia de software e inteligência artificial: construção do ambiente WebAPSEE," in ProQuality, vol. 3, UFLA, 2007, pp. 43-48.

[6] E. Sales, C. L. Reis and R. Q. Reis, "Apoio à gerência de artefatos de software integrado a execução de processos de software," in 22nd Software Engineer Brazilian Symposium, 2008.

[7] G. Murphy, "Attacking information overload in software development," in IEEE Symposium of Visual Languages and Human-Centric Computing, 2009.

[8] G. H. Travassos and A. R. Rocha, "O Modelo de Integração de Ferramentas da Estação TABA", Ph.D. Thesis, 1994

[9] I. M. Gimenes, "Uma introdução ao processo de engenharia de software: ambientes e formalismos" in 13ª Jornada de Atualização em Informática, 1994.

[10] M. Kersten and G. C. Murphy, "Mylar: a degree-of-interest model for IDEs," in Proceedings of the 4th International Conference on Aspect-Oriented Software Development, pp. 159-168, 2005.

[11] M. Kersten and G. C. Murphy, "Using task context to improve programmer productivity", in Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 1-11, 2006.

[12] R Matinnejad and R. Ramsin, "An analytical review of process-centered software engineering environments," in Proceedings of the IEE 19th International Conference and Workshops on Engineering of Computer-Based Systems, 2012.

A Solution to the State Space Explosion Problem in Declarative Business Process Modeling

Renata M. de Carvalho, Natalia C. Silva, Cesar A. L. Oliveira, Ricardo M. F. Lima
Center for Informatics, Federal University of Pernambuco, Brazil
{rwm, ncs, calo, rmfl}@cin.ufpe.br

Abstract

Declarative business process models focus on modeling what must be done but do not determine how. The existing engine for controlling the execution of declarative processes uses automata-based model checking. Unfortunately, the well-known state space explosion problem limits the ability to explore large processes through automata-based approaches. In this work, we propose a novel mechanism to control the execution of declarative business processes. Our approach has the advantage of not requiring the computation of all reachable states. This allows for the modeling and execution of larger business processes when compared to the automata-based approach.

1 Introduction

Declarative business processes surged from the necessity for supporting process execution in complex or changing environments [4]. The declarative approach describes business processes by means of *business rules* that state what one can, can not, or should do to produce the desired output. It informs *what* has to be done, but not *how*.

The first work to propose the declarative paradigm of process modeling was published by Pesic [4]. Her work also proposes the *DECLARE* system, which is a tool capable of modeling and interpreting declarative process models [4]. *DECLARE* employs Linear Temporal Logic (LTL) to formally model and verify business rules. In order to control the execution of a business process, *DECLARE* converts the LTL formula into a finite non-deterministic automaton (FNDA). This automaton contains all possible execution paths for the business process, according to its rules.

Unfortunately, the number of states in an automaton increases exponentially with the size of the LTL formula [2]. As the number of business rules increases, the corresponding automaton becomes very complex and its generation too expensive, if not prohibitive. Since companies usually have a large number of business rules, the process execution may turn out to be impossible in many practical situations.

To tackle this problem, we propose a mechanism to verify the process rules at runtime. Our approach does not rely on generating all possible paths. Instead, we employ an efficient algorithm to block the execution paths that would lead to unacceptable behavior. For example, our algorithm prevents the process from reaching deadlock states. At the same time, we allow the execution of all activities that are valid according to the process business rules. Such strategy allows for the execution of larger business processes when compared to the automata-based approach.

2 Declarative Business Process

Declarative processes propose the use of *declarative languages* to define business rules that drive the business process execution [4]. This approach allows the process participants to take context-aware decisions during the process execution. On the other hand, one can define constraining business rules to prevent participants from performing prohibited or undesired activities.

The *DECLARE* framework, proposed by Pesic [4], is a tool for modeling and executing declarative processes. It offers rule templates that can be used to construct a process' business rules. These templates are mapped into formal expressions described in Linear Temporal Logic (LTL) and interpreted through a reasoning engine to identify enabled and prohibited activities during process execution.

ConDec, a graphical template language, is a constraint-based language developed by Pesic [4] for modeling business rules in declarative business processes. Its semantics is formally specified in LTL. Its graphical representation aims at improving its usability by non-LTL experts. Each constructor of the graphical language corresponds to a constraint template modeled by an LTL formula. These templates are classified in four groups, but for this paper only the following three are considered:

- **existential templates**: express the number of times an

activity can or needs to be executed in a process instance. This group comprises: *existence (A, N)* – indicates that an activity A must be executed at least N times; *absence (A, N)* – indicates that an activity A must be executed *at most* N times; and *init (A)* – indicates the first activity to be executed in the process.

- **relational templates**: express dependencies between activities. This group has five templates: *precedence (A, B)* – indicates that B cannot be executed before an activity A; *response (A, B)* – indicates that, for each execution of A, a certain activity B must be executed afterwards; *succession (A, B)* – corresponds to the conjunction of response and precedence; *co-existence (A, B)* – indicates that, if A is ever executed, certain activity B must also be executed and vice-versa; and *responded-existence (A, B)* – indicates that if A is ever executed, activity B must also be executed (either before or after A);

- **negation templates**: a negative version of the relational templates. For example, the template *not response (A, B)* indicates that after the execution of A, the activity B cannot be executed anymore; *not co-existence* indicates that after the execution of A, the activity B cannot be executed anymore, and vice-versa.

The analysis of LTL formulae rely on the generation of a Büchi automaton representing all acceptable traces that conform to the LTL formula [2]. The problem with this approach is that this automaton grows exponentially with model size and tends to become very large for complex models. Experiments we have conducted using ConDec showed that the number of states becomes too large even with a few dozen rules. In real business applications, a business process may require about thirty to fifty rules or more. This makes the use of ConDec impractical for most real business processes.

3 An approach for declarative processes that avoids the state space explosion problem

This paper proposes a mechanism to verify rules at runtime and control the execution of declarative processes. Our approach does not generate all possible execution paths.

Our proposed approach is called **ReFlex**. Once ReFlex starts the process execution, it interacts with the user to show the enabled activities at each execution point. ReFlex also warns the user when the process termination is enabled or disabled according to the pending activities. Figure 1 shows an overview of the proposed execution environment. During the modeling phase, the user specifies a business process in ConDec language (activities and constraints). This model is then compiled into a structure that stores the necessary information (activities states, counters, and rules fulfillment). ReFlex uses it to interpret the rules.

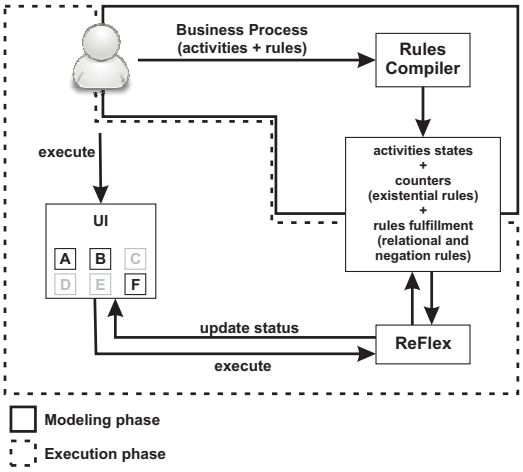


Figure 1. Overview of ReFlex.

In our approach, we adopt an additional state-based rule template that is not present in ConDec. The new rule is called "precedent-obliged (A, B)" and indicates that B cannot be executed while A is in the "obliged" state.

In the execution phase, ReFlex manages the structure generated by the compiler. This task involves changing activity states, and enabling/disabling the process termination. ReFlex is also responsible for interacting with the user interface. The interface notifies the user about activities enabled/disabled for execution. The user can only select enabled activities for execution. After executing an activity, ReFlex updates the structure and sends the new status to the user interface.

3.1 Problem representation

The concept of *activation* of a ConDec constraint was introduced by Burattin *et al.* [3]. The *activation* of a constraint in a trace is an event whose occurrence imposes some obligations on other events in the same trace. If the imposed obligation is fulfilled then the activation is considered a *fulfillment*; if the obligation is not fulfilled, it will be called a *violation*. Based on the concepts of *fulfillments* and *violations* introduced, Definition 3.1 shows how activities of a business process are represented in this work.

Definition 3.1 (Business process activities representation) *The activities of a process P are represented in a tuple $S_P = (\texttt{enabled}_P, \texttt{disabled}_P, \texttt{blocked}_P, \texttt{obliged}_P)$, where:*

- $\texttt{enabled}_P$ *is the set of enabled activities (the user is free to execute any activity in the set);*

- $\texttt{disabled}_P$ *is the set of disabled activities (the user cannot execute such activities at current execution point; these activities can became enabled in the future);*
- $\texttt{blocked}_P$ *is the set of blocked activities (activities that cannot be executed anymore). It is the set of activities that represent a violation to some rule in the process;*
- $\texttt{obliged}_P$ *is the set of obliged activities (activities that must be executed before the process termination). It is the set of activities that represent a fulfillment to some rule in the process.*

Throughout the process execution, a number of properties must remain valid. Definition 3.2 lists these properties.

Definition 3.2 (Properties of BP activities) *Let A be the set of activities in a process P.*

- $\forall a \in A, a \in \texttt{enabled}_P \cup \texttt{disabled}_P \cup \texttt{blocked}_P$*;*
- $\texttt{enabled}_P \cap \texttt{disabled}_P = \emptyset$*;*
- $\texttt{blocked}_P \cap \texttt{enabled}_P \cap \texttt{disabled}_P = \emptyset$*;*
- $\texttt{obliged}_P \subseteq \texttt{enabled}_P \cup \texttt{disabled}_P$

Burattin *et al.* also introduce the notion of *healthiness* of a trace. The *healthiness* of a process trace can be quantified based on the number of *activations*, and the number of *fulfillments*, and *violations* of these *activations*. A trace is "healthy" with respect to a constraint if the *fulfillment* ratio is 1 (one) and the *violation* ratio is 0 (zero).

The main concern of this work is to guarantee that the traces generated by the proposed approach are "healthy". In other words, we want to guarantee that all *activations* will be fulfilled and none of them will be violated.

Whenever an activity is executed, the activities states are updated to represent the process status. Table 1 describes how the activities states are updated according to the behavior of each type of rule.

During the compilation process, we identify triples (A, B, C) of activities that are inter-related such that A obliges C and B blocks C. These triples may cause deadlock if A and B are both executed before C. To avoid this situation, we apply the following *liveness-enforcing rule*.

Definition 3.3 (Liveness-enforcing rule) *For every triple (A, B, C) where A obliges C and B blocks C, include a new rule not_response(B, A) and a new rule precedent_obliged(C, B).*

Notice that the addition of these new rules may cause the emergence of new triples, which require the addition of new rules until all triples that may cause a deadlock have been handled.

Table 1. Behavior of each rule in the proposed approach.

Rules	Behavior
Existential Rules	
init(A)	All activities but A are disabled. After A is executed, the remaining rules determine the process status.
existence(A, n)	A is obliged until it is executed n times.
absence(A, n)	After $n-1$ executions of A, it is blocked.
exactly(A, n)	A is obliged until it is executed n times, but after n executions, it is blocked.
Relational Rules	
response(A, B)	After the execution of A, B is obliged.
precedence(A, B)	While A is not executed, B is disabled.
succession(A, B)	After the execution of A, B is obliged, but it is disabled while A is not executed.
coexistence(A, B)	If A is executed, B is obliged, and vice-versa (only for the first execution).
responded_existence(A, B)	The first execution of A obliges B.
Negation Rules	
not_response(A, B)	After the execution of A, B is blocked.
not_coexistence(A, B)	After the execution of A, B is blocked, and *vice-versa*.
State-Based Rules	
precedent_obliged(A, B)	While A is obliged, B is disabled.

4 Complexity of the proposed approach

In this section we analyze the memory requirements of our algorithm comparing it against the LTL-based approach.

Space complexity of an algorithm is defined in terms of what limits the number of tape cells a Turing Machine (TM) needs to use during its computation. Definition 4.1 presents the formal definition of deterministic space bounded computation [1].

Definition 4.1 (Space-bounded computation [1]) *Let $S : \mathbb{N} \rightarrow \mathbb{N}$, $s \in S$, and $L \subseteq \{0,1\}^*$. We say that $L \in$ **SPACE**$(s(n))$ if there is a constant c and a TM M deciding L such that at most $c \times s(n)$ locations on M's work tapes (excluding input tape) are ever visited by M's*

head during its computation on every input of length n.

Considering a TM with two tapes (a read-only input tape and a read/write work tape), on the read-only tape the input head can read symbols but not change them. The work tape may be read and written in the usual way. Only cells of work tape contribute to the space complexity in such a TM [6].

A complexity class is a set of problems of related complexity defined by factors such as: (i) the model of computation; and (ii) the resources that are being bounded. **L** is the class of problems that are decidable in logarithmic space on a deterministic Turing machine [6]. In other words, $L = SPACE(log\ n)$.

In our case, the input tape stores all activities and rules of a process. Let us say that this requires n cells. Looking at the declarative problem defined in Section 3, the work tape is used to store: (i) for each activity, its state; (ii) for each existential rule (except init), a single cell containing a counter for how many times the associated activity was executed; and (iii) for each relational and negation rule, a single cell with a mark indicating if the rule was fulfilled. Hence, the amount of data stored in the work tape is always smaller then the input tape. Rules require more than a single cell to be stored in the input tape, and, on the other hand, they require only one cell in the work tape. Figure 2 shows a representation of the TM that represents our approach. We consider that the existential counters are limited to a number less than infinite, so they can fit in a single cell if we use a sufficiently large tape alphabet.

Once the size of the work tape is always less than the input n, our algorithm is in the **L** complexity class.

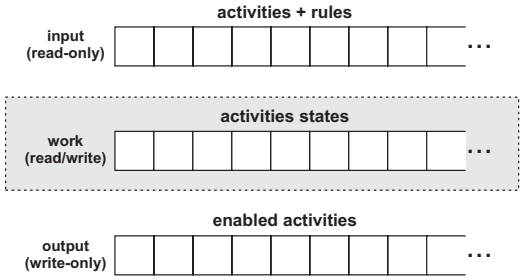


Figure 2. Space bounded computation for the proposed approach.

Comparing to the LTL approach, it is known that the complexity of model checking for LTL is in **PSPACE** complexity class [5]. **PSPACE** is the class of problems that are decidable in polynomial space on a deterministic TM. This is due to the state space explosion problem. As our approach does not share this problem, the problem is decidable in logarithmic space (**SPACE(log** n**)**).

5 Conclusions

In this paper, we approached the state space explosion problem in the context of declaraive business processes. LTL-based approaches require the computation of an automaton representing all acceptable traces. Unfortunately, this is often impracticable for most real business processes.

To tackle this problem, we proposed a mechanism that stores in activities and rules states all the information necessary to control the process. At the compilation process, a liveness-enforcing mechanism blocks the execution of paths that result in deadlocks, without affecting valid execution paths. Such mechanism disables or blocks activities at the necessary moments in order to prevent the user from mistakenly driving the process into unacceptable states. This strategy allows for the efficient execution of large business process models using the declarative paradigm.

We compared our approach to the traditional LTL-based approach, in terms of space complexity. While the LTL-based approach faces the state space explosion problem, which is an problem of **PSPACE** complexity, the mechanism proposed in this paper reduces the problem to the **L** class of complexity, which allows for an efficient implementation of a declarative business process engine.

References

[1] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, New York, NY, USA, 1st edition, 2009.

[2] Luboš Brim, Ivana Černá, Pavel Krčál, and Radek Peláek. Distributed ltl model checking based on negative cycle detection. In *Foundations of Software Technology and Theoretical Computer Science*. Springer Berlin / Heidelberg, 2001.

[3] Andrea Burattin, Fabrizio Maria Maggi, Wil M. P. van der Aalst, and Alessandro Sperduti. Techniques for a posteriori analysis of declarative processes. In *EDOC*, pages 41–50, 2012.

[4] Maja Pesic. *Constraint-Based Workflow Management Systems: Shifting Control to Users*. PhD thesis, Technische Universiteit Eindhoven, The Netherlands, 2008.

[5] Philippe Schnoebelen. The complexity of temporal logic model checking. In *Proceedings of the 4th Workshop on Advances in Modal Logic (AIML'02)*, pages 481–517. King's College Publications, 2003.

[6] Michael Sipser. *Introduction to the Theory of Computation*. International Thomson Publishing, 2nd edition, 2006.

Context Factors: What they are and why they matter for Requirements Problems

Corentin Burnay, Ivan J. Jureta
Fonds de la Recherche Scientifique – FNRS, Brussels
Department of Business Administration
PReCISE Research Center
University of Namur, Belgium
{corentin.burnay,ivan.jureta}@unamur.be

Stéphane Faulkner
Department of Business Administration
PReCISE Research Center
University of Namur, Belgium
stephane.faulkner@unamur.be

***Abstract*—When eliciting requirements, it is important to understand why some information may remain implicit, while other are shared by stakeholders. This requires knowing which variables influence if an individual shares implicit information during requirements elicitation. Based on our past experimental work on decision-making, we identify variables – *Context Factors* (CFs) – which influence whether implicit information is shared, and we define a procedure to validate CFs. Our contribution is that we present and define a set of CFs, we define an experimental procedure to validate CFs, and we discuss how the understanding of CFs helps identify information that can remain implicit during elicitation, and can thereby help to increase the completeness of requirements. We relate CFs to the common Requirements Problem concept, and we highlight the main limitation of our results.**

***Keywords*—Requirements Problem, Elicitation, Context Factors, Implicit Information**

I. Introduction

Although Requirements Engineering (RE) is a complex activity, the basic problem that it aims to solve within a systems engineering project – called the *Requirements Problem* (RP) – can be stated in simple terms: *Given assumptions about the domain in which the system-to-be should run, and the requirements of the system's stakeholders, find and describe a design of that system which is consistent with the domain assumptions, and together with domain assumptions, satisfies the requirements.* If we denote domain assumptions with K, requirements with R, and the description – specification – of the system-to-be with S, the idea above amounts to the RP statement from Zave & Jackson [1]: given K and R, find S such that $K,S \vdash R$, where $\vdash$ is some consequence relation.

An RP instance is the result of elicitation and design activities a requirements engineer does. If we denote with I all information that she can potentially elicit from stakeholders and discover or design on her own, we can split I into I_X, the information that the engineer manages to document, and other information I_M that remains **implicit**. Since we assume that K, R, and S are documented, they are parts of **explicit** information I_X.

The obvious **risk** that this **explicit/implicit distinction** highlights, is that *the implicit K_M, S_M, and R_M may include key information on stakeholders' expectations and assumptions, so that if they remain implicit, the requirements engineer may end up solving a wrong RP instance.* The goal of this paper is to advance our understanding of this risk and contribute to research on how to mitigate it. To do so, we view the relation between I_M and I_X as a function:

$$I_X = f(CF, I), \tag{1}$$

where I_X depends on the available information and other variables, which we call *Context Factors* (CFs). The objective of the paper is then to *identify, justify and suggest a way to validate the content of CFs*. Our contribution is that we identify variables that go into CF, and that we define an experimental procedure for validating CFs, *based on our past experimental work* on the role of implicit information in decision-making [2], [3].

The rest of this paper is organized as follows. We first discuss the relation between RP and CFs, and introduce the notion of a filtered RP (§II). We define CFs, and provide their taxonomy (§III). We define our experimental procedure (§IV). We discuss related work (§V), and summarize conclusions and directions for future research (§VI).

II. Filter in Requirements Elicitation

A. Introductory Example

Consider the chief financial officer of a firm. She wishes her company's accounting software to be connected with the ecommerce software, so that ecommerce sales can be automatically recorded. Denote this proposition with p. Once p is communicated to the engineer, it is part of explicit information I_X. Also, since p seems to be a requirement, it would likely be part of R_X in the RP, provided p remains relevant during RE.

How p fits an RP instance, depends on other information in both I_X and I_M. In I_X, because it may already be part of I_X that accounting software is not to be changed through work on the new system-to-be, so that p as a requirement falls outside the scope of the systems engineering project. In I_M, because the requirements engineer may assume that, the stakeholder was not informed about the scope of the systems engineering project, and so gave p on the basis of wrong assumptions.

Suppose now that the stakeholder makes assumptions when giving p, that the requirements engineer does not know so that

they are in I_M. For instance, that the accounting department needs to be motivated to use the system-to-be, that this motivation can arise if they see the system-to-be will simplify their work and that the connection with the ecommerce software is a simplification of that work. Those assumptions directly relate to the context of the business.

The way we see the situation in the example, is that the explicit information which will be shared by a stakeholder i depends on information accessible to her, and on other variables, the CFs: $I_{Xi} = f(CF(i), I_i)$. By $CF(i)$, we mean that the importance of specific CFs will vary across stakeholders. We do not write CF_i, because we are interested in CFs that apply across different stakeholders. That is, we assume that CFs will influence information sharing for all stakeholders, but we cannot assume that every CF has exactly the same effect on every stakeholder.

Equation $I_{Xi} = f(CF(i), I_i)$ is not good enough, as it does not account for the role of elicitation strategies that the engineer applies. If we denote E elicitation strategies, such as interviews, observation, study of documentation, etc., and we denote g the engineer who applies these strategies, we have the following:

$$I_{Xi} = f(CF(i), E_g, I_i). \tag{2}$$

We can then relate CFs to the RP, as follows: The *explicit* Requirements, Domain assumptions, and Specifications, are based on information which is *filtered*. We consider this observation important, because it leads us to consider what the filters may be, and how they work. This suggests there are three types of filters:

- *Expertise*, in the sense that individual stakeholders, based on their interests, responsibilities, and so on, have access to part of all information that may be relevant. Hence the subscript on I: I_i need not be equivalent to I_j of a stakeholder j;
- *Context Factors* (*CF*) influence which information will be shared, and depend on the environment other than the elicitation strategies;
- *Elicitation strategies* (E) that the engineer applies.

For instance, as financial officer of the company, the stakeholder may have limited knowledge about the system's design and the kind of information that it requires to be correctly performed. Because of her *expertise* in a domain, she may not be conscious that some information she has is relevant to RE. She will therefore filter it involuntarily. The situation in which the elicitation happens also acts as a filter. For instance, considering that the company has financial troubles would probably influence the communication of the officer, because in such context an error would be even more damageable. Hence, the stakeholder can be influenced by some specific *context factors*. Finally, a stakeholder will filter information depending on what is being asked to her and how it has been asked, regardless of her expertise or the context. In other words, the stakeholder is influenced by *elicitation strategy* adopted by the engineer.

B. Requirements Problem Filters

As an answer to the previous observations, we suggest the **filtered requirements problem**. Considering the previous example, we claim it is more relevant to write the RP as $K_X, S_X \vdash R_X$, instead of $K, S \vdash R$: the former reminds us that domain assumptions, requirements, and the specification are the result of filtering. In $K_X, S_X \vdash R_X$, we read K_X, S_X, R_X as follows:

- R_X is the result of the engineer's analysis of individual stakeholders' requirements, where for each stakeholder i, we have $R_{Xi} = f(CF(i), E_g, R_i)$;
- K_X is the result of analysis of individual stakeholders' domain assumptions, where for each stakeholder i, we have $K_{Xi} = f(CF(i), E_g, K_i)$;
- S_X is the result of the engineer's decisions on the design of the system-to-be.

The main problem of RE is then to make sure having the largest K_X, R_X and S_X and hence minimize the implicit/filtered information. We are aware that there can be interaction between CFs and Elicitation strategies, and that it may be more appropriate to put them together, as one combination of variables. Yet, we keep the distinction, because we believe that we should first attempt to understand CFs separately from E_g, in part because research on the design of Elicitation strategies can be informed by a better understanding of CFs.

C. Internal and External Factors

Before we proceed further on CFs, we explain the reasoning that led us to the filtered RP, and the three types of filters. Our basic assumption is that a stakeholder's reasoning, when deciding to share information, is non-monotonic: the stakeholder checks the information to share against her assumptions, chooses to share, does share, and can retract the information (change her mind) if she finds out new information, which invalidates her prior assumptions. Research on non-monotonic reasoning distinguishes two types of variables that influence reasoning, namely internal and external variables.

Internal variables concern the way an individual uses knowledge and heuristics in reasoning [4], [5], [6], [7]. Factors are internal when they are not specific to the environment in which the reasoning takes place. We leave all internal variables as being related to I_i in $I_{Xi} = f(CF(i), E_g, I_i)$, that is, as influencing the information from which the stakeholder picks that, which she will share. We are not interested in internal variables in this paper, mainly because our aim is to work on factors that can be influenced in a more straightforward manner, being related to the environment of the stakeholder. This restriction in the scope of our research also means that we cannot recognize if/when there are interactions between internal and external variables. We consider, however, that this limitation should not hold us back from drawing relevant conclusions about external variables.

External variables concern the environment of stakeholders and are not related to reasoners [8], [9], [10]. We divide these variables into CFs and E_g, as we want to emphasize that, while elicitation is an external influence on stakeholders, it is not the only one which influences how stakeholder shares.

The rest of the paper focuses on a taxonomy for CFs and ways to validate it.

III. Context Factors

CFs are variables characterizing the context of the stakeholder during elicitation. The purpose of identifying and analyzing CFs is to adjust elicitation strategies to the specifics of the context. In this paper, we consider operational definitions of context [11], [12], as their are composed of a finite set of well defined dimensions.

Operational definitions are useful when deciding about elicitation strategies, since there exists a finite number of context dimensions – or CFs – to investigate. To make sure the engineer does not miss key implicit information, questions should be asked about each CF. This brings us to the related question of *which CFs there are*. We explore this question below: we give an operational definition of context, that we argue accounts for the dimensions of context that are relevant in the scope of a requirements elicitation.

A. Taxonomy of Context Factors

Strictly speaking, CFs are not a set of concepts defining a particular context. By defining CFs, we are not defining an entity-relationship-like model of a particular context. If we model objects in a given context as instances of concepts, then CFs are meta-concepts, the instances of which are concepts that we would expect to recur over different specific contexts. We see the set of these meta-concepts as a non-exhaustive checklist to use in improving the elicitation of information about a specific context. CFs are issued from an extended review of the literature on context and ubiquitous computing [3], and is composed of six categories that have been proved to be relevant for RE.

Some CFs are dimensions dealing with the range of a context, i.e. the *scope of context*. The engineer must be careful to identify and investigate these factors, in order to adapt the elicitation strategy, determine the scope of context and hence increase completeness of I_X.

Items deal with *salient entities existing inside the context*. They can be instantiated across two groups, human and artificial: human items are real and living entities that likely interact with the system (e.g. stakeholders), while artificial items are typically objects that have been created by human items (e.g. softwares, devices, furniture, etc.). Such distinction matters, since human and artificial instances likely have different impact on filtering process, and therefore on the filtered RP instance.

Rules deal with *constraints existing in the context, which hold true regardless of Items' states*. They refer to notions such as laws, cultures, habits, etc. Rules deal with any constraint that applies to elements of the context and which survives after their death. Rules deal with the nature and the content of the constraints, but also with their justification and status inside the context.

Localization deal with *the position of the context*. Localization divides into two subcategories: one relating to the time, the other dealing with place. Defining the scope of a context requires the review of Items and Rules combined with Localization Factors to support engineer in determining at what time and what place formerly identified instances are to be considered.

Some other CFs must be studied to make sure scoped elements of the context have been sufficiently detailed, i.e. the *depth of context*. Depth CFs do not make sense if considered without regards to scope. It is not relevant – or at least not efficient – to ask detailed questions to stakeholders if the scope is not correctly defined.

Connections deal with *the relationships/links between Items and/or Rules*. They focus on the way entities of the scoped context relate to each other.

Activities deal with *objectives of those Items instances, which are intentional*. They refer to set of goals and intentions of such Items existing in the scope of context.

Granularity deal with *the nature, the quantity and the level of information that is provided about Items*. They must be considered at two different levels. A first level called "Micro" handles factors related to instances. A second level called "Macro" deals with information that only makes sense a the level of the entire context.

The six previous categories together form what we consider to be the set of CFs. A major drawback of our approach is the impossibility to demonstrate these categories are the only to be part of CFs. If the set of factors we identify as being part of CFs is referred to $B = \{I, R, L, C, A, G\}$, then we have that $CF \supseteq B$. We claim this set of factors is necessary but not sufficient for eliciting information about context. By necessary, we mean that omitting any of these categories would lead to misunderstanding about how CFs filter, and hence to gaps in the identification of I_{Xi}.

B. Elicitation Strategy

Beside the filter they represents in the RP, CFs can be used to adjust the Elicitation strategies E_g to the specifics of the context, and hence improve completeness of the elicitation. By asking the right questions in the right way, we claim it is possible to enforce the stakeholder's filter and get more information than what would be shared spontaneously. This section illustrates what CFs are, and how they suggest questions to be asked to stakeholders in order to increase completeness of I_{Xi}. This discussion in not exhaustive and suggests no methodology. It simply shows how CFs can identify types of information to elicit. Questions are to be considered in the context of our introductory example.

Items factors lead to questions such as:

- Who are the members of the accounting team? Who else need access to the system?
- From which system(s) do e-commerce data come? What hardware is used to exchange data?
- What are the expected output of the system? What is meant by sales results? What are the reports? Who are the requesters of these outputs?

Rules factors may lead to questions such as:

- What are the regulations for the treatment of e-commerce data? Who/What is the source of these norms, guidelines, etc.? What if the rule is violated?
- What are the internal rules from the management? What conditions make these rules applicable?
- What are the best practice, the norms and the guideline in e-commerce?

Location factors may lead to questions such as:

- When and where the reports will be accessed, be published, be communicated?
- What is the frequency of sales results updates? When do transactions happen? Where are they recorder? How much time do they occur in time?
- How and how often do Items change over time?

Connection factors may lead to questions such as:

- What are the relations between the departments of the company? Do they work in collaboration or in conflict? How much are part of the relation?
- How strong is a connection between departments? What are the consequences if it disappears? What is the purpose of the relation? Which department is active in this connection?

Activity factors may lead to questions such as:

- Why are the reports important to the financial officer? What does she expect from it?
- What is the core business of the company? What is the vision and strategy for business?
- Who sets the targets of the company? What are the mechanisms to ensure the compliance?

Granularity factors lead to questions such as:

- Do we have a clear understanding of the metrics expected by the business? Do we need to refine?
- Do we have enough details about the kind of data that are required to compute sales results?
- Is there some imprecision in the available data? How detailed and precise is the information?

IV. Prototype Experiment

We claimed in this paper that stakeholders retain information according to three filters: expertise in a domain, CFs and Elicitation strategy. Moreover, we discussed the significant relations between CFs and Elicitation strategies. Although CFs presented in this paper are issued from a literature review on context, our ongoing work is the empirical validation of CFs we presented, as actual and valid members of the filter function $I_{Xi} = f(CF(i), E_g, I_i)$. This section presents a prototype experiment to be used for such validation, and is based on our past experiments on context and non-monotonic reasoning [3], [2].

A. Questionnaire

It consists of six distinct groups of "assertions" that are to be read as pieces of context. Those assertions are potential instances of the CFs meta-concepts. Two tasks are required for each assertion. Subjects are first asked whether they judge the assertion relevant in the scope of an IT project, i.e. if it would be more difficult to perform RE without that assertion being clarified. In a second time, subjects are asked about the likelihood that a customer discuss the assertion spontaneously. Based on this double evaluation, it is possible to determine whether CFs are relevant to RE, but also to analyze whether practitioners consider CFs as being filters in the communication of stakeholders.

The list of assertions used in our experiment is presented in Table 1 (one line per CF). The table is the result of authors experience, and aims to stay relevant with the sample of questions suggested in section III-B. Evaluation of the assertions is done on a 5-level scale. Subjects have the possibility to suggest other relevant assertions, for each CF.

A pilot study has been conducted with a dozen subjects, to ensure the questionnaire is easily understandable, the instructions are clear, and the Internet based collection method is appropriate. Feedback of subjects were positive, and preliminary results suggest interesting results for a larger scale experiment. The pilot study also contributed to the definition of the final assertion list.

Situation described to subjects: Your colleague asks you some advice for next mission. She has to design a system, and plans a first conference call with the customer to collect as much relevant information as possible. She is not sure about how to proceed during the interview, and asks you some advice.

Instruction described to subjects: Knowing about your experience, she asks you to evaluate some assertions that she thinks will enable her to have a good idea of the business. Her concern is to cover as much content as possible. You are asked to judge the relevance of her assertions, and the likelihood that a customer discuss it spontaneously. Use a value between 1 and 5 to express the relevance/confidence toward statements, 5 being the largest grade.

B. Target Population and Procedure

We will mainly submit the questionnaire to professionals in computer sciences, software and requirements engineers. We also target participants with experience in project management, ICT or other relevant management skills. Information about participant's professional status will be collected and treated to ensure the validity of the experiment.

To collect feedback on the questionnaire, and repeat the experiment, the questionnaire will be published on the Internet, under the form of a regular questionnaire. Participants will be asked to carefully read the assignment before answering. The assignment will clearly ask to read the problem statement and the different choices that are offered before starting the grading activity.

The approach of the experiment is top-down. We start from some broad meta-concept – CFs – and try to validate instances of these CFs that are relevant in RE. This implies the list of assertions suggested in this experiment is not complete. This also implies the work should be extended through additional studies.

TABLE I. SUMMARY OF ASSERTIONS SUBMITTED TO SUBJECTS' EVALUATION

	Q1	Q2	Q3	Q4	Q5
I	Units and Structure of the business (department, team, ...)	People interacting with the system	Objects related to the system	Other IT systems of the company	Input and Output of the system
R	Applicable laws and regulations	Best practices in business	Recommendations and Constraints from management	Norms, guidelines or culture	Habits, traditions of the business
L	Place/time where or when the system is used	Frequency of use of the system	Aspects of the business changing over time	Phenomena occurring at regular interval	Synergies inside the business
C	Types of relations between items (friends, enemies, etc.)	Respective power of agents inside a relation	Nature of important relations in the business	Importance of these relations to the company	Strength of the relation
A	Core business of the company	Vision and Strategy for the business	Purpose of the system	Intention behind the IT solution	Goal and Targets assigned to employees
G	Metrics relevant to the business	Legal or Financial status	Atmosphere in offices, on market	History, Evolution in the past of the business	Particularities of the company

V. RELATED WORK

The empirical validation of factors that influence decision-making – and among other the decision to share information – has been a center of attention in many fields of research. Yet, considering origins of our contribution, we specifically focus this section on context studies in non-monotonic reasoning literature (NMR) and context in RE.

As already explained, we see two different categories of CFs. A first one is related to human cognition and influences what information I_i is accessible to stakeholder. Ford and Billington [4], [5] propose an experiment to validate the impact of such subjects-related factors. They present factors such as *the reluctance to draw conclusion based on conflicting rules* or *the number of positive and negative sentences*, which they argue influence the consistence of subjects when reasoning. Hewson and Vogel [6] present an experiment which suggests human reasoning is consistent with some basic assumptions of NMR literature, but found that people do not always satisfy literature's predictions when reasoning about a chain of negative sentences. Vogel [7] proposes an extension of previous study, designing an experiment to test other forms of negative reasoning.

A second category includes factors which are not, strictly speaking, specific to the person and influence the content of I_{Xi}. Wason and Shapiro [10] propose experiments that emphasize the difference in performance between subjects, depending on the way problems are introduced to them. They suggest these differences are due *to the concreteness of terms that are used* – some are concrete while others are abstract terms– , thereby emphasizing the intrinsic influence of information. Elio and Pelletier [8], [9] propose that reasoning is likely to be influenced by several external factors. They highlight that whether an objects is *naturally-occurring* or *artificial* influences the way people think about this object. They also discuss the influence of other factors like the *quantity of information* that is provided or the information about the *relative size* of the objects.

In addition to previous NMR studies, we performed several experiments to get better insight into the influence of external factors on RE and elicitation. For instance, we showed that some external factors proposed by Elio and Pelletier do not have the same influence in the context of RE [2]. We also have some other ongoing experiments on context [3]. We propose a Context Framework – issued from a literature review on context-aware and ubiquitous computing – to account for several dimensions of context and their influence on reasoning in RE.

There has been limited attention regarding the question of accurate context's definition in RE. Yet, context as a source of information is not new. Many papers propose high level discussions about context in RE: Potts and Hsi [13], [14] emphasize the existence of *Contextualism* – opposed to abstractionism – as a possible alternative design philosophy for information systems. Sommerville et al. [15] propose discussions about how *ethnographic analysis* is value-added to RE, thereby broadening the scope of RE context to culture questions. Beyer and Holtzblatt propose the Contextual Design model [16], which increases the scope of relevant information to any data about *the field where people are living*. Previous works illustrate the trend to include even more data in the scope of RE relevant information.

Cohene and Easterbrook [17] discuss a topic closer to what we address in this paper. They suggest elicitation techniques that are used in a interview should be adapted to fit the kind of information engineers are trying to find, i.e., adapt the elicitation technique to the situation – or context.

Previous related works highlight how valuable information about context is to RE. But few papers propose a structured definition of context. Sutcliffe et al. [18] go on a method for requirements analysis which aims to accounts for individual, personal goals and the effect of time and context on requirements. They suggest a list of aspects to deal with, but do – to the best of our kowledge – no empirical validation. RE community seems to agree on the importance of further research on the link between context and RE. Cheng and Atlee [19] stress the importance of context and empirical validation of RE models as a direction for future research to accelerate the transfer of research results into RE practice.

VI. Conclusions

In this paper, we tried to contribute to the clarification of how stakeholders may keep some relevant information implicit during the elicitation of requirements. We argued that instances of the classical RP are obtained through a filtering process, we suggested a conceptualization of variables that influence the filters, and called them CFs. We discussed how CFs relate to requirements problem, suggested a list of CFs, and proposed an experimental procedure for validating the relevance of some individual CFs.

Our ongoing work consists of applying the experimental procedure, the resulting refinement of CFs, and the study of how CFs relate to requirements elicitation strategies. We believe that this work is relevant with regards to increasing the rigor in requirements elicitation, and for enabling future design of systematic elicitation strategies. We also see this research as part of ongoing RE research on defining and using the notion of context in requirements.

References

[1] P. Zave and M. Jackson, "Four dark corners of requirements engineering," *Transactions on Software Engineering and Methodology*, vol. 6, no. 1, pp. 1–30, 1997.

[2] C. Burnay, I. Jureta, and S. Faulkner, "Influence of context on decision making during requirements elicitation," in *ARCOE 2012*, 2012, pp. 39–51.

[3] ——, "Context-driven elicitation of default requirements," *CoRR*, vol. abs/1211.2620, 2012.

[4] M. Ford and D. Billington, "Strategies in human nonmonotonic reasoning," *Computational Intelligence*, vol. 16, no. 3, pp. 446–468, 2000.

[5] M. Ford, "Human nonmonotonic reasoning: the importance of seeing the logical strength of arguments," in *Synthese*, 2005, pp. 71–92.

[6] C. Hewson and C. Vogel, "Psychological evidence for assumptions of path-based inheritance reasoning," in *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*, 1994, pp. 409–414.

[7] C. Vogel, "Human reasoning with negative defaults," in *Lecture Notes in Artificial Intelligence: Practical Reasoning*. Springer, Berlin, 1999, pp. 606–621.

[8] R. Elio and F. J. Pelletier, "Human benchmarks on ai's benchmark problems," in *Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society*, 1993, pp. 406–411.

[9] ——, "On relevance in non-monotonic reasoning: Some empirical studies," in *Relevance: American Association for Artificial Intelligence 1994 Fall Symposium Series*, 1994, pp. 64–67.

[10] P. C. Wason and D. Shapiro, "Natural and contrived experience in a reasoning problem," *Quarterly Journal of Experimental Psychology*, vol. 23, pp. 63–71, 1971.

[11] D. Lenat, *The Dimensions of Context-Space*. Technical Report, Cycorp, 1998.

[12] A. Zimmermann, A. Lorenz, and R. Oppermann, "An operational definition of context," in *Proceeding of the Sixth International and Interdisciplinary Conference on Modeling and Using Context*, 2007, pp. 558–571.

[13] C. Potts and I. Hsi, "Abstraction and context in requirements engineering: Toward a synthesis," *Ann. Software Eng.*, vol. 3, pp. 23–61, 1997.

[14] C. Potts, "Requirements models in context," in *Proceedings of the 3rd IEEE International Symposium on Requirements Engineering*, 1997, pp. 102–104.

[15] S. Viller and I. Sommerville, "Social analysis in the requirements engineering process: From ethnography to method," in *Proceedings of the 4th IEEE International Symposium on Requirements Engineering*, 1999, pp. 6–13.

[16] H. Beyer and K. Holtzblatt, *Contextual design: defining customer-centered systems*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998.

[17] T. Cohene and S. M. Easterbrook, "Contextual risk analysis for interview design," in *Proceedings of the 13th IEEE International Conference on Requirements Engineering*, 2005, pp. 95–104.

[18] A. G. Sutcliffe, S. Fickas, and M. M. Sohlberg, "Pc-re: a method for personal and contextual requirements engineering with some experience," *Requirements Engineering*, vol. 11, no. 3, pp. 157–173, 2006.

[19] B. H. C. Cheng and J. M. Atlee, "Research directions in requirements engineering," in *Workshop on the Future of Software Engineering, FOSE 2007*, 2007, pp. 285–303.

Detecting traceability links through neural networks

Andre Di Thommazo
Instituto Federal de São Paulo, IFSP
Universidade Federal de São Carlos, UFSCar
São Carlos, Brazil
andredt@ifsp.edu.br

Thiago Ribeiro, Guilherme Olivato, Rafael Rovina
Instituto Federal de São Paulo, IFSP
São Carlos, Brazil
{guilhermeribeiro.olivatto, thiagoribeiro.d.o,rafarovina}
@gmail.com

Vera Werneck
Universidade do Estado do Rio de Janeiro, UERJ
Rio de Janeiro, Brazil
vera@ime.uerj.br

Sandra Fabbri
Universidade Federal de São Carlos, UFSCar
São Carlos, Brazil
sfabbri@dc.ufscar.br

Abstract— Although a Requirements Traceability Matrix (RTM) is one of the most commonly used ways to represent requirements traceability, it is difficult to create it manually. This scenario motivates the investigation of alternatives to generate the RTM automatically. This paper presents an approach to automatically create an RTM based on a neural network, called RTM-N, that combines two other approaches, one based on the entry data of functional requirements – called RTM-E – and another based on natural language processing – called RTM-NLP. Data obtained from a first experimental study which evaluated RTM-E and RTM-NLP were used to train a neural network and make it capable of detecting the requirements traceability automatically, thus generating the RTM-N. With the aim of evaluating the RTM-N and re-evaluating the RTM-E and RTM-NLP, another experimental study was conducted. On average, the approaches showed the following results in relation to the reference RTM: RTM-E had 78% effectiveness, RTM-NLP had 76% effectiveness, and RTM-N had 85% effectiveness. The results show that using a neural network to combine and generate a new RTM was more effective in determining the requirement dependencies and, consequently, the requirements traceability links.

Keywords—requirements traceability; neural networks; requirements traceability matrix.

I. INTRODUCTION

The importance of requirements management for the software development process is highlighted by several authors in the Computer Science literature [1],[2],[3]. Salem [4] considers that the majority of software errors are derived from errors in requirements elicitation and in keeping up with their evolution throughout the software development process.

Data from a research performed by the Standish Group [5] found that the three most important factors (approximately 37%) for software project unsuccessful were: user specification gaps, incomplete requirements, and constant changes in requirements. These factors are directly related to requirements management, where the requirements traceability matrix (RTM) is an essential element as it records the existing relationship among system requirements. Due to its importance, it is the main focus of many researches. Sundaram, Hayes, Dekhtyar, and Holbrook [6] consider traceability determination an essential task for many software engineering activities, although it is a time-consuming and error-prone task. The authors claim that this task can be facilitated through computational support, as a tool can reduce the effort and cost of elaborating and maintaining the requirements traceability. These authors also mentioned that the automatic generation of RTMs is still very limited in the existing tools.

According to Cleland-Huang, Gotel, and Zisman [1], the researches which have recently addressed requirements traceability have focused on automatic traceability definition. Moreover, Wang, Lai, and Liu [7] point out that these current researches make use of spatial vector models, semantic indexing, or probability network models as ways to automate traceability. In spatial vector modeling, the frequency of terms in the text of the artifacts is used to determine the traceability links. The semantic indexing—proposed by Deerwester, Dumais, Furnas, Landauer, and Harshman [8]—uses the context of the terms in the text to identify the traceability links. The probability network models [9] create a matrix where dependency between each term is mapped in relation to the other document terms. Based on this matrix, the traceability links are generated.

All the quoted proposals are also detailed by Cleland-Huang, Gotel, and Zisman [1] as possible approaches for traceability detection.

Given the aforementioned context, this paper presents an approach to automatically create an RTM based on neural networks. This approach is called RTM-N and combines two other proposed approaches: one based on the functional requirements (FR) entry data – called RTM-E – and another based on natural language processing (NLP) –called RTM-NLP.

A neural network can store knowledge acquired through examples and make inferences about new ones. Therefore, using the data from an experimental study that evaluated the RTM-E and RTM-NLP approaches [10], we created a neural network that is capable of determining the level of dependence between FRs.

A second experimental study was conducted to evaluate the effectiveness of the RTM-N and re-evaluate the RTM-E and RTM-NLP. This experimental study is detailed in this paper. To make the experiment possible, the three RTM automatic generation approaches were implemented in the COCAR tool [11]

This paper is organized as follows: in Section II, the requirements management, traceability, and RMT are introduced; Section III presents a brief definition of the neural networks theory; in Section IV, the three RMT automatic creation approaches are presented; Section V shows the experimental study performed to evaluate the approaches' effectiveness; conclusions and future work are discussed in Section VI.

II. Requirements Traceability

The main objective of requirements management is organizing and storing all requirements as well as managing any changes to them [2],[3]. The requirements are constantly changing during the software development process and managing them usually becomes a laborious and extensive task [7]. One way of managing requirements is to define the requirements traceability, which concerns the ability to describe and monitor a requirement during its whole lifecycle [12]. Traceability is a technique which allows the dependency relationship between pairs of requirements and between requirements and other artifacts generated during the software development process to be identified and visualized.

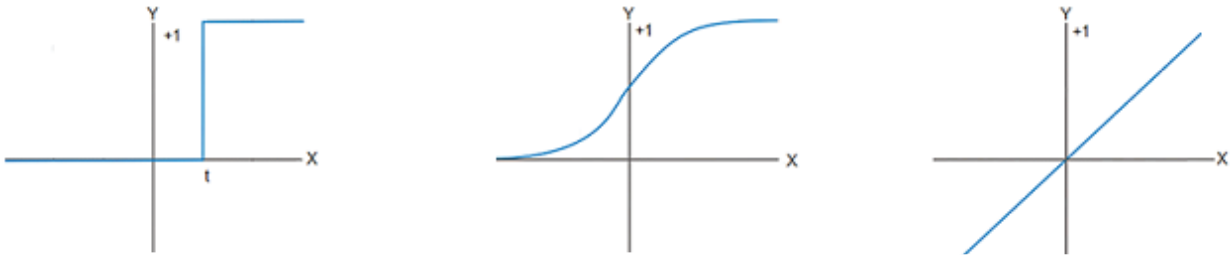
According to Guo, Yang, Wang, Yang, and Li [12], requirements traceability is an important requirements management activity as it can provide the basis for evolutional changes in requirements besides acting directly on the quality assurance of the software development process. Although various researches have considered the traceability between requirements and other artifacts, minor attention has been given to the relationships between requirements [13]. A way to map such relationships between requirements is by creating an RTM.

In general, the RTM is constructed as follows: each FR is represented in the i-th line and in the i-thcolumn of the RTM and the dependency between them is recorded in the cell corresponding to each FR intersection [2]. The importance of and need for the RTM in the software development process is debated by several authors [2],[12],[13],[14]. The RTM allows the impact of a change (or the insertion of a new requirement) on the system as a whole to be predicted.

Many authors also highlight the difficulty in determining and maintaining the RTM, noting that this task is laborious and error prone. Sommerville [2] emphasizes the difficulty in obtaining this kind of matrix and overcomes it by proposing a way to subjectively indicate not only whether the requirements are interdependent but how strong such dependency is.

III. Neural Networks

Neural networks have the ability to acquire knowledge for pattern recognition [16]. They are inspired by the human brain and are composed of several artificial neurons. The artificial neurons were created by McCulloch and Pitts[17]. Each neuron (or node) in a neural network receives a number of input values. A function, called the activation function, is applied to these input values and the neuron activation level is generated as the function result that corresponds to the output value provided by the neuron. The activation function can vary and the main types are represented in Figure 1. The x-axis represents the input value for each neuron and the y-axis represents the output (or the level of activation) provided by the neuron.



(a) Step function (b) Sigmoid function (c) Linear function

Figure 1. Examples of activation functions (adapted from [16]).

Figure 2 shows the representation of a neuron receiving various inputs (x_i) and their weights (w_i). The output value provided by the neuron depends on the inputs, the weights, and the activation function used. The behavior of the neuron can be defined by Equation 1.

$$X = \sum_{i=1}^{n} w_i x_i \qquad (1)$$

According to the calculated value of X, the Y value is determined by the activation function, generating the output provided by the neuron.

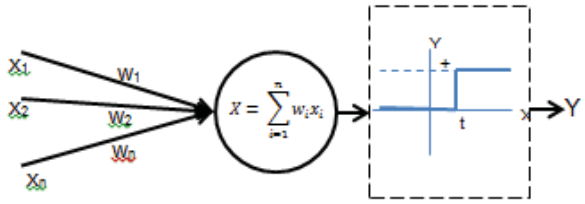


Figure 2. Schematic of an artificial neuron.

There are multiple classifications for neural networks. These classifications may depend on different characteristics: (i) the number of layers or the type of connectivity, fully connected or partially connected; (ii) the flow of the processed signals, feed-forward or feed-back; (iii) the way the training is done, supervised (when desired input and output data are presented to the neural network) or unsupervised (when only the input data are presented to the neural network and it is in charge of setting the output values). The training consists of presenting input patterns to the network such that it can modify their weights. Thus, its outputs should present an adequate response when the input data provided are similar but not necessarily identical to those used in training [18].

The RTM-N approach, based on neural networks, for determining the traceability is detailed in Section IV. The results of the two other approaches – RTM-E, which explores the relationship between the entry data manipulated by the FRs, and RTM-NLP, which uses NLP – are used as the neural network input data for training the neural network. As output data, the network determines the level of dependency between the FRs ("no dependence", "weak dependence", "strong dependence").

IV. APPROACHES TO RTM DEFINITION

The three RTM automatic generation approaches proposed in this work take only the FRs into account when establishing the degree of relationship between each pair.

The approaches were implemented in the COCAR tool, which uses a template [19] to collect the requirements entry data. This template contains the necessary data for composing a Requirements Document (RD) and supporting the application of the approaches. The main objective of using such a template is to standardize the registration of the FRs, avoiding inconsistencies, omissions, and ambiguities. An important field of this template that deserves attention is called "Entry" and is essential for the RTM-E approach application, as explained below. In a previous work dependency graphs were also generated [10]. However, as the focus of this paper is the RTM generation, visualization graphs will not be explorated

In the following, the approaches are presented.

A. RMT-E Approach

The objective of this approach is to detect the traceability links through the FRs entry data.

The dependency relationship between two FRs is determined by the percentage of data common to them both. To calculate such a relationship, the Jaccard Index [20] is used. It compares the similarity and/or diversity degree between any two data sets (A and B, for example). Equation 2 represents this index where the numerator is given by the quantity of data intersecting both sets (A and B) and the denominator corresponds to the quantity of data in the union of those sets.

$$J(A,B) = \frac{n(A \cap B)}{n(A \cup B)} \quad (2)$$

Considering FRa the entry data set of a functional requirement A and FRb the entry data set of a functional requirement B, their dependency level is calculated by Equation 3. Hence, each position (i,j) of the traceability matrix RTM(i,j) corresponds to the value generated by Equation 4. The RTM generated by the RTM-E approach is named RTMe.

$$J(FRa, FRb) = \frac{n(FRa \cap FRb)}{n(FRa \cup FRb)} \quad (3)$$

$$RTM(i,j) = J(FRi, FRj) \quad (4)$$

It is worth noticing that initiatives to automatically determine the RTM using FR entry data were not found in the literature. Similar initiatives do exist to help determine traceability links between other artifacts, mainly models (UML diagrams) and source code, like the ones proposed by Cysneiros and Zisman [21].

From two FRs, the RTM-E approach generates a number that represents the percentage dependency between them. Then, based on this number, the dependence is classified as "no dependence", "weak dependence", or "strong dependence". The dependency level values were chosen according to an interactive and iterative method based on the data provided by three RDs from different ranges. These levels were "no dependence" for a dependency level value equal to 0%, "weak dependence" for values between 0% and 50%, and "strong dependence" for values greater than 50%.

B. RMT-NLP Approach

The objective of this approach is to detect the traceability links through NLP.

Even though there are many initiatives that make use of NLP to determine traceability in the software development process, few of them consider traceability inside the same artifact [13], as mentioned before. In addition, the proposals found in the literature do not use a requirements description template and do not determine dependency levels as done in this work.

Aiming to determine the dependency level between two FRs, this approach uses the Processing field of the

template and the Frequency Vector and Cosine Similarity methods [22]. Such methods provide the percentage similarity between two text excerpts.

Text pre-processing is performed before applying the Frequency Vector and Cosine Similarity methods, with the aim of improving the process efficiency. The first step is to eliminate all words, called stopwords, that might be considered irrelevant (articles, prepositions, and conjunctions). Then, a process known as steaming is applied. This step reduces all words to their original radicals, leveling their weights in the text similarity determination. After the two aforementioned steps, the method calculates the similarity between two FRs texts using the Processing field of the template. The RTM generated by the RTM-NLP approach is named RTMnlp.

As done in the RTM-E, the dependency level values were chosen in an interactive and iterative way according to the same three RDs from different ranges. The obtained levels were “no dependence” if the value was between 0% and 40%, “weak dependence” for values between 40% and 70%, and “strong dependence” for values above 70%.

C. *RTM-N Approach*

The objective of this approach is to detect traceability links combining the RTM-E and the RTM-NLP through neural networks.

A Multilayer Perception (MLP) neural network was used to develop the RTM-N approach. It is composed of source nodes that represent the network input layer, one or more intermediate layers, and an output layer. Except for the input layer, the others are composed of neurons (circles). Furthermore, the MLP network connectivity is feed-forward; that is, the output of each neuron connects only to all the neurons of the next layer, without the presence of feedback loops. Thus, the signal propagates in the network progressively.

Aiming to model the FR traceability, the MLP neural network was created with two entries: the values generated by the RTM-E and by the RTM-NLP. The outputs were: "no dependence", "weak dependence", and "strong dependence" and the network topology is detailed in Figure 3.

To train the neural network it is necessary to "teach" the network, providing the correct patterns. Therefore, we provided it with data from an experimental study conducted to evaluate the approaches RTM-E and RTM-NLP [10]. In this experimental study a reference RTM was created for each of the 18 systems that were analyzed. This reference RTM was constructed based on the detailed analysis of each FR pair, determining the dependency between them as “no dependence”, “weak dependence”, or “strong dependence”. Thus, in the training phase, values between 0 and 1, meaning the percentage dependency calculated by the RTM-E and RTM-NLP approaches, were provided to the neural network as input data. The correct relationship, marked in the reference RTM, was provided as output data.

Once the neural network has been created and trained, when it is provided with new input data obtained by the RTM-E and RTM-NLP approaches, the level of dependence between the involved FRs can be automatically identified. The RTM generated by the RTM-N approach is called RTMn.

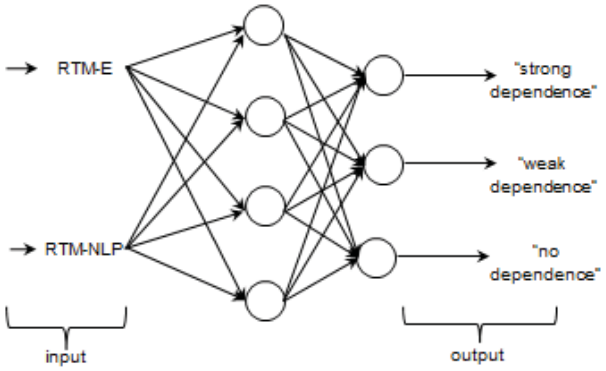


Figure 3: RTM-N Topology

V. Experimental Study

This study was conducted to evaluate the effectiveness of the RTM-N and re-evaluate the RTM-E and RTM-NLP aproaches. It has been conducted following the guidelines in Table I.

The results of the comparison between the data in RTMe, RTMnlp, and RTMn are presented in Table II. The first column contains the name of the specified system, the second contains the FR quantity, and the third presents the total number of possible dependencies between FRs that may exist (“no dependence”, “weak dependence”, or “strong dependence”) and whose formula was shown in Figure 6. The fourth, sixth, and eighth columns contain the total number of coincidental dependencies between the RTMe, RTMnlp, and RTMn matrices. For example, if the RTM-Ref has determined a “strong” dependency in a cell and the RTM-E approach has also registered the dependency as “strong” in the same position, the correct relationship has been identified. The fifth, seventh, and ninth columns show the effectiveness of the RTM-E, RTMnlp, and RTMn approaches, respectively, which are calculated by the relation between the quantity of correct dependencies found by the approach and the total number of dependencies that could be found (third column).

The statistical analysis was conducted using SigmaPlot software. The normal distribution of data was confirmed applying the Shapiro-Wilk test and the results were expressed in mean ± standard deviation. To compare the effectiveness between the proposed approaches (RTM-E, RTM-NLP, and RTM-N), variance analysis (ANOVA) was used with Holm-Sidak post-test. The significance level adopted was 5%. The RTM-N approach was found to be more effective than RTM-E (85% ± 0.05 versus 77% ± 0.05; $p=0.001$) and than the RTM-NLP approach (85% ± 0.05 versus 76% ± 0.07; $p<0.001$)

TABLE I – EXPERIMENTAL STUDY GUIDELINES

Context	The experiment has been conducted in the context of the Software Engineering class at UFSCar – Federal University of São Carlos. The experiment consisted in each pair of students conducting the requirements gathering of a system involving a real stakeholder. The final RD had to be created using the COCAR tool.
Objective	Evaluation of the effectiveness of the RTM-E, RTM-NLP, and RTM-N approaches in comparison to a reference RTM (called RTM-Ref) constructed by a detailed analysis of the RD.
Participants	Twenty-eight graduation students of the Bachelor's Computer Sciences course at UFSCar
Artifacts	RD: • produced by a pair of students on their own; • related to a real application, with the participation of a stakeholder with broad experience in the application domain; • related to the information systems domain with basic creation, retrieval, update, and deletion of data, and inspected by a different pair of students to identify and eliminate possible defects; • Included in the COCAR tool after the identified defects have been removed.
	RTM-Ref • created from the RD input into the COCAR tool; • built based on the detailed reading and analysis of each FR pair, determining the dependency between them as "no dependence", "weak dependence", or "strong dependence"; • recorded in a spreadsheet so that the RTM-Ref created beforehand could be compared to the RTMe, RTMnlp, and RTMn for each system; • built by the paper´s authors, who always contacted the RD's authors whenever a doubt was found.
Metrics	The effectiveness of the three approaches with regard to the coincidental dependencies found by each approach in relation to the RTM-Ref. The effectiveness is calculated by the relation between the quantity of dependencies correctly found in each approach against the total of all dependencies that can be found between the FRs. Considering a system consisting of n FRs, the total quantity of all possible dependencies (T) is given by Equation 6: $T = \frac{n(n-1)}{2}$ (6) Therefore, the effectiveness rate is given by Equation 7: $\text{Effectiveness} = \frac{\text{Quantity of correct relationships found}}{T}$ (7) The Precision and Recall [1] metrics could not be used, given that such metrics only take in account the fact that a dependency exists and not their level ("weak" or "strong").
Threats to validity	• Students' inexperience to develop the requirements with the stakeholders. To minimize this risk, known domain systems were used as well as RD inspection activities. • RTM-Ref had been built by people who did not have direct contact with the stakeholder. To minimize this risk, any doubts about relationship occurrence were asked to the students.

TABLE II . EXPERIMENTAL STUDY RESULTS

			RTM-E		RTM-NLP		RTM-N	
System	**# FRs**	**Number of possible dependencies**	**Correct**	**Effectiveness**	**Correct**	**Effectiveness**	**Correct**	**Effectiveness**
Zoo	19	171	131	77%	138	81%	149	87%
Habitation	24	276	233	84%	205	74%	235	85%
Student flat	28	378	295	78%	325	86%	337	89%
Taxi	15	105	82	78%	77	73%	91	87%
Clothing store	27	351	295	84%	253	72%	319	91%
Freight	16	120	98	82%	85	71%	102	85%
Court	24	276	204	74%	181	66%	218	79%
Financial control	17	136	94	69%	101	74%	107	79%
Administration	19	171	134	78%	129	75%	148	87%
Book store	19	171	129	75%	145	85%	149	87%
Ticket	15	105	88	84%	91	87%	95	90%
Movies	16	120	88	73%	82	68%	91	76%
Bus	15	105	72	69%	78	74%	81	77%
School	15	105	82	78%	77	73%	91	87%

VI. CONCLUSIONS AND FUTURE WORK

This paper presented an approach based on neural networks—RTM-N—to automatically generate the RTM. Neural networks are able to acquire knowledge for pattern recognition. Thus, from patterns generated in an experimental study, a neural network was trained for automatic detection of the level of dependency between FRs.

Two other approaches proposed and presented in this paper—the RTM-E, which is based on the percentage of entry data that two FRs have in common, and the RTM-NLP, which uses NLP to determine the level of dependency between requirements—were used as input of the neural network.

From the three approaches presented, it is worth mentioning that there are already some reported proposals in the literature using NLP for traceability link determination, mainly involving different artifacts (requirements and models, models and source-code, or requirements and test cases). Such a situation is not found in the RTM-E, and no similar attempt was found in the literature to generate traceability links between FRs.

All approaches were implemented in the COCAR environment. Then, the experimental study was performed to evaluate the effectiveness of each approach. The results showed that RTM-N presented superior effectiveness compared to the other two. This was because the RTM-N used the results presented in the other two approaches and in the RTM-Ref to train the neural network to recognize the level of traceability (“no dependence”, “weak dependence”, and “strong dependence”).

Hence, considering that RTM determination is a difficult task, the RTM-N approach combined the two approaches and knowledge of analysts who constructed the RTM-Ref to create a way to automatically detect the traceability links.

The results motivate the continuity of this research and new investigations of how to better combine the approaches for the creation of the RTM using neural networks.

The main contributions of this particular work are incorporated into the COCAR environment and correspond to the automatic relationship determination between the FRs. This facilitates the evaluation of the impact of a change in a requirement on the others. As future work, it is intended to improve the NLP techniques used, considering the use of a *tagger* and the incorporation of a terms glossary for synonym treatment. Another investigation to be done concerns how an RTM can aid the software maintenance process, and more specifically, the support for regression tests generation.

REFERENCES

[1] J. Cleland-Huang, O. Gotel, and A. Zisman, Software and Systems Traceability, Springer, 491 p., 2012.

[2] I. Sommerville, Software Engineering. 9th edition – New York-Addison Wesley, 2010

[3] A. Zisman and G. Spanoudakis, "Software Traceability: Past, Present, and Future", The Newsletter of the Requirements Engineering Specialist Group of the British Computer Society, September 2004

[4] A.M. Salem, "Improving Software Quality through Requirements Traceability Models," 4th ACS/IEEE Int. Conf. Computer Systems and Applications (AICCSA 2006), Dubai, Sharjah, UAE, 2006.

[5] Standish Group, CHAOS Reports, 1994. Available at http://www.standishgroup.com/sample_research/chaos_1994_2.php. Last accessed February 2007.

[6] S.K.A. Sundaram, J.H.B. Hayes, A.C. Dekhtyar, and E.A.D. Holbrook, "Assessing traceability of software engineering artifacts," 18th Int. IEEE Requirements Engineering Conf., Sydney, Australia, 2010.

[7] X.Wang, G. Lai, and C. Liu, "Recovering relationships between documentation and source code based on the characteristics of software engineering," Electron. Notes Theor. Comput. Sci., 2009.

[8] S. Deerwester, S.T. Dumais, G.W. Furnas, T.K. Landauer, and R. Harshman, “Indexing by latent semantic analysis,” J. Am. Soc. Inf. Sci., vol. 41, no. 6, pp. 391–407, 1990.

[9] R. Baeza-Yates, A. Berthier, and A. Ribeiro-Neto, Modern Information Retrieval. ACM Press/Addison-Wesley, 1999.

[10] A. Di Thommazo, G. Malimpensa, G. Olivatto, T. Ribeiro, and S. Fabbri, "Requirements traceability matrix: automatic generation and visualization," in Proc. 26th Brazilian Symp. Software Engineering, Natal, Brazil, 2012.

[11] A. Di Thommazo, M. D. C. Martins, and S. C. P. F. Fabbri, “Requirements Management in COCAR enviroment” (in portuguese) WER 07 - Workshop de Engenharia de Requisitos, 2007, Toronto, Canada.

[12] Y. Guo , M. Yang , J. Wang, P. Yang and F. Li, "An Ontology based Improved Software Requirement Traceability Matrix", 2nd International Symposium on Knowledge Acquisition and Modeling, KAM , Wuhan, China, 2009

[13] A. Goknil, I. Kurtev, K. Van den Berg and J.W. Veldhuis, "Semantics of trace relations in requirements models for consistency checking and inferencing", Software and Systems Modeling , Volume 10 Issue 1, February 2011

[14] E. V. Munson, and T. N. Nguyen, “Concordance, conformance, versions, and traceability”; Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering, Long Beach, California, 2005.

[15] D. Cuddeback, A. Dekhtyar and J.H. Hayes, "Automated requirements traceability: The study of human analysts", Proceedings of the 2010 18th IEEE International Requirements Engineering Conference, RE2010, Sydney, Australia, 2010

[16] B. Coppin, Artificial Intelligence Illuminated, Jones and Bartlett Publishers, 729 p. , 2004.

[17] W. S Mcculloch, W. Pitts “A logical calculus of the ideas immanent in nervous activity” The bulletin of mathematical biophysics, v. 5, n. 4, p. 115–133, 1 dez. 1943.

[18] A. O. Artero, Inteligência Artificial - Teorica e Pratica (in Portuguese), Livraria da Fisica, 230 p., 2009.

[19] K.K. Kawai, "Guidelines for preparation of requirements document with emphasis on the Functional Requirements" (in portuguese). 2005. 170 f. (Master in Computer Cience)- Universidade Federal de São Carlos, São Carlos, 2005.

[20] “The Probabilistic Basis of Jaccard's Index of Similarity” – Avalilable at: http://sysbio.oxfordjournals.org/content/45/3/380.full.pdf Last access on November, 2012

[21] G. Cysneiros and A. Zisman, "Traceability and Completeness Checking for Agent Oriented Systems". Proceedings of the 2008 ACM symposium on Applied computing, New York, USA, 2008.

[22] G. Salton and J. Allan, "Text retrieval using the vector processing model," in 3rd Symp. Document Analysis and Information Retrieval, University of Nevada, Las Vegas, 1994.

Generating Ontologies through Organizational Modeling

Karen Najera
INFOTEC
Mexico D.F., Mexico
Karen.najera@infotec.com.mx

Alicia Martinez
CENIDET
Cuernavaca, Mexico
amartinez@cenidet.edu.mx

Anna Perini
FBK
Trento, Italy
perini@.fbk.eu

Hugo Estrada
INFOTEC
Mexico D.F., Mexico
hugo.estrada@infotec.com.mx

Abstract— Ontologies are recognized as important component of information systems supporting business processes within and across organizations. At modeling time, they contribute identifying key elements from business processes; at development time, their structure can be translated automatically into information system's source code; finally, at run time, through queries and reasoning, they provide proper data for decision making. Additionally, ontologies provide the basis for sharing and publishing organizational information through the Semantic Web. However, representing organizational information directly into an ontology requires specialized expertise in the ontology engineering domain, thus, ontologies are not generated using a domain language easily for the organizational domain experts. In this work, we propose the use of specialized organizational modeling techniques as starting point to model the organization, thereby, ensuring the proper definition of the organizational knowledge. Then, a mechanism is provided to automatically transform the organizational knowledge in its corresponding ontological representation. Our proposed approach is based on Model Driven Engineering ideas and it involves: a) the development of an ontology representing the metamodel of two widely used organizational modeling techniques *i and Tropos and b) the systematic transformation of *i** based modeling primitives into instances of the ontology.**

Keywords-Organizational modeling; ontologies; organizational knowledge base; Model-Driven Engineering

I. INTRODUCTION

Ontologies are becoming popular to be an important component of information systems that supports business processes within and across organizations. Ontologies can give support to the system lifecycle: at modeling time, ontologies can be used to identify and describe key elements from business processes such as data, activities and profiles involved in the process itself (e.g. [1]); at development time, the structure of an ontology can be automatically translated into the source code of an information system by using an appropriate development support environment, as described for instance in [2] where business knowledge represented as OWL ontology is automatically translated into an information system implemented in the Mercury programming language. Hence, if changes of a business process are reflected in the ontology, the information system will also automatically reflect those changes. Finally, at run time, ontologies can add semantics to specify the behavior of business process [3], for instance, by using queries and reasoning to retrieve proper data for decision making or process validation (e.g., [4, 5, 6]). Additionally, ontologies provide the basis for sharing and publishing organizational information through the Semantic Web [7].

However, representing organizational information directly into an ontology is not an easy task. A crucial step in building good quality ontologies is the right involvement of domain experts. As argued in [8, 9], traditional methodologies and tools are based on the idea that knowledge engineers drive the modeling process. This often creates an extra layer of indirectness which makes the task of producing and revising ontologies too rigid and complex, e.g., for the needs of business enterprises. Therefore, in this work we propose the use of specialized organizational modeling techniques as starting point for the capture and representation of organizational knowledge, thereby ensuring their proper definition. Then, we aim to provide a mechanism to automatically generate the corresponding ontological representation from the organizational model.

The *i** visual modeling language [10] is one of the most widely used organizational modeling techniques [11]. It supports the description of networks made up of social actors of an enterprise and the social intentional relationships and dependencies among them together with the representation of the internal behaviors required to satisfy actor dependencies. *i** provides the modeling basis to software engineering methodologies that support the early requirements elicitation stage such Tropos [12]. Therefore, we propose to start capturing the needed organizational knowledge with the *i** or Tropos modeling languages and automatically generate the corresponding ontological representation in the standard semantic web language Web Ontology Language (OWL) [13]. This is done to enable enterprise domain experts, with low knowledge engineering skills, to effectively represent organizational knowledge such as: strategy, structure, processes, and behavior, information and system requirements, in terms of ontologies.

In this paper, we present an approach based on Model Driven Engineering ideas that involves: a) the development of an

ontology representing the metamodel of two widely used organizational modeling techniques *i** and Tropos and b) the systematic transformation of the knowledge represented in a specific *i** based model into instances of the ontology. Following this approach, we provide the automatic generation of an Organizational Knowledge Base (that we called Organizational KB), where OntoiStar embody the terminological knowledge (Tbox), that is, the knowledge about the terminology of the organizational domain, and OntoiStar instances represent the assertional knowledge (Abox), which is the knowledge coming from a specific organization description represented in an *i** based model.

As a result, and according to the organizational knowledge represented into an organizational model, we can provide: 1) organizational knowledge available to be exploited and consumed in the Semantic Web; 2) system requirements captured in the ontology for software development, such as agent systems; 3) proper data for decision making through ontology reasoning.

This paper is structured as follows: Section 2 presents the background in the organizational modeling domain. Section 3 provides the overview of our proposal. Section 4 describes the development of the proposed approach. Section 5 describes related work and finally, Section 6 concludes this work and summarizes our ongoing and future work.

II. ORGANIZATIONAL MODELING

We have as goal to represent organizational knowledge in terms of ontologies. Therefore, we propose, as starting point, the use of specialized organizational modeling techniques to model the organization, thereby, ensuring the proper definition of the organizational knowledge. For this purpose, we have selected *i** [10] and Tropos [12].

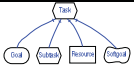
*i** supports the description of organizational networks made up of social actors who have freedom of action, but also depend on other actors to achieve their objectives and goals. It provides a visual language which includes two models: the *strategic dependency model*, a graph to represent social and intentional relationships (dependencies that describe an 'agreement') among the network of actors of an enterprise; and the *strategic rationale model*, a graph to describe and to support the internal behavior of each actor required to satisfy their dependencies on other actors. Examples of *i** primitives are presented in Table 1.

Tropos is an agent-oriented software methodology based on *i**. it provides a development process that is organized into five phases: *Early requirements*, to produce a model of the organization; *Late requirements*, to introduce the system-to-be in the model analyzing its impact in the organization; *Architectural design*, to obtain a representation of the architecture of the system in terms of subcomponents and the relationships among them; *Detailed design*, to define the software agent rationale, including capabilities and interaction specifications; and *Implementation*, which involves the production of code. The Tropos visual language uses the core concepts of *i** presented in Table 1 (with minimal differences omitted due to space).

Due to the growing interest around *i** [11], variants based on the original framework have been defined (such as Tropos and several more). Therefore, approaches for dealing the heterogeneity of the *i** variants have been proposed. We have analyzed two of these approaches [14, 15] as we aim to support the ontological representation of organizational knowledge represented not only with *i** and Tropos but also with other *i** variants. In [14] a metamodel is proposed where following a union approach the constructs of *i** and Tropos were included in the metamodel; in [15] a metamodel focused in *i**, Tropos and GRL is proposed where following an intersection approach the common constructs of the three variants were included in the metamodel; in [15] the iStarML specification language is proposed. iStarML is an XML interchange format which provides a common representational framework for *i** variants diagrams. It includes a set of tags corresponding to the core constructs of different *i** variants and a definition of attributes in each tag to represent particularities of the constructs (see Table 1, where attributes have been omitted due to space).

We have implemented the approach presented in this work, based on the metamodels and the iStarML format proposed in [14, 15].

TABLE I. *I** AND TROPOS CONSTRUCTS

***i** and Tropos core constructs*	**Modeling primitive**	**Types**	**iStarML tag**
Actor		None Role Position Agent	<actor>
Actor link		Is_a Is_part_of Occupies Covers Instance Plays	<actorLink>
Intentional element		Goal Softgoal Resource Task (*i**) Plan (Tropos)	<ielement>
Dependency (depender, dependum, dependee)		Goal Softgoal Resource Task (*i**) Plan (Tropos)	<dependency> <depender> <dependee>
Boundary			<boundary>
Intentional element link		Decomposition Means-End Contribution	<ielementLink>

III. OVERVIEW OF THE PROPOSAL

In this section, we present the overview of our proposal, which is presented in Fig. 1. The phase 1 corresponds to the development of an OWL ontology, called OntoiStar, for the ontological representation of the metamodel of *i** and Tropos.

The phase 2 consists of the automatic generation of an Organizational KB by transforming the knowledge represented in a specific *i** based model into instances of the ontology OntoiStar. *Phase 1* has been divided in two processes. Process 1 is related to the analysis of *i** based metamodels [14, 15], which was addressed to determine the constructs to be represented in OntoiStar. Process 2 refers to the generation of OntoiStar, where constructs identified in the process 1 were manually mapped into OWL constructs following MDE ideas. *Phase 2* was also divided in two processes. Process 3 is related to the graphical representation of the organization with any of the organizational modeling technics *i** or Tropos, generating an *i** based model. This process can be realized with *i** modelers or editors that enables producing a model specified in iStarML [15], for instance jUCMNav[1] or HiME[2]. Process 4 refers to the automatic transformation of the *i** based model specified in iStarML to instances of the ontology OntoiStar. In order to support this process, we have developed a tool that implements transformation rules between the iStarML format and the ontology OntoiStar according to the MDE approach. The output of the tool corresponds to the ontology OntoiStar instantiated with the knowledge described in the *i** based model, namely, the Organizational KB.

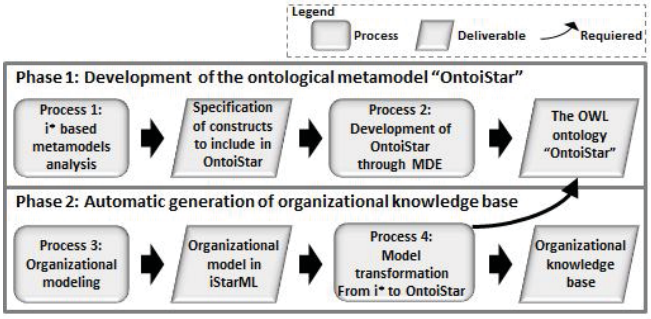


Figure 1. Overview of the proposed approach

IV. ONTOLOGY GENERATION APPROACH

In this section, we describe our proposed approach to represent the organizational knowledge in terms of ontologies. Our approach starts from models described with the organizational modeling techniques *i** and Tropos to generate an Organizational KB in the ontology domain. The approach is based on MDE ideas. MDE is a methodology which focuses on creating and exploiting domain models for the software development. It is based on layered architectures, where models, metamodels and metametamodels correspond to the M1, M2 and M3 layers, respectively. Fig. 2 shows the layered architectures of the domains that we are addressing. On the left side, it is found the layered architecture of *i** based modeling languages, and on the right side, it is found the layered architecture that we have proposed for the ontology domain. Transformation bridges [16] can be defined to move from a layered architecture to another. A transformation bridge comprises a set of transformation rules which together describe how a model conforming to the source metamodel can be transformed into a model conforming to the target metamodel. A transformation rule defines how one or more constructs in the source metamodel can be transformed into one or more constructs in the target metamodel. A transformation bridge is defined in two steps:

1) Constructs in each metamodel are identified.
2) The relationships between the constructs of both metamodels are analyzed and specified, i.e. transformation rules are defined.

A transformation bridge can be defined at the level of metamodels (M2) as well as the level of metametamodels (M3). Thus, a transformation bridge defined at level Mn can then be used to automate model to model transformation at level Mn-1.

In our approach, we have defined two transformation bridges:

M_3 transformation bridge, which has been defined in M3 layer to generate the OWL ontology OntoiStar. Therefore, it contains the transformation rules between concepts from the *i** metametamodel (represented in the Unified Modeling Language, such classes and associations) in the *i** layered architecture and concepts from the OWL metamodel (such classes and properties) in the OWL ontology architecture.

M_2 transformation bridge, which has been defined in M2 layer to transform any *i** based model into instances of the ontology OntoiStar. It contains the transformation rules between concepts from the *i** metamodel (represented in iStarML) in the *i** layered architecture and concepts from the ontology OntoiStar in the OWL ontology architecture. We have developed a tool that implements the transformation rules of this bridge in order to automate the transformation process of any *i** based models.

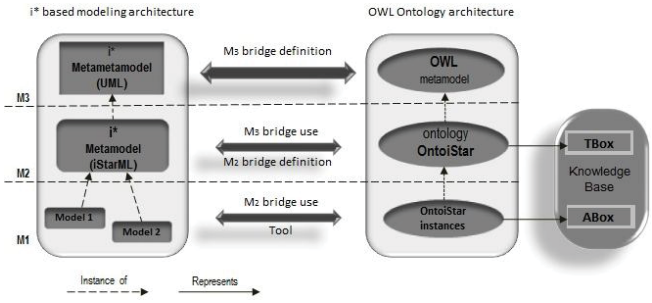


Figure 2. Arquitectural solution of the approach

The implementation of the proposed approach is described in the following subsections.

A. *Ontological metamodel development phase*

In this subsection, we describe the phase 1 of our proposed approach, that is, the development of the ontology OntoiStar which represent the ontological metamodel of *i** based modeling languages.

Process 1 was carried out in order to determine the constructs of the *i** based modeling languages to be included into OntoiStar. It consisted of an analysis of two metamodels that address the integration of several *i** variants [14, 15], as we mentioned in section II. The metamodel proposed in [14] includes all the elements of *i** and Tropos; the metamodel proposed in [15] includes only the common concepts of the

1 The jUCMNav website. http://jucmnav.softwareengineering.ca/ucm/bin/view/ProjetSEG/
2 The HiME website. http://www.lsi.upc.edu/ llopez/hime/

variants *i**, Tropos and GRL. The main differences between these metamodels lie in concepts not common, the representation of concepts relationships, the class hierarchy and the class properties. For the development of OntoiStar, we have adopted the common characteristics of both metamodels including concepts, relations and attributes, but also some specific characteristics of each one. From [14] we have adopted: a) super classes to define a class hierarchy between constructs (see *i** and Tropos core concepts in Table 1); b) enumeration classes to represent specific attributes such as contribution types: positive and negative; c) the representation of all concept relationships in terms of classes and associations. From [15] we have adopted the most classes' properties, such as, disjoint and complete. As a result of the analysis we have generated an *i** metamodel including the adopted characteristics. The resultant *i** metamodel is the basis to generate the ontology OntoiStar.

Process 2 was carried out in order to generate the ontology OntoiStar. OntoiStar has been generated by applying the *M_3 transformation bridge* (from i* metametamodel to OWL metamodel), which is defined as follows:

(1) Constructs from the *i** metametamodel and from the OWL metamodel are identified.
The *i** metametamodel is described with UML therefore, the relevant constructs to consider for the transformation are: class, attribute, association and class property.

The significant constructs of the OWL language to develop OntoiStar are: OWL Class, Object property and Data property; the axioms: subClassOf, ObjectPropertyDomain, ObjectPropertyRange, DataPropertyDomain, disjointWith and unionOf.

(2) Relationships between constructs from the *i** metametamodel and the OWL metamodel are analyzed and specified. The following transformation rules were proposed:
Rule 1: Classes and class associations from *i** metametamodel as classes in OWL.
Rule 2: Associations from *i** metametamodel as object properties in OWL.
Rule 3: Class properties from *i** metametamodel as axiom class properties in OWL.
Rule 4: Enumeration elements from *i** metametamodel as class instances in OWL.
Rule 5: Attributes (we define two kinds of attribute representation):
(a) Class attributes from *i** metametamodel as data properties in OWL.
(b) Attributes (of enumeration class) from *i** metametamodel as object properties in OWL.

The *M_3 transformation bridge* is manually applied in layer M2, transforming the *i** metamodel into its ontological representation: OntoiStar. Table 2 presents partially results of the application of Rule 1, the OWL class representation of classes and class associations of the *i** metamodel.

An example of application of Rule 2 is the actor is_a link. It represents the relationship between two actors. Therefore, the actor class "Actor" and a class to represent the is_a link "isALink" have been generated following Rule 1. Then two object properties have been defined to represent the source and target associations between the isALink class and the Actor class.

```
<owl:ObjectProperty rdf:ID="has_Actor_IsALink_source_ref">
  <rdfs:range rdf:resource="#Actor"/>
  <rdfs:domain  rdf:resource="#IsALink"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="has_Actor_IsALink_target_ref">
  <rdfs:domain rdf:resource="#IsALink"/>
  <rdfs:range rdf:resource="#Actor"/>
</owl:ObjectProperty>
```

After applying the transformation rules to all the elements of our proposed *i** metamodel, we obtain the ontological metamodel OntoiStar. Thus, we obtain the Tbox part of the Organizational KB.

TABLE II. M3 TRANSFORMATION BRIDGE

***i** and Tropos core constructs**	**Types**	**OWL construct**
Actor	None	<owl:Class rdf:about="#Actor"/>
	Role	<owl:Class rdf:ID="Role"/>
	Position	<owl:Class rdf:ID="Position"/>
	Agent	<owl:Class rdf:ID="Agent"/>
Actor link	Is_a	<owl:Class rdf:about="#IsALink"/>
	Is_part_of	<owl:Class rdf:about="#IsPartOfLink"/>
	Occupies	<owl:Class rdf:about="#OccupiesLink"/>
Intentional element	Goal	<owl:Class rdf:ID="Goal"/>
	Softgoal	<owl:Class rdf:ID="SoftGoal"/>
	Resource	<owl:Class rdf:ID="Resource"/>
	Task (*i**)	<owl:Class rdf:ID="Task"/>
	Plan (Tropos)	<owl:Class rdf:ID="Plan"/>
Dependency (depender, dependum, dependee)	Goal Softgoal Resource Task (*i**) Plan (Tropos)	<owl:Class rdf:ID="Dependency"/> <owl:Class rdf:ID="DependeeLink"/> <owl:Class rdf:ID="DependumLink"/> <owl:Class rdf:ID="DependerLink"/>
Boundary		<owl:Class rdf:ID="ActorBoundary"/>
Intentional element link	Decomposition	<owl:Class rdf:ID="DecompositionLink"/>
	Means-End	<owl:Class rdf:ID="MeansEndLink"/>
	Contribution	<owl:Class rdf:ID="ContributionLink"/>

B. Organizational Knowledge Base generation phase

In this subsection, we describe the phase 2 of our proposed approach, that is, the automatic generation of the Abox part of the Organizational KB. First, we present a tool that supports the automatic transformation of the knowledge represented in a specific *i** based model into instances of the ontology OntoiStar. Then, we describe the processes of this phase, which must be performed whenever that it is desired to represent the organizational knowledge of a specific organization in terms of ontologies.

The tool is called TAGOOn – (**T**ool for the **A**utomatic **G**eneration of **O**rganizational **O**ntologies). It is based on MDE ideas. Therefore, through a set of transformation rules implemented in the tool, it automatically populates the ontology OntoiStar with instances that represent the *i**

elements belonging to a specific organizational model. Thus, generating the Abox part of the Organizational KB.

OntoiStar instantiation is carried out by implementing in TAGOOn the *M_2 transformation bridge* (from i* metamodel, described in the iStarML specification, to OntoiStar) as shown in Fig. 3. Transformation rules between iStarML constructs (see Table 1) and the ontology OntoiStar has been defined. The transformation rules are automatically applied to *i** based models on layer M1. In this way, TAGOOn supports the automatic transformation of any *i** based model into instances of the ontology OntoiStar.

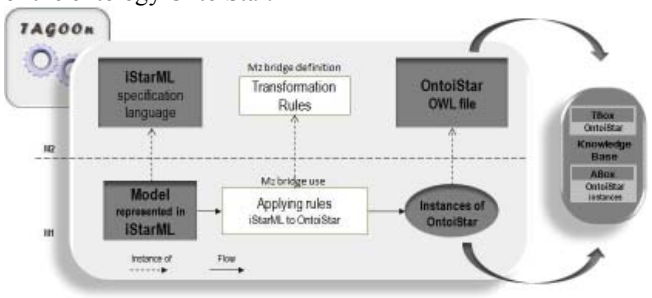


Figure 3. *M_2 transformation bridge* implemented in TAGOOn

Following we describe the Organizational KB generation process flow (Fig. 4). Process 3 consists in modeling the organization by using the visual *i** or Tropos modeling languages. This model can be realized with *i** modelers or editors that enables producing a model specified in the iStarML format. The model in iStarML is the input for the process 4, which is performed by the tool TAGOOn. The tool then parses the iStarML file, and according to the defined transformation rules, instantiate the corresponding classes and properties in the ontology OntoiStar. The output of the tool is the ontology OntoiStar with instances that represent the knowledge content in the *i** based model. The ontology OntoiStar with their instances shapes an Organizational KB in which is possible to apply services offered by the ontology technology such as reasoning and querying. The output can be edited with an ontology editor, for modifying the ontology or its instances or it can be the input of development or reasoning platforms supported by ontologies.

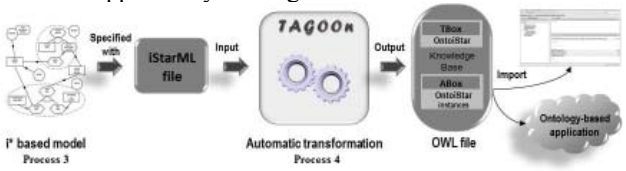


Figure 4. Organizational KB generation process flow

TAGOOn has been validated with a case study carried out in [17]. It consists of a real project to model the processes of a postgraduate institution (www.cenidet.edu.mx) that offers Master and PhD programs. We present a fragment of the process to register students in the academic semesters. The registration process involves: 14 actors, 43 actor dependencies, 117 intentional elements and 44 intentional element links. The fragment is related with the actors "Student" and "Thesis advisor" and their Softgoal dependency "Choose appropriated courses" and their goal dependency "Choose courses" (Fig. 5). Table III presents parts of the iStarML file and the resultant OWL file.

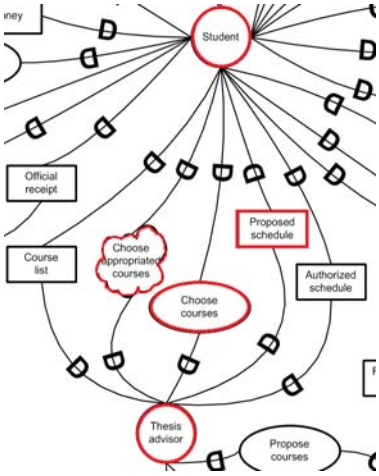


Figure 5. Fragment of a *i** model and its iStarML

TABLE III. FRAGMENTS OF THE ISTARML AND THE OWL FILES

Fragments of the iStarML file	Fragments of the OWL file
<actor id="05" name="Student"/> <actor id="06" name="Thesis advisor"/> <ielement id="01" name="Choose appropriated courses" type="Softgoal"> <dependency> <depender aref="05"/> <dependee aref="06"/> </dependency> </ielement> <ielement id="02" name="Choose courses" type="Goal">	<OntoiStar:Actor rdf:ID="Student"/> <OntoiStar:Actor rdf:ID="Thesis_advisor"/> <OntoiStar:Softgoal rdf:ID="Choose appropriated courses"/> <OntoiStar:Goal rdf:ID="Choose courses"/> <OntoiStar:Resource rdf:ID="Proposed schedule"/>

V. RELATED WORK

Solutions to the problem of modeling in various aspects of an enterprise were proposed in several works, both in terms of definition of the metamodel and in terms of methodologies to support the creation of the model itself. Concerning the metamodel the TOVE Ontology Project [18] proposes a set of integrated ontologies for the modeling of an enterprise which spans several aspects of an enterprise, such as activities, states, resources, time, and so on. The Common KADS model set [19] is a collection of models (organization, task, agent) for structuring knowledge in an organization. The organizational ontology [7] is a core ontology to represent organizational structures developed with the objective of supporting linked data publishing of organizational information across different domains. In [20], a set of ontologies are proposed to support business process integration. Focusing on methodologies for ontology / model creation, we can notice that most of them - e.g., TOVE Enterprise methodology [18], CommonKADS [19], Methontology [21] and the Enterprise ontology [22] - are built around the knowledge engineer, who executes and coordinates all the different phases of the knowledge acquisition and formalization process. The novelty of our approach w.r.t. these proposals lies in using a Model Driven

Engineering ideas to represent organizational knowledge in terms of ontologies from specialized techniques for modeling organizations, where concepts are familiar to enterprise domain experts such as a) the representation of social and intentional relationships among the network of actors of an enterprise, and b) the representation of the internal behaviors required to satisfy actor dependencies. The approach is also complemented by an intuitive visual representation language which can facilitate the involvement of enterprise domain experts in the modeling process. The ontology generation is managed in a transparent manner for the user.

VI. Conclusion and Future Work

In this paper, we presented a semi-automated approach to generate ontologies from an organizational model described with *i** or Tropos modeling languages. Specifically, it describes how the ontological metamodel of *i** based modeling languages (OntoiStar) has been developed, and it also explains the transformation process that can be applied to automatically populate OntoiStar with instances of *i** elements belonging to a specific organizational model. Thus, providing an Organizational KB where OntoiStar represent the Tbox and OntoiStar instances, represent the Abox.

Services offered by the ontology technology such as reasoning and querying can be applied to this Organizational KB. Furthermore, it can be opened and edited with an ontology editor, or it can be the input of development or reasoning platforms supported by ontologies. As the organizational knowledge is represented in the standard Semantic Web language OWL, it could be available to be exploited and consumed in the Semantic Web by paradigms such as Linked Data.

Although, we considered the case of *i** notation, the approach can be generalized to other organizational modeling frameworks, since it follows MDE ideas.

In our ongoing work, we are currently addressing the integration of other *i** variants to OntoiStar, thereby the ontology will be useful for any *i** variant. Moreover, we are consolidating the tool to support the automated generation of an Organizational KB through an *i** based model described with any of the variants integrated in OntoiStar.

References

[1] A. Prieto and A. Lozano-Tello. Use of ontologies as representation support of workflows oriented to administrative management. Journal of Network and Systems Management, 17(3):309–325, 2009.

[2] M. V, E. Bossche, P. Ross, I. Maclarty, B. V. Nuffelen, and N. Pelov. Ontology driven software engineering for real life applications, 2007.

[3] I. Weber, J. Hoffmann, and J. Mendling. Beyond soundness: on the verification of semantic business process models. Distributed and Parallel Databases, 27(3):271–343, 2010.

[4] M. Dimitrov, A. Simov, S. Stein, and M. Konstantinov. A bpmo based semantic business process modelling environment. In Semantic Business Process and Product Lifecycle Management, volume 251 of CEUR, 2007.

[5] G. Gröner and S. Staab. Modeling and query patterns for process retrieval in owl. In 8th International Semantic Web Conference (ISWC 2009), volume 5823 of LNCS, pages 243–259, Springer, Washington, DC, 2009.

[6] C. Ghidini, C. D. Francescomarino, M. Rospocher, P. Tonella, and L. Serafini. Semantics based aspect oriented management of exceptional flows in business processes. IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, 2011.

[7] D. Reynolds. An organizational ontology. W3C, https://dvcs.w3.org/hg/gld/raw-file/default/org/index.html, March, 2013.

[8] V. Dimitrova, R. Denaux, G. Hart, C. Dolbear, I. Holt, and A. G. Cohn. Involving domain experts in authoring owl ontologies. In Proceedings of the 7th Int. Semantic Web Conference (ISWC 2008), volume 5318/2010 of LNCS, pages 1–16. Springer Berlin / Heidelberg, Karlsruhe, Germany, 2008.

[9] C. Ghidini, M. Rospocher, and L. Serafini. Moki: a wiki-based conceptual modeling tool. In ISWC 2010 Posters & Demonstrations Track: Collected Abstracts, volume 658 of CEUR Workshop Proceedings (CEUR-WS.org), pages 77– 80, Shanghai, China, 2010.

[10] E. S.-K. Yu. Modelling strategic relationships for process reengineering. PhD thesis, Toronto, Ont., Canada, Canada, 1996.

[11] X. Franch. The *i** framework: The way ahead. In Sixth International Conference on Research Challenges in Information Science RCIS'12, pages 1–3, Paris, France, 2012.

[12] P. Giorgini, J. Mylopoulos, A. Perini, and A. Susi. The Tropos Methodology and Software Development Environment. In E. Yu, P. Giorgini, N. Maiden, and J. M. eds., editors, Social Modeling for Requirements Engineering, pages 405–423, Chapter 11, MIT Press, Cambridge, MA, 2010.

[13] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWL Web Ontology Language Reference. Technical report, W3C, http://www.w3.org/TR/owl-ref/, February 2004.

[14] M. Lucena, E. Santos, C. T. L. L. Silva, F. M. R. Alencar, M. J. Silva, and J. Castro. Towards a unified metamodel for i*. In Second International Conference on Research Challenges in Information Science RCIS'08, pages 237–246, Marrakech, Morocco, 2008.

[15] C. Cares, X. Franch, A. Perini, and A. Susi. Towards interoperability of *i** models using istarml. Computer Standards & Interfaces, 33(1):69{79, 2011.

[16] S. Staab, T. Walter, G. Gr¨oner, and F. S. Parreiras. Model driven engineering with ontology technologies. In Reasoning Web, pages 62–98, 2010.

[17] H. Estrada. A service-oriented approach for the *i** framework. PhD thesis, Valencia University of Technology, Valencia, Spain, 2008.

[18] M. S. Fox and M. Gr¨uninger. Enterprise modeling. AI Magazine, 19(3):109–121, 1998.

[19] G. Schreiber, H. Akkermans, A. Anjewierden, R. Dehoog, N. Shadbolt, W. Vandevelde, and B. Wielinga. Knowledge Engineering and Management: The CommonKADS Methodology. The MIT Press, December 1999.

[20] M. Grüninger, K. Atefi and M. Fox. Ontologies to Support Process Integration in Enterprise Engineering. In: Computational & Mathematical Organization Theory vol 6, n 4, pp. 381-394, 2000.

[21] M. Fernandez-Lopez, A. Gomez-Perez, and N. Juristo. Methontology: from ontological art towards ontological engineering. In Proc. Symp. on Ontological Eng. Of AAAI, Providence, Rhode Island, 1997.

[22] J. L.G. Dietz. Enterprise Ontology. Springer, Berlin / Heidelberg, 2006.

Using NLP Techniques for Identifying GUI Prototypes and UML Diagrams From Use Cases

Rafael T. Anchiêta, Rogério F. de Sousa and Raimundo S. Moura
Department of Computing
Federal University of Piauí
Teresina (PI), Brazil
E-mail: {rta, rfigsousa, rsm}@ufpi.edu.br

***Abstract*—Use cases are an effective way of modeling the interaction between user and computer system, which covers not only how the user will interact with the system but how the system will respond to the user. They are generated from the requirements specification and written in natural language narratives. This paper presents a prototype to generate GUI prototypes, class diagrams, and use case diagrams from use case descriptions written in the Portuguese language using natural language processing techniques. We use the ANTLR tool, text patterns, part-of-speech tagger, stop words, and stemming algorithms to solve the problem. A preliminary evaluation highlights promising results for the approach used.**

Keywords*— *Requirement engineering, use cases, natural language processing

I. Introduction

Requirements Engineering (RE) is an important field of the software development process. Requirements should be documented in the Software Requirement Specification (SRS).

In the SRS, we have used a textual use case, as introduced by Booch et al. [1], which constitutes the interaction between the use cases and the actors. Use cases can also be modeled in a graphical form using the Unified Modeling Language (UML) notation, serving as a table of contents for the functional requirements. They describe the interactions between a system and its environment.

In use cases, the most important part is the description of events flow (scenarios). It provides a means of communication between actors and the system. In his book [2], Alistair Cockburn presented an effective technique for specifying the interaction between a software system and its environment. The technique is based on natural language specification for scenarios and extensions. Scenarios and extensions are specified by phrases in plain English language. This makes requirements documents easy to understand and to communicate even to non-technical people.

The use of natural language to specify the behavior of a system is, however, a critical point, due to the inherent ambiguity, duplication, and omission from different interpretations of the natural language descriptions [3]. The specification of communication (interaction) between actors and system (use cases) is fundamental to the development of Graphical User Interface (GUI) prototypes and, normally, application's users validate the software requirements by the GUI prototypes.

According to IEEE STD 830-1998 [3], prototypes are useful for the following reasons:

1) The customer may be more likely to view the prototype and react to it than to read the SRS and react to it. Thus, a prototype provides quick feedback.
2) The prototype displays unanticipated aspects of the systems behavior. Thus, it produces not only answers but also new questions. This helps reach closure on the SRS.
3) An SRS based on a prototype tends to undergo less change during development, thus shortening development time.

Through the use case descriptions, it is also possible to generate class diagrams and use case diagrams.

Class diagrams help decrease the gap between the requirements analysis phase and the design phase. This gap creates inconsistency and it needs to be bridged. Class diagrams are also used for object-oriented system design and are widely used by the software industry and engineering community, because they represent the static view of the systems. Use case diagrams describe scenarios which show features of the system from the viewpoint of the user, and the customer should see the use case diagrams as the key features of your system.

In this context, in this paper we present a prototype to automatically identify GUI prototypes from use case descriptions, written in Portuguese language and that follow the Praxis[1] process using Natural Language Processing (NLP) techniques.

Previous research has shown that it is possible to identify elements of user interfaces through textual use cases [4].

The main contributions of our approach include: (1) a decrease in the inconsistency between analysis phase and design phase; (2) a minimization of development time of the GUI prototypes; (3) assistance to requirements engineers in the process of requirements elicitation; and (4) a decrease in the gap between requirements engineers and system user.

The Praxis process is used in many computer science courses and in real software development projects, especially in Synergia - Software Engineering Laboratory and Systems. It is based on object-oriented technology; its rating analysis and design is the UML 2.0, created by the Object Management Group (OMG) consortium, which brings together hundreds of

[1] http://homepages.dcc.ufmg.br/~wilson/praxis/

the world's leading producers of software. The methods of the Praxis cover the areas of the Capability Maturity Model Integration (CMMI), which is a process improvement approach that provides organizations with essential elements of effective processes. The patterns of the Praxis are in accordance with the standards of the IEEE Engineering software, the most comprehensive and respected area, specifically IEEE STD 830-1998 [3].

The rest of the paper is organized as follows. In Section II we describe an overview about NLP techniques. Section III we show some related works. In Section IV we present the developed prototype. In Section V we discuss preliminary evaluations. Finally, in Section VI we present conclusion and future works.

II. NLP Techniques: overview

Traditionally, NLP techniques have been used to analyze texts and extract structured information from non-structured data. NLP algorithms aim to identify the real importance of each term in certain contexts, enabling a gain in the quality of results produced; in other words, NLP attempts to extract a fuller meaning representation from free text. NLP typically makes use of linguistic concepts, part-of-speech, grammatical structure, as well as dealing with ambiguity and anaphora. This area includes various knowledge representations, such as a lexicon of words and their meanings and grammatical properties and a set of grammar rules and often other resources such as an ontology of entities and actions, or a thesaurus of synonyms or abbreviations [5].

According to Luisa et al. [6], several scientific studies have used NLP in RE, and have generated many contributions:

1) Assistance in the management and documentation of requirements [7], [8].
2) Assistance in the ambiguity localization and duplication in the requirements specification document [9] .
3) Extraction directly from text of natural language elements to generate conceptual models [10], [11].

A. Stop Words

One of the frequently used methods on semantic natural language processing is the elimination of stop words, which are words with little or no additional semantic information. Depending on application, they can be considered a homogeneously distributed noise signal to be filtered from other words, as they have no semantic discriminating power but dilute the apparent distribution of the words we are interested in.

In this paper, we take grammatical classes such as article, pronoun, adjective, adverb, conjunction, and numerals and put them in a set called stop list. This set is not used to identify GUI elements nor to generate use case diagrams.

B. Taggers

In NLP, taggers are systems that analyze a text and insert morphological, grammatical or syntactical tags for each lexical item. A part-of-speech tagger is a morphosyntactic tagger which analyzes the text and identifies parts of speech such as nouns, adjectives, pronouns and verbs, among others.

Basically, the tagger inserts a tag into each word; this action is performed according to the algorithm of the tagger used. The tagger can use a lexicon and a set of procedures that support the process of defining a tag to be used on a particular word. These two components are part of the language model used for the task of tagging (e.g. *Engenharia*/NOUN *de*/PREPOSITION *Requisitos*/NOUN). Figure 1 illustrates a generic scheme for a morphosyntactic tagger with the tokenization phase incorporated.

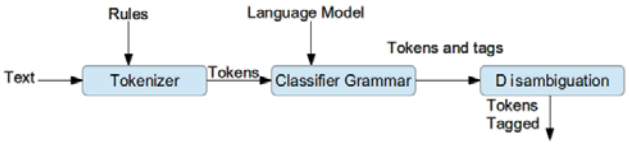


Fig. 1. Tagger morphosyntatic: overview

After the text is tokenized, the grammatical classification phase begins. For each token, the classifier looks for the possible grammar classes in the lexicon. If the token is not found in the lexicon, then the tagger uses specific procedures aimed to find a classification. When there is ambiguity, the disambiguation uses context information to solve this problem.

The language model used by the tagger can be based on rules, cases, or decision trees; in this case the tagger is called symbolic or linguistic. The model-based representation can use Hidden Markov Models, probabilistic decision trees or statistical distribution; in this case, the tagger is called statistical or probabilistic.

In this paper, we use Tree Tagger [12], a tagger based on probabilistic decision trees obtained from annotated corpora. In our experiments, it got better results when compared with MXPOST [13] and QTAG [14]. Futhermore it can easily be adapted for other languages.

C. Stemming

Stemming is the process of reducing inflected words to their stem, such as "*confirma*, *confirmar*, *confirmação*". These words have the same stem (*confirm*), but belong to different grammatical classes, noun, verb, and noun respectively. The stemming process is used when we want to group words with different spellings and grammatical categories, but related to the same concept.

In this paper, we use the Orengo and Huick algorithm [15], developed exclusively for the Portuguese language.

III. Related Work

In literature, there are many researches that were conducted to automate generation of user interfaces and class diagrams. In order to generate user interfaces automatically, A. M. Rosado [16] uses Object Constraint Language (OCL), which is a declarative language to describe rules that apply to UML models to add preciseness and semantic richness both to the domain and use case. Through the UML class diagram, a

simple user interface is generated. To generate a complete user interface, the author uses OCL constraints and a use case model.

M. Elkoutbi et al. [17] make use of UML scenarios to create a semi-automatic user interface. These scenarios are acquired in the form of collaboration diagrams, as defined by the UML, and are enriched with user interface information. Afterwards, the diagrams are transformed into specifications of UML state diagrams of user interface objects involved. From the set of specifications, the user interfaces are generated. In this work, based on the feedback given by the user, the collaboration diagram and the interface can be iteratively refined, and the overall process is a specification consisting of the state diagrams of all the objects involved, along with the user interface generated and refined.

Ren [18] makes a research on analysis of interaction process between actors and use cases. This work makes use of the pattern Model View Controller (MVC) to generate an interaction model. Moreover, he discusses automatic generation of user interfaces based on use cases. The authors focus on analysis of the interaction process between actors and the system in use cases, and build an interaction process model and an interface hierarchical structure model in the end.

W. C da Silva [10] presents a tool to automatically generate class diagrams from requirements models based on the Portuguese language. The author has used NLP techniques to perform a preprocessing to remove all accents from the words, and separate all punctuation marks; then the simple text is put on the input format used by the tagger tool. The authors have also developed a tool which makes use of textual patterns, but the textual patterns are in fact quite simple, since the tool generates a large number of false positives and false negatives.

A. Fantechi et al. [7] discuss the use of methods based on a linguistic approach to analyze functional requirements described by means of a textual natural language (use case descriptions). The authors use lexical analysis of requirements to identify occurrences of certain patterns and generate class diagrams.

The goal of our work is to generate GUI prototypes, class diagrams, and use case diagrams from use case descriptions written in the Portuguese language and that follow the Praxis process format, using NLP techniques. Our approach differs from others because our starting point are the textual use cases of a SRS and these narratives have been written naturally in the Portuguese language. Finally, researches to explore the possibilities to automatically generate GUI prototypes, class diagrams, and use case diagrams from natural language are highly challenging in the world. To apply NLP techniques to the Portuguese language is also another great challenge due to the complexity of the language (phrasal structure and word accentuation), but we are trying to overcome these difficulties by using textual patterns and hybrid NLP techniques.

IV. Prototype

The prototype was developed in the Java language and automates the creation of GUI prototypes, class diagrams, and use case diagrams from use case descriptions. It receives as input an use case description and after looking for some predefined textual patterns, identifies and generates GUI prototypes and diagrams. The Figure 2 shows a general overview process.

Fig. 2. GUI prototypes and diagrams from use case descriptions. Overview

The prototype has resources such as drag and drop components; insert, edit, and delete components; read PDF, TXT, and DOC formats, export diagrams in PNG, JPG, and XML Metadata Interchange (XMI) formats. The XMI format is according to XMI v2.1.1 version[2], which is an OMG standard for exchanging metadata information via Extensible Markup Language (XML). It also allows changes between the prototype and diagrams, so that a change in the prototype alters the class diagram associated with this prototype and vice versa. This is done interactively by the prototype developed.

The prototype has five integrated modules for generating GUI prototypes and diagrams. The main module is the syntactic module; it is responsible for generating GUI prototypes and diagrams. It receives tagged tokens from the tagger, interacts with an actions base and the stemming module to accomplish this task. Figure 3 shows a prototype screenshot.

Fig. 3. Prototype screenshot

A. Lexical and Tagger Module

The lexical module tokenizes the text into words and/or punctuation marks. In this module, we use the ANTLR tool (Another Tool for Language Recognition) [19], which is a generator of lexical and syntactic analyzers.

The tagger receives tokens and makes the tagging according to their grammatical class. The tagged tokens are sent to the syntactic module.

B. Syntactic Module

The syntactic module was also implemented by the ANTLR tool. It gets tagged tokens and makes the search for predefined textual patterns to identify UML elements (attributes,

[2]http://www.omg.org/spec/XMI/2.1.1/

operations, actors, ...); next, it generates GUI prototypes, class diagrams, and use case diagrams.

Table I shows the textual patterns used to identify GUI prototypes and class diagrams. Table II shows the textual patterns used to identify use case diagrams. These patterns were defined by manual analysis of several use case descriptions. The symbol ('?') means that the grammatical class may or may not occur.

TABLE I
TEXTUAL PATTERNS GUI PROTOTYPE AND CLASS DIAGRAM

Candidate	Textual Patterns
Interface Name/Class	Noun AND Preposition AND $Noun_2$
Fields / Attributes	(Conjunction OR Comma OR Colon) AND [Article]? AND Noun
Fields / Attributes	(Conjunction OR Comma OR Colon) AND (Article OR Preposition)? AND Noun AND Preposition AND Noun
Buttons / Operations	Verb OR Noun

1) GUI prototype and Class Diagram: When the syntactic module finds one candidate pattern **Interface Name / Class**, then it is checked whether the second noun ($Noun_2$) belongs to a predefined suffix base. If it does, then the word stem is recovered to eliminate words that have different degrees, numbers, and genders. This stem is stored in a data structure to eliminate duplicates. The predefined suffix base is commonly used to identify possible interface names, classes, and actors and it was implemented using a relational DB. This base has 32 suffixes and was defined according to a PhD thesis by Miriam Sayão [20].

For example, in the text "*tela de usuário*", the pattern "Noun AND Preposition AND $Noun_2$" is identified; the word "*usuário*" will be analyzed as (stem: "*usu*"; suffix: "*ário*"); because "*ário*" belongs to our predefined suffix base, then the stem "*usu*" is stored in our data structure, then it is identified an interface with the name "*tela de usuário*" and as a class with the name "*usuário*".

In order to identify attributes, we use only the predefined patterns. For example, in the text "*O Gerente preenche os dados do Usuário: nome, login, senha e grupos do usuário*". The words "*nome, login, senha e grupos*" will be defined as attributes/fields of the class or interface identified before.

Additionally, if some attribute is equal to the name of an interface or a class, a simple association is created with multiplicity one to one between the UML classes. For example, in the text "*O Gestor de Compras informa o Fornecedor*" the words "*compras*" and "*fornecedor*" were identified as classes/interfaces, and, because the word "*fornecedor*" is also identified as an attribute, then an association one to one is created between "*compra*" and "*fornecedor*". On the other hand, if the expression "***um ou mais***" appears before an attribute, then an association with multiplicity one to many is created.

In operations, we use a small base of words consisting of nouns and verbs related to a CRUD pattern (Create Read Update Delete). In the context of use case descriptions, the relevant types are Input, Output, Create, Read, Update, Delete and Include [21]. This base of words, or actions base, introduces semantic information into the analysis and has been implemented using a relational DB, but, if necessary, it could be represented as an OWL ontology. In use case descriptions, there are commonly appearing words such as "*inclusão, alteração, consulta, exclusão, impressão*", and so on. By using the actions base, it is possible to suggest the words "*incluir, editar, pesquisar, excluir e imprimir*", respectively. This base is useful because we can get quite accurately the desired action by the end user in the description.

After identifying the possible interface names (fields and buttons), and classes (attributes and operations), we store this information in a data structure. From this structure, it is possible to generate GUI prototypes and class diagrams. Figure 4(a) shows an GUI prototype and 4(b) shows a class diagram generated from the partial description presented in Figure 5.

TABLE II
TEXTUAL PATTERNS USE CASE DIAGRAM

Candidate	Textual Patterns
Input Actor	$Noun_1$ AND Verb
Input Actor	$Noun_1$ AND Preposition AND Noun
Output Actor	$Noun_1$ AND Noun
Use Case	$Noun_1$ AND Preposition AND Noun

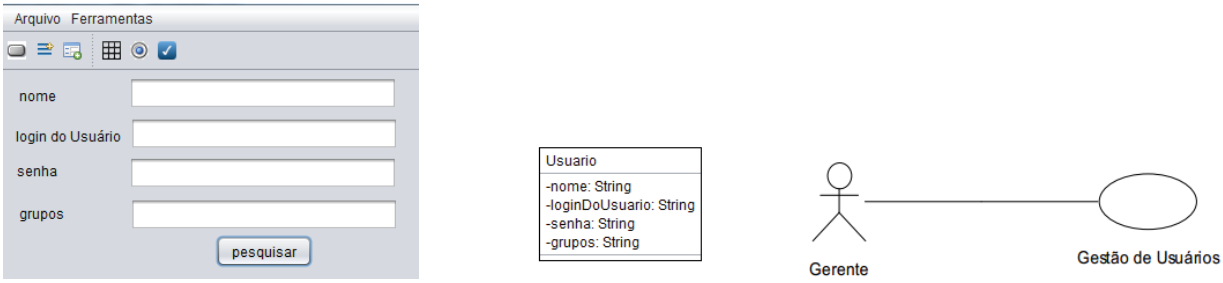
2) Use Case Diagram: In the process of use case diagrams generation, we use the predefined suffix base to identify the actors and make a distinction between actors and use cases of the use case descriptions. When a pattern is found, it is verified if the first noun ($Noun_1$), belongs to the predefined suffix base; if so, this word will be stored in a structure data.

For example, in the text "*Gestor de Compras*", the pattern "$Noun_1$ AND Preposition AND Noun" is identified; the word "*gestor*" will be analyzed as (stem: "*gest*"; suffix: "*or*"); since the word "*or*" belongs to our predefined suffix base, the stem "*gest*" is stored in our data structure. To improve the processing, some grammatical classes that are not used such as article, pronoun, adjective, adverb, conjunction, and numeral were ignored. Words of these grammatical classes are considered stop words and, therefore, they will not be processed by the syntactic module.

In order to identify the use cases, we use only the predefined textual patterns. The relationships between use cases and other use cases, such as extend and include, are identified by the words "*<ponto de extensão>*" and "*<ponto de inclusão>*" respectively, which is a standard in written Praxis (e.g., *<ponto de extensão> Emissão de Nota*). The associations between actors and use cases are identified by the prototype easily, because they are always in the same detailed use case.

After identifying the actors and use cases, we store this information in a data structure. From this structure, it is possible to generate use case diagrams. Figure 5 shows a partial description of the use case "User Management" and Figure 4(c) shows an use case diagram generated from the partial description.

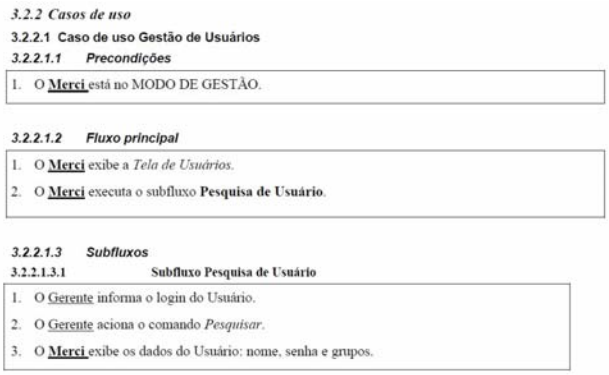
In this paper, we use the ANTLR tool, which is a language tool that provides a framework for constructing recognizers,



(a) GUI prototype (b) Class diagram (c) Use case diagram

Fig. 4. GUI prototype and diagrams generated by the prototype

interpreters, compilers, and translators from grammatical descriptions containing actions in a variety of target languages. The grammatical rules of the parser are described by a context-free grammar in EBNF notation (Extended Backus Naur Form).



3.2.2 Casos de uso

3.2.2.1 Caso de uso Gestão de Usuários

3.2.2.1.1 Precondições

1. O **Merci** está no MODO DE GESTÃO.

3.2.2.1.2 Fluxo principal

1. O **Merci** exibe a *Tela de Usuários*.
2. O **Merci** executa o subfluxo **Pesquisa de Usuário**.

3.2.2.1.3 Subfluxos

3.2.2.1.3.1 Subfluxo Pesquisa de Usuário

1. O Gerente informa o login do Usuário.
2. O Gerente aciona o comando *Pesquisar*.
3. O **Merci** exibe os dados do Usuário: nome, senha e grupos.

Fig. 5. Partial description of the use case "user management"

V. Preliminary Evaluation

As a preliminary evaluation of the prototype we use five SRS: (1) a generic system sales; (2) a system of images manipulation using Kinect; (3) a system dashboard; (4) a routes management system; and (5) an electronic document management system.

All use case descriptions were transcribed for the prototype. Figure 6 shows the quantity of GUI prototypes, entity class diagrams, and use case diagrams contained in SRS.

SRS	GUI Prototypes	Fields	Buttons	Classes	Attributes	Use cases	Actors
1	11	72	46	7	28	10	6
2	4	14	16	2	12	4	2
3	5	5	6	4	5	6	2
4	4	6	20	3	10	6	1
5	2	7	7	2	5	4	3

Fig. 6. Quantity GUI prototypes, classes, and use cases

All SRS are real systems developed in the Laboratory of Software Engineering and Industrial Informatics (EaSII), except the first SRS, which is a supporting material in many courses in Software Engineering from Brazilian Universities.

Figure 7 shows the accuracy of the prototype in identifying GUI prototypes, class diagrams and use case diagrams.

SRS	GUI Prototypes	Fields	Buttons	Classes	Attributes	Use cases	Actors
1	11	**67**	**40**	7	28	10	6
2	4	**8**	16	2	**4**	4	2
3	5	5	6	4	5	6	2
4	**3**	6	**17**	3	10	6	1
5	**3**	7	7	2	5	4	3

Fig. 7. Accuracy of the prototype. GUI prototypes, classes, and use cases

The fields in highlight (bold) show the elements that were not identified by the prototype, i.e., those that were identified as false negatives, except for the last row and second column which signals one GUI prototype that was identified, i.e., it was identified as a false positive.

Note that our prototype identified all classes, actors, and use cases. Regarding GUI prototypes, our prototype has not identified five fields in (1) SRS and six fields in (2) SRS. The buttons which were not identified did not belong to the CRUD pattern. Figure 4(a) shows a GUI prototype generated by the prototype.

Regarding class diagrams, the prototype correctly identified all classes. It is noteworthy that the prototype identified all simple associations contained in the use case descriptions. Certainly, a future work is to identify other types of relationships between a class and other classes, such as generalization, composition, and so on. Figure 8 shows a class diagram generated by the prototype.

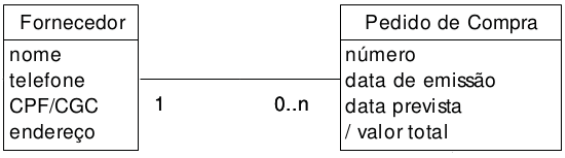


Fig. 8. Class diagram generated by the prototype

In use case diagram, the prototype correctly identified all actors and use cases contained in the use case descriptions. Moreover, it also correctly identified all relationships between actors and use cases, and between use cases and other use cases. Figure 9 shows an use case diagram generated by the prototype.

Fig. 9. Use case diagram generated by the prototype

A simple comparison with the five SRS was made to analyze which elements should be identified by the prototype. Although preliminary, our results are very promising. The prototype has identified all classes, actors, use cases, relationships between actors and use cases, relationships between use cases and other use cases, and associations between a class and other classes. The GUI prototype that was not identified was not in agreement with the textual pattern.

It is noteworthy that, through textual patterns, the prototype has not identified only five attributes (fields) in (1) SRS and six attributes (fields) in (2) SRS. These numbers show good results of false negatives and false positives, i.e., the prototype does not generate many false positives (dirt) and correctly identifies many elements.

VI. Conclusions and Future Work

In this paper, we show a prototype to automatically identify GUI prototypes, class diagrams, and use case diagrams from use case descriptions written in the Portuguese language and that follow the Praxis process format using hybrid NLP techniques.

Some Tests were performed with the SRS of real systems developed in the Laboratory of Engineering and Industrial Computer Sofware (EaSII).

Through a preliminary evaluation, our prototype has identified all classes, actors, use cases, relationships between actors and use cases, relationships between use cases and other use cases, and associations between a class and other classes. In our tests, it only did not identify five attributes (fields) in (1) SRS and six fields in (2) SRS. The operations (buttons) which were not identified did not belong to the CRUD pattern. Nonetheless, results can be considered very promising, because the numbers show good results of false positives and false negatives.

Other evaluations were conducted using other requirements specifications using the same textual patterns, but the results were not satisfactory because the specifications were not in accordance to IEEE STD 830-1998. The specifications did not describe the attributes associated with the classes, thereby, the prototype did not identify any attribute. But the prototype identified all GUI prototypes, actors, use cases, and classes.

The prototype is able to visualize the generated diagrams and GUI prototypes, as well as drag and drop components, insert, delete and edit components, and export diagrams in XMI file according to XMI v2.1.1 version.

As current/future work, we are developing: (i) exploration of the use of other semantic relations in the text (anaphora, synonyms ...); (ii) incorporation of statistical data to identify the elements as collocations; (iii) improvement of the prototype to identify more relationships between a class and other classes; and (iv) an experiment to evaluate the prototype in more detail.

References

[1] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*, ser. The Addison-Wesley object technology series. Pearson Education, 1999.

[2] A. Cockburn, *Writing effective use cases*, ser. Agile software development series. Addison-Wesley, 2001.

[3] "IEEE recommended practice for software requirements specifications," *IEEE Std 830-1998*, pp. 1–40, 1998.

[4] R. T. Anchieta, R. F. de Sousa, and R. S. Moura, "Identifying of user interface elements from use case descriptions," in *Informatica (CLEI), 2012 XXXVIII Conferencia Latinoamericana En*, oct. 2012, pp. 1 –6.

[5] A. Kao and S. R. Poteet, *Natural Language Processing and Text Mining*. Springer, Berlin, 2007.

[6] M. Luisa, F. Mariangela, and I. Pierluigi, "Market research for requirements analysis using linguistic tools," *Requir. Eng.*, vol. 9, no. 1, pp. 40–56, Feb. 2004.

[7] A. Fantechi, S. Gnesi, G. Lami, and A. Maccari, "Application of linguistic techniques for use case analysis," in *Requirements Engineering, 2002. Proceedings. IEEE Joint International Conference on*, 2002, pp. 157 – 164.

[8] G. Fliedl, C. Kop, H. C. Mayr, A. Salbrechter, J. Vöhringer, G. Weber, and C. Winkler, "Deriving static and dynamic concepts from software requirements using sophisticated tagging," *Data Knowl. Eng.*, vol. 61, no. 3, pp. 433–448, Jun. 2007.

[9] V. Ambriola and V. Gervasi, "On the systematic analysis of natural language requirements with circe," *Automated Software Engg.*, vol. 13, no. 1, pp. 107–167, Jan. 2006.

[10] W. C. da Silva and L. E. G. Martins, "Paradigma: Uma ferramenta de apoio à elicitação e modelagem de requisitos baseada em processamento de linguagem natural," in *Anais do WER08 - Workshop em Engenharia de Requisitos, Barcelona, Catalonia, Spain, September 12-13, 2008*, C. Quer, J. P. Carvallo, and L. F. da Silva, Eds., 2008.

[11] R. Gaizauskas and H. M. Harmain, "Cm-builder: An automated nl-based case tool," in *Proceedings of the 15th IEEE international conference on Automated software engineering*, ser. ASE '00. Washington, DC, USA: IEEE Computer Society, 2000, pp. 45–.

[12] H. Schmid, "Probabilistic part-of-speech tagging using decision trees," 1994.

[13] A. Ratnaparkhi, "A maximum entropy part-of-speech tagger," in *Proceedings of the Empirical Methods in Natural Language Processing Conference*, University of Pennsylvania, 1996.

[14] S. E. Lee and S. S. Han, "Qtag: introducing the qualitative tagging system," in *HT '07: Proceedings of the 18th conference on Hypertext and hypermedia*. New York, NY, USA: ACM, 2007, pp. 35–36.

[15] V. M. Orengo and C. Huyck, "A Stemming Algorithm for Portuguese Language," in *Proc. of Eigth Symposium on String Processing and Information Retrieval (SPIRE 2001) - Chile*, 2001, pp. 186–193.

[16] A. M. R. da Cruz and J. P. Faria, *Automatic Generation of User Interface Models and Prototypes from Domain and Use Case Models*, rita mátrai ed., ser. User Interfaces. Intech, May 2010.

[17] M. Elkoutbi, I. Khriss, and R. K. Keller, "Automated prototyping of user interfaces based on uml scenarios," *Automated Software Engg.*, vol. 13, no. 1, pp. 5–40, Jan. 2006.

[18] X. Ren and C. Wei, "Research on the interaction process in use case for automatic generation of user interface prototype," in *Computer and Electrical Engineering, 2008. ICCEE 2008. International Conference on*, dec. 2008, pp. 721 –725.

[19] T. Parr, *The Definitive ANTLR Reference: Building Domain-Specific Languages*, 1st ed., ser. Pragmatic Programmers. Pragmatic Bookshelf, May 2007.

[20] S. Miriam, "Verificação e validação em requisitos: Processamento da linguagem natural e agentes," Tese de Doutorado, Pontifícia Universidade Católica do Rio de Janeiro, PUC, 2007.

[21] A. Sinha, M. Kaplan, A. Paradkar, and C. Williams, "Requirements modeling and validation using bi-layer use case descriptions," in *Model Driven Engineering Languages and Systems*, ser. Lecture Notes in Computer Science, K. Czarnecki, I. Ober, J.-M. Bruel, A. Uhl, and M. Völter, Eds. Springer Berlin Heidelberg, 2008, vol. 5301, pp. 97–112.

A fuzzy based approach for requirements prioritization in goal oriented requirements elicitation process

(A step towards extension of AGORA method)

Mohd Sadiq and S K Jain
Computer Engineering Department,
National Institute of Technology, Kurukshetra-136119, Haryana, India.
E-mail: sadiq.jmi@gmail.com, skj_nith@yahoo.com

***Abstract* -- Requirements elicitation is the first sub-process of requirements engineering (RE) and involves group decision making approaches for the selection and prioritization of requirements. Prioritizing requirements means to determine the implementation order of the requirements as well as the order of importance to some stakeholders or class of stakeholders along one or more dimensions e.g., preference, business value, cost of implementation etc. Based on our literature review, we identify that existing goal oriented requirements elicitation processes do not support to prioritize the requirements when the stakeholders' opinion are often vague and contain ambiguity and multiple meaning. Therefore, to address this issue, we present a fuzzy based approach for requirements prioritization in goal oriented requirements elicitation process by combining α-level weighted F-preference relation in group decision making process and binary sort tree method to get the prioritized list of requirements. Finally, the utilization of the proposed approach is demonstrated with the help of an example.**

Keywords- Requirements elicitation, Requirements prioritization, Decision making process, and Fuzzy set theory.

I. Introduction

Prioritizing requirements means to determine the implementation order of the requirements as well as the order of importance to some stakeholders or class of stakeholders along one or more dimensions e.g., preference, business value, cost of implementation etc. [4]. In early phase of requirements engineering (RE), requirements prioritization depends on specified requirements and on the prediction of benefit and cost of the individual requirements [6]. We visualize RE as a process that includes five sub-processes namely requirements elicitation, requirements modeling, requirements analysis, requirements verification & validation, and requirements management [20]. Out of these sub-processes, requirements elicitation plays an important role since it has cascading effect on other sub-processes. Requirements prioritization [4] is an essential activity of requirements elicitation because it provides the following benefits to the project: (i) It improves customer satisfaction by increasing the likelihood that the customer's most important requirements are delivered first and (ii) It enables the project manager and customers to modify the project schedule to deal with the project realities of limited resources and fixed deadlines. For this reason, we mainly focus on the requirements prioritization in this work.

Several researchers [11, 15, 27] advocate the use of fuzzy logic to deal with problems related to the prioritization and decision making. Lai et al. [11] proposed a fuzzy based method to rank the customer requirements in a competitive environment. Zhu et al. [27] proposed a fuzzy qualitative and quantitative softgoal interdependency graphs model for non-functional requirements correlation analysis in trustworthy software. In a similar study, owing to the vague concepts frequently represented in decision making environments, Lin et al. [15] proposed a fuzzy based decision making procedure for data warehouse system selection.

The contribution of this paper is to extend the Attributed Goal-Oriented Requirements Analysis (AGORA) method [7] by applying the fuzzy based approach and binary sort tree method for the prioritization of requirements in goal oriented requirements elicitation process.

This paper is organized as follows: Section 2 presents the literature review. In section 3, basic concepts of fuzzy sets, linguistic variable, fuzzy triangular numbers, and fuzzy preference relation are reviewed. In section 4, we present fuzzy based approach for requirements prioritization when multi-criteria decision making approach is used in fuzzy environment. An example is provided in section 5 to show how the proposed approach works under fuzzy group decision making. Finally, conclusions are drawn in section 6.

II. Literature Review

In Goal Oriented Requirements Engineering (GORE) literature [2, 3, 4, 7, 8, 10, 12, 17, 22, 24, 28], there is no fuzzy based decision making process for the prioritization of requirements in requirements elicitation process. Therefore, to address this issue, we present a fuzzy based approach for the prioritization of requirements when multi-criteria decision making approach is used in fuzzy environments and multiple stakeholders participate in requirements analysis method.

Based on our literature review of requirements prioritization techniques we identify the following requirements prioritization methods: Numerical assignment [9], Cost benefit analysis [6] , TOPSIS [5], Cumulative voting [6, 9], Priority groups [6, 9], Top-10 requirements [6,9], Multi-attribute utility theory [6, 9], Weighting method [6, 9], Analytic hierarchy process (AHP) [19], Hierarchy AHP, Minimal Spanning Tree [1], Bubble sort [1], Binary search tree [1], and Thakurta's framework [23] for prioritization of quality requirements for inclusion in a software project . These methods are based on the concepts of accurate measure and crisp evaluation.

III. Fuzzy Set Theory

The fuzzy set, originally proposed by Zadeh [25, 26] in 1965, is defined as follows: In a universe of discourse U_x, a fuzzy subset A of U_x is characterized by a membership function

$f_A(x)$, where $f_A: U_x \rightarrow [0, 1]$ and the membership function associates with each member of x of U_x a number of $f_A(x)$ in the interval [0,1], representing the grade of membership of x in A. Linguistic variables are variables whose values are words or sentences in a natural or artificial language [26]. For example, *poor* is a linguistic variable if its values are assumed to be the fuzzy variables labelled *very poor, poor, fair, good,* and *very good*; rather than the numbers 0,1,2,3 etc. There are several formats of fuzzy numbers, such as Triangular, Trapezoidal, Gaussian, or Sigmoid that can be used in decision making process. In our method, we have used triangular fuzzy numbers because Li's [14] α-level weighted F-preference relation is based on triangular fuzzy numbers; and it can be defined as follows:

Let ***R*** is the real line, which is viewed as a universal set of all fuzzy sub-sets. A triangular fuzzy number A is normal, convex fuzzy subset of ***R***, with a piece wise linear relationship function μ_A, defined by:

$$\mu_A(x)= \begin{cases} \frac{(x-a)}{(b-a)}, a \le x \le b, \\ \frac{(c-x)}{(c-b)} b \le x \le c, \\ 0, \quad otherwise \end{cases} \qquad (1)$$

The fuzzy numbers can be denoted by the triplet of (a, b, c).

A fuzzy preference relation P on **R** is a fuzzy subset of **R** x **R** with membership function $f_P(A, B), \forall A, B \subseteq \boldsymbol{R}$, where $f_P(A, B)$ represents the degree of preference of A over B.

1. P is reciprocal iff $f_P(A, B) = 1 - f_P(B, A), \ \forall A, B \subseteq \boldsymbol{R}$ [14].
2. P is transitive iff $f_P(A, B) \geq 1/2$ and $f_P(B, C) \geq 1/2 \rightarrow f_P(A, C) \geq 1/2, \forall A, B, C \subseteq \boldsymbol{R}$ [14].
3. P is a fuzzy total ordering iff P is reciprocal, transitive, and comparable [13].

IV. A Fuzzy based Approach for Requirements Prioritization

The section presents a fuzzy based approach for requirements prioritization by combining: (i) α-level weighted F-preference relation proposed by Li [14] when multi-criteria decision making approach is used in fuzzy environment and multiple stakeholders participate in requirements analysis method and (ii) binary sort tree method i.e. in-order tree traversal of binary search tree [1]. The proposed method is presented simply in the following:

Step 1: Identify the primary stakeholders and secondary stakeholders. Primary stakeholders include those who are central to any project initiative, i.e., beneficiaries, financial, politicians, sponsors, and decision maker. Secondary stakeholders include developers, experts, operators etc.

Step 2: Identify the high level objective of the primary stakeholders and suggest the different configuration of the system on the basis of the cost, operability, reliability, performance etc.

Step 3: Detect the confliction among QRs, if any.

Step 4: Create the decision matrix with the help of AHP [18, 19, 21]; and select the best configuration, say G1, according to the need of primary stakeholder.

Step 5: Decompose the selected configuration, say G1, into sub goals and construct AND/OR graph. In AND decomposition, if all of the sub goals are achieved, their parent goals can be achieved or satisfied. On the other hand side, in OR decomposition, the achievement of at least one-sub goals leads to the achievement of its parent goal. Let the decomposed goals are represented by $g_1, g_2, g_3, g_4, \text{---------}g_h$.

Step 6: Identify the functional requirements from $g_1, g_2, g_3, g_4, \text{----------}g_h$.

Step 7: Group the functional requirements on the basis of implementation order (IMO) and order of importance (OI) of requirements. For example, $G_{IMO} = \{FR_1, FR_2, FR_3, \text{---}FR_m)$ and $G_{OI} = \{FR_4, FR_5, FR_6, \text{-------}FR_n)$ represents the groups of functional requirements on the basis of implementation order (IMO) and order of importance (OI) of requirements respectively.

Step 8: Let "dm" decision makers are participating during requirements analysis to prioritize the requirements on the basis of "C" decision criteria. Here decision criteria are the quality requirements.

Step 9: Collect the experts' fuzzy assessments and express their opinions on the importance of each requirements.

Step 10: Aggregate fuzzy performance rating through all decision maker by means of extended addition and scalar multiplication to form a comprehensive performance matrix P, in which performance rating $p_{ij} = (1/n) \odot (p^1_{ij} \oplus p^2_{ij} \oplus, \ldots\ldots\ldots, \oplus p^n_{ij})$ is a triangular fuzzy number of the form:
$(p_{1ij}, p_{2ij}, p_{3ij}) = (\frac{1}{n}\sum_{k=1}^{n} p^k_{1ij}, \frac{1}{n}\sum_{k=1}^{n} p^k_{2ij}, \frac{1}{n}\sum_{k=1}^{n} p^k_{3ij})$.

Now calculate the fuzzy weight through all decision makers by means of extended addition and scalar multiplication to form a comprehensive weight vector W [14].

Step 11: Aggregate fuzzy ratings with fuzzy weights by means of extended multiplication to form a weighted, comprehensive decision matrix D, in which $d_{ij} = p_{ij} \odot w_j$ is a fuzzy number with parabolic membership functions in the form of [14]:
$(\lambda_{1ij}, \lambda_{2ij}, \lambda_{3ij} / d_{ij} / \Delta_{1ij}, \Delta_{2ij}, \Delta_{3ij})$; where

$\lambda_{1ij} = (w_{2j}\text{-}w_{1j})(p_{2ij}\text{-}p_{1ij})$; $\lambda_{2ij} = w_{1j}(p_{2ij}\text{-}p_{1ij}) + p_{1ij}(w_{2j}\text{-}w_{1j})$,

$\lambda_{3ij} = w_{1j}p_{1ij}$; $\Delta_{1ij} = (w_{3j}\text{-}w_{2j})(p_{3ij}\text{-}p_{2ij})$; $\Delta_{2ij} = w_{3j}(p_{3ij}\text{-}p_{2ij}) + p_{3ij}(w_{3j}\text{-}w_{2j})$; $\Delta_{3ij} = w_{3ij}p_{3ij;}$ and $d_{ij} = w_{2j}p_{2ij}$

Step 12: Define each alternative (sub-goals) as a fuzzy number A_i, i=1, 2 ...m by means of extended addition and scalar multiplication through the criteria, i.e.,

$A_i = 1/C \odot (d_{i1} \oplus d_{i2} \oplus, \ldots\ldots, \oplus d_{iC})$ with parabolic membership function in the form of [14]:

$(\lambda_{1i}, \lambda_{2i}, \lambda_{3i} / \underset{\circ}{A}_i / \Delta_{1i}, \Delta_{2i}, \Delta_{3i})$; where

$\lambda_{Ii} = \frac{1}{C}\sum_{j=1}^{C} \lambda_{Iij}$, I= 1,2,3; $\Delta_{Ii} = \frac{1}{C}\sum_{j=1}^{C} \Delta_{Iij}$, I=1, 2, 3, and

$\underset{\circ}{A}_i = \frac{1}{C}\sum_{j=1}^{C} d_{ij.}$

Step 13: Define extended average $\underset{\circ}{A}$ by means of extended addition and scalar multiplication through all alternatives (sub-goals), i.e., $A_{1i} = 1/h \odot (g_1 \oplus g_2 \oplus, \ldots, \oplus g_h)$ with parabolic membership function in the form of [14]:

$(\lambda_1, \lambda_2, \lambda_3 / \hat{A} / \Delta_1, \Delta_2, \Delta_3)$; where

$\lambda_I = \frac{1}{h}\sum_{j=1}^{h} \lambda_{Ii}$, I=1, 2, 3; $\Delta_I = \frac{1}{h}\sum_{j=1}^{h} \Delta_{Ii}$, I=1, 2, 3, and

$\hat{A} = \frac{1}{h}\sum_{j=1}^{h} \underset{\circ}{A}_i$

Step 14: Define the extended difference, $A_i \ominus \hat{A}$, for each $A_i \in R$, with parabolic membership function in the form of [14]:

$((\lambda_{1i} - \Delta_1), (\lambda_{2i} + \Delta_2), (\lambda_{3i} - \Delta_3) / \hat{A}_i - \hat{A}/ \quad (\Delta_{1i} - \lambda_1), (-\Delta_{2i} - \lambda_2), (\Delta_{3i} - \lambda_3))$.

Step 15: Calculate the ranking values rv_i for each alternative (requirements) A_i by means of F-preference relation R [14]:

if $(\lambda_{3i} - \Delta_3) < 0$, $(\Delta_{3i} - \lambda_3) \geq 0$, $\hat{A}_i \geq \hat{A}$;

then $rv_i = \mu_R (A_i \ominus \hat{A}, 0) = \prod^+ / (\prod^+ + \prod^-)$; else if $(\lambda_{3i} - \Delta_3) \leq 0$, $(\Delta_{3i} - \lambda_3) > 0$, $\hat{A}_i \leq \hat{A}$; then $rv_i = \mu_R (A_i \ominus \hat{A}, 0) = \Psi^+ / (\Psi^+ + \Psi^-)$; else if $(\lambda_{3i} - \Delta_3) = 0$, $(\Delta_{3i} - \lambda_3) = 0$, $\hat{A}_i = \hat{A}$ then $rv_i = \mu_R (A_i \ominus \hat{A}, 0) = 0.5$; else if $(\lambda_{3i} - \Delta_3) \geq 0$, $(\Delta_{3i} - \lambda_3) > 0$, $\hat{A}_i \geq \hat{A}$ then $rv_i = \mu_R (A_i \ominus \hat{A}, 0) = 1$; else if $(\lambda_{3i} - \Delta_3) < 0$, $(\Delta_{3i} - \lambda_3) \leq 0$, $\hat{A}_i \leq \hat{A}$ then $rv_i = \mu_R (A_i \ominus \hat{A}, 0) = 0$. Where

$\prod^+ = -[1/4(\Delta_{1i} - \lambda_1) - 1/3 (\Delta_{2i} + \lambda_2) + 1/2 (\Delta_{3i} - \lambda_3)] + [1/4 (\lambda_{1i} - \Delta_1) (1-\Upsilon^4) + 1/3 (\lambda_{2i} + \Delta_2) (1- \Upsilon^3) + 1/2 (\lambda_{3i} - \Delta_3) (1-\Upsilon^2)]$;

$\prod^- = (1/4 (\lambda_{1i} - \Delta_1) \Upsilon^4 + 1/3 (\lambda_{2i} + \Delta_2) \Upsilon^3 + 1/2 (\lambda_{3i} - \Delta_3) \Upsilon^2)]$;

$\Upsilon = [- (\lambda_{2i} + \Delta_2) + sqrt \ \{(\lambda_{2i} + \Delta_2)^2 - 4(\lambda_{1i} - \Delta_1) (\lambda_{3i} - \Delta_3)\}]/ [2(\lambda_{1i} - \Delta_1)]$;

$\Psi^+ = 1/4 (\Delta_{1i} - \lambda_1) \ddot{\Upsilon}^4 + 1/3 (-\Delta_{2i} - \lambda_2) \ddot{\Upsilon}^3 + 1/2 (\Delta_{3i} - \lambda_3) \ddot{\Upsilon}^2$;

$\Psi^- = - [1/4 (\lambda_{1i} - \Delta_1) + 1/3 (\lambda_{2i} + \Delta_2) + 1/2 (\lambda_{3i} - \Delta_3)] - [1/4 (\Delta_{1i} - \lambda_1) (1-\ddot{\Upsilon}^4) - 1/3 (\Delta_{2i} + \lambda_2) (1-\ddot{\Upsilon}^3) + 1/2 (\Delta_{3i} - \lambda_3) (1-\ddot{\Upsilon}^2)]$;

$\ddot{\Upsilon} = [(\Delta_{2i} + \lambda_2) - sqrt \ \{(-\Delta_{2i} - \lambda_2)^2 - 4 (\Delta_{1i} - \lambda_1) (\Delta_{3i} - \lambda_3)\} / [2 (\Delta_{1i} - \lambda_1)]$; where "*sqrt*" is a square root function?

Step 16: Construct the binary search tree of rv_1, rv_2, rv_3, ----------$r_{i,,}$ i.e., BSTRV.

Step 17: Apply in-order tree traversal technique on BSTRV and as a result, we will get the prioritized list of requirements.

V. An Example

To illustrate the proposed approach, we assume that 10 stakeholders are participating in requirements elicitation and analysis, i.e., S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8, S_9, and S_{10} (Step1). S_1 is the primary stakeholder and its objective is to get the benefit from the system implementation. S_1 is also the beneficiary and sponsor of the system, who wants to develop a system. S_2 is the secondary stakeholder and is responsible for the identification of functional requirements (FR) and to deal with the decisions that are related to the organization goals and objective. The objective of stakeholder S_3 is to elicit the non functional requirements (NFR)/quality requirements (QR) and also to take the decisions related to the organizations goals and objective. Stakeholder S_4 is the in charge of the system throughout all the life cycle phases. Stakeholder S_5 is requirements analyst and developer; and is directly involved in information system development. S_6 is operator and deals with assuring effectiveness when performing operation within the organization. S_7 are both primary and secondary stakeholders and will act as an expert and the decision maker. S_8 is the regulator and is responsible for quality, security and cost or other aspects of the system. S_9 and S_{10} are the financers; and they benefit indirectly from the system, obtaining financial rewards and are selected on the basis of geographical location.

Step 2: Let S_1 wants to develop a software system; and its purchasing choice depends on Cost (QR_1), Operability (QR_2), Reliability (QR_3), and Flexibility (QR_4). Requirements analyst i.e., S_5 identified the high level objective of S_1 and offered 3 configuration of the system, i.e. System A, System B, and System C. Configuration 1: System A is cheap and easy to operate but is not very reliable and could not easily be adapted to other users. Configuration 2: System B is somewhat more expensive and is easy to operate. It is reliable but not very adaptable. Configuration 3: system C is very expensive, not easy to operate, less reliable than System B but it includes a wide range of alternative uses. Selection of a system on the basis of different selection criteria, i.e. quality requirements (QR) is clearly a multi-criteria decision making problem. For the detection of confliction among NFR we use the catalogue proposed by Mairiza and Zowghi [16]. In our case, there is no confliction among QRs (Step 3).

Step 4: The AHP is a decision support tool which can be used to solve such types of decision making problems. After applying the AHP, we obtain the following relative value vector for QR_1, QR_2, QR_3, and QR_4.

$QR_1 = 0.232$, $QR_2 = 0.402$, $QR_3 = 0.061$, and $QR_4 = 0.305$.

These values indicate that the first priority of S_1 is operability, i.e. $QR_2 = 0.402$. The 0.305 shows that S_1 like the idea of flexibility. $S_1$is not worried about cost and are not interested in reliability. Now the decision matrix is formed on the basis of Cost (QR_1), Operability (QR_2), Reliability (QR_3), and Flexibility (QR_4) (see Table 1).

Table 1: Decision Matrix

Alternatives	**Criterion**				**Final Priority**
	QR_1	QR_2	QR_3	QR_4	
	0.232	0.402	0.061	0.305	
System A	0.751	0.480	0.077	0.066	0.392 (II)
System B	0.178	0.406	0.231	0.615	0.406 (I)
System C	0.071	0.114	0.692	0.319	0.204 (III)

Results of Table 1 indicate that S_1 will select the System B because it has high priority. System A scores 0.392, and it is slightly worse in terms of the requirements of System B. System C is well behind at 0.204 and would do rather badly at satisfying the requirements of S_1 in this illustrative case.

Table 2: Triangular Fuzzy numbers of linguistic values for each FR and for the relationship between FR and QR

Triangular fuzzy numbers of linguistic values for each FR		**Triangular fuzzy numbers of linguistic values for the relationship between FR and QR**	
Linguistic values	***Triangular fuzzy number***	***Linguistic values***	***Triangular fuzzy number***
VL (Very Low)	(0,0,0.25)	VW (Very Weak)	(2,2,4)
L (Low)	(0,0.25,0.5)	W (Weak)	(2,4,6)
M (Middle)	(0.25,0.5,0.75)	M (Medium)	(4,6,8)
H (High)	(0.5,0.75,1)	S (Strong)	(6,8,10)
VH (Very High)	(0.75,1,1)	VS (Very Strong)	(8,10,10)

Table 3: Fuzzy Importance Weight by 10 Stakeholders for Each FR

FRs	**Stakeholders**										**Fuzzy importance weight**
	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	
FR_1	VH	H	VH	H	M	L	M	VL	H	L	(0.35,0.58,0.78)
FR_2	H	L	VH	VH	H	M	L	M	M	VL	(0.33,0.55,0.75)
FR_3	M	VH	L	H	VH	H	VH	L	VL	M	(0.38,0.6,0.78)
FR_4	H	M	VH	L	VL	L	H	H	VH	H	(0.38,0.6,0.8)
FR_5	H	VL	L	VL	M	M	H	VL	L	M	(0.18,0.35,0.6)
FR_6	L	VL	H	VH	H	VH	L	M	M	M	(0.33,0.55,0.75)
FR_7	VL	L	H	L	H	VH	M	M	VL	VL	(0.23,0.4,0.63)
FR_8	L	VH	L	VL	M	VH	M	M	VL	L	(0.28,0.48,0.68)
FR_9	L	L	VL	M	H	VH	VH	H	VL	VL	(0.28,0.45,0.65)
FR_{10}	H	VH	H	L	M	M	VH	L	L	VL	(0.3,0.53,0.73)

After applying steps 5, 6 and 7, let us assume that, we identify the following set of requirements for system B: G_{IMO} = {FR_1, FR_2, FR_4, FR_7) and G_{OI} = {FR_3, FR_5, FR_6, FR_8, FR_9, FR_{10}) and there are five decision makers in a cross functional team to evaluate the relationship between functional requirements and quality requirements (step 8).

In step 9, the linguistic values of the importance weight of each functional requirements and the relationship between functional requirements and quality requirements are defined. In our example, five ranks are used and the triangular fuzzy numbers of linguistic values for each functional requirements and the relationship between functional requirements and quality requirements are listed in Table 2. Ten stakeholders were asked to evaluate the importance of each FR and the results are summarized in Table 3, where the stakeholders' opinions are quite different and the relationship between FR and QR evaluated by cross functional team with five decision makers .The fuzzy relationship between FR and QR is given in Table 4 (step 10)

Table 4: Fuzzy Relationship between FR and QR

FRs	QR_1	QR_2	QR_3
FR_1	(5.2,7.2,8.8)	(6.8,8.8,10)	(4.8,6.4,8)
FR_2	(5.2,7.2,8.8)	(4.4,6,7.6)	(4.8,6.4,8)
FR_3	(4.8,6.4,7.6)	(4.4,6,7.6)	(5.2,6.8,8.4)
FR_4	(5.6,7.6,9.2)	(4.8,6.8,8.4)	(5.6,7.6,9.2)
FR_5	(3.6,5.2,7.2)	(4,5.6,7.6)	(4,5.6,7.6)
FR_6	(6.4,8.4,9.6)	(6,8,9.2)	(4.8,6.8,8.4)
FR_7	(4,5.6,7.6)	(6,8,9.2)	(4.4,6,8)
FR_8	(5.2,7.2,9.2)	(4.4,6.4,8)	(4.8,6.8,8.8)
FR_9	(5.6,7.6,8.8)	(4.8,6.4,8)	(6,8,9.6)
FR_{10}	(4.8,6.4,8)	(6,8,9.2)	(6,8,9.2)

Step 11: A weighted comprehensive decision matrix, depicted in Table 5, can be established by applying the extended multiplication, where QF denotes the quadratic membership function.

Table 5: Quadratic Membership Functions

FRs	Criteria		
	QR_1	QR_2	QR_3
FR_1	4.176_{QF}	4.84_{QF}	3.84_{QF}
FR_2	4.176_{QF}	3.3_{QF}	3.84_{QF}
FR_3	3.712_{QF}	3.3_{QF}	4.08_{QF}
FR_4	4.408_{QF}	3.74_{QF}	4.56_{QF}
FR_5	3.02_{QF}	3.08_{QF}	3.36_{QF}
FR_6	4.87_{QF}	4.40_{QF}	4.08_{QF}
FR_7	3.25_{QF}	4.40_{QF}	3.6_{QF}
FR_8	4.18_{QF}	3.52_{QF}	4.08_{QF}
FR_9	4.408_{QF}	3.52_{QF}	4.8_{QF}
FR_{10}	3.712_{QF}	4.40_{QF}	4.8_{QF}

After applying the steps 12 and step 13, the extended average of all requirements by means of extended addition and scalar multiplication is in the form of:
3.98QF=0.406,1.78,1.79/3.98/0.306,2.86,6.54

After executing steps 14 and 15, we get the following ranking values of requirements:

FR_1=0.587, FR_2=0.50, FR_3=0.41, FR_4=0.571, FR_5=0.27, FR_6=0.625, FR_7=0.435, FR_8=0.488, FR_9=0.573, and FR_{10}=0.588.

In Step 16 and 17, we used binary sort tree (BST) method to prioritize the given set of requirements. BST method is a sorting method that builds on binary search tree and traverses the tree using IN-ORDER tree traversal. Therefore, after applying the binary sort tree method on the above set of data, we get the following prioritized list of requirements:

$FR_5 < FR_3 < FR_7 < FR_8 < F R_2 < FR_4 < FR_9 < FR_1 < FR_{10} < FR_6$

VI. Conclusion

This paper presents a fuzzy based approach for requirements prioritization when multiple stakeholders participate in requirements elicitation and analysis process. In order to strengthen the GORE techniques and to make decision when the information is fuzzy and imprecise, in proposed approach, we used α-level weighted F-preference relation in group decision making process. To simply show how proposed approach works, a numerical example is shown to illustrate the fuzzy group decision making approach in goal oriented requirements elicitation and analysis process. In our example, we assumed that there are ten requirements, three criteria for the prioritization of requirements, ten stakeholders are involved in determining the weight of each goal and five stakeholders' are participating as decision maker. On the basis of our analysis we identify that FR_6 is the most important requirements and it is the best alternative for the achievement of its parent goal, i.e., G1. This paper is the first step towards an extension of AGORA in which we used fuzzy decision making approach for the prioritization of requirement. However, the method discussed in this paper can be further exploited by considering more sub-goals/requirements and criteria as well as much larger group of stakeholders in group decision making process. The future research agenda can be listed as follows:

(a) To propose a fuzzy attributed goal oriented requirements elicitation and analysis (FAGOREA) method by using fuzzy contribution values and fuzzy preference matrix.

(b) To develop a tool to support the decision making processes of FAGOREA.

References

1. Aho A V, Hopcroft J E, and Ullman J D, "Data Structures and Algorithms", Addison-Wesley, 1983.
2. Anton A I, "Goal Based Requirements Analysis", IEEE International Requirements Engineering Conference, pp. 136-144, 1996.
3. Anwer S, Ikram N, "Goal Oriented Requirement Engineering: A Critical Study of Techniques", 12th Asia Pacific Software Engineering Conference, 2006.
4. Firesmith D, "Prioritizing Requirements", Journal of Object Technology, Vol.3, No.8, pp. 35-47, September-October, 2004.
5. Hwang C L, and Yoon K, "Multiple Attributes Decision Making Methods and Applications, Springer, Verlag.
6. Herrmann A and Daneva M, "Requirements Prioritization Based on Benefit and Cost prediction: An Agenda for Future Research", IEEE international Requirements Engineering Conference, pp.125-134, Chicago, Illinois, USA, 2008.
7. Kaiya H et al., "AGORA: Attributed Goal Oriented Requirements Analysis", Proceedings of the IEEE Joint ICRE'2002.
8. Kaiya H et al., Improving the Detection of Requirements Discordances Among Stakeholders", Requirements Engineering- Springer, pp. 289-303, 2005.
9. Karlsson J, Wohlin C, Regnell B, "An Evaluation of Methods for Prioritizing Software Requirements", Information and Software Technology, Volume 39, pp.939-947, 1998.
10. Kavakli E, "Goal Oriented Requirements Engineering: A Unifying Framework", Requirements Engineering-, Springer, 2002.
11. Lai X, Xie M, Tan K-C, and Yang B, " Ranking of Customers in a Competitive Environments", Computers and Industrial Engineering, Vol.54, pp.202-214, 2008.
12. Lamsweerde A V, "Goal-Oriented Requirements Engineering: A Guided Tour", Proceedings of 5th IEEE International Symposium on Requirements Engineering, pp.249-263, 2001.
13. Lee H-S, "On Fuzzy Preference Relation in Group Decision Making", International Journal of Computer Mathematics, Taylor and Francis, Vol.82, pp.133-140, 2005.
14. Li R J, "Fuzzy Method in Group Decision Making", Computers and Mathematics with Application-Elsevier, Vol.38, pp. 91-101, 1999.
15. Lin H Y, Hsu P Y, and Sheen G J, "A Fuzzy Based Decision Making Procedure for Data Warehouse System Selection", Expert system with Applications-Elsevier, Vol. 32, pp.939-953, 2007.
16. Mairiza D and Zowghi D, "Constructing a Catalogue of Conflicts among Non-functional Requirements", ENASE-Springer, pp.31-44, 2011.
17. Oshiro K et al., " Goal-Oriented Idea Generation Method for Requirements Elicitation", 11th IEEE International Requirements Engineering Conference, pp. 1-2, 2003
18. Sadiq M et al, "More on Elicitation of Software Requirements and Prioritization using AHP", IEEE International Conference on Data Storage and Data Engineering", pp.230-234, 2010.
19. Sadiq M, Ghafir S, Shahid M, " An Approach for Eliciting Software Requirements and its Prioritization using Analytic Hierarchy Process", IEEE International Conference on Advances in Recent Technologies in Communication and Computing, pp 799-795, 2009.
20. Sadiq M, Jain S.K., "An Insight into Requirements Engineering Processes", 3rd International Conference on Advances in Communication, Network, and Computing, LNCSIT-Springer, pp. 313-318, Chennai, February, 2012, India.
21. Sadiq M, Shahid M, "Elicitation and Prioritization of Software Requirements", International Journal of Recent Trends in Engineering, Vol. 2, No. 3, November, 2009.
22. Shibaoka M et al., "GOORE: Goal Oriented and Ontology Driven Requirements Elicitation Method", ER Workshop, LNCS, Springer-Verlag, pp. 225-234, 2007.
23. Thakurta R, "A Framework for Prioritization of Quality Requirements for Inclusion in a Software Project", Software Quality Journal, Springer, 2012.
24. Yu E S K, "Towards Modeling and Reasoning Support for Early and Late Phase of Requirements Engineering", IEEE International Conference on Requirements Engineering, 1997.
25. Zadeh L A, " The Concept of a Linguistics Variable and its Application to Approximate Reasoning(I)", Information Science, Vol.8, pp.199-249, 1975.
26. Zadeh L A, "Fuzzy Sets", Information Control, Vol.8, pp.338-353, 1965.
27. Zhu M X, Luo X-X, Chen X H, and Wu D D, "A Non-functional Requirements Tradeoff Model in Trustworthy Software" Information Science-Elsevier, Vol. 191, pp.61-75, 2012.
28. Zickert F, "Evaluation of the Goal Oriented Requirements Engineering Methods KAOS", American Conference on Information system", pp.1-9, 2010.

Integrating Functional with Non-functional Requirements Analysis In Object Oriented Modeling Tool Based on HOOMT

Jinwu Wang, Fan Zhang
Department of Computer Science
Hunan Univ. of Science & Technology
Xiangtan, China

Xiaoqing (Frank) Liu, Eric Barnes
Department of Computer Science
Missouri Univ. of Science & Technology
Rolla, US

Buqing Cao, Mingdong Tang
Department of Computer Science
Hunan Univ. of Science & Technology
Xiangtan, China

***Abstract*—Due to the rapid growth of software size and complexity, system modeling has become an increasingly important factor in software development. Most research on system modeling focuses on functional requirements and is inadequate for non-functional requirement (NFR) analysis. This paper proposes the Computer Aided Software Analysis Tool based on HOOMT (CASAT-HOOMT), which integrates functional requirement and object structure modeling with NFR modeling. The tool allows for automated decomposition for NFRs to match the decomposition of the object, process, and state diagram decompositions. CASAT-HOOMT has been fully implemented and its operation is described in this paper.**

Keywords-component: HOOMT; CASAT-HOOMT; non-functional requirements modeling; Modeling Tool.

I. INTRODUCTION AND RELATED WORK

There are many modeling methodologies used within the software industry. Chief among these are object-oriented modeling techniques, such as UML [1-3], methods presented by Booch [4] and Coad [5], OMT [6-7]. Another such technique is HOOMT [11-14], which supports the uniform decomposition of both object structures, functionalities, and behaviors. A significant amount of research and development work has been done on UML-based development methodology and environment. Many computer aided modeling tools have been developed based on UML, such as Rational Rose [8], Sybase Power Designer[9], and Microsoft Visio[10].

However, there are three problems with the above UML-based modeling tools. The first is that they lack well-defined processes and mechanisms for structured analysis of object, functionality and dynamic behavior consistently based on hierarchical decomposition, the second is lack of a unique point to begin modeling from, and the third is a lack of integration of NFRs with other object-oriented modeling elements.

This work presents a tool for creating and displaying HOOMT models which is integrated with NFR analysis techniques based on these models. In describing NFRs in terms of the object structure, functionality, and behavior of a system at various levels of detail, system architects can better describe how these NFRs describe the quality of the system.

II. HIGH-ORDER OBJECT ORIENTED MODELING WITH NFRS IN CASAT-HOOMT

CASAT-HOOMT, was developed in order to apply HOOMT to software design projects. The tool was implemented through the use of Windows Forms Programming and GDI+ graphics programming in Microsoft's .NET Framework. The data is stored in SQLITE database. A screenshot of CASAT-HOOMT is given in Figure 2.

Non-functional Requirements (NFRs), are constraints on the quality of a system. These include factors such as system performance, reliability, maintainability, scalability etc. CASAT-HOOMT not only maintains the advantages of HOOMT, but also supports NFR modeling and unified modeling, integrating functional requirements with NFRs. CASAT-HOOMT currently supports the existing models of HOOMT: High Order Object Model (HOOM), Hierarchical Object Information Flow Model (HOIFM), and Hierarchical State Transition Model (HSTM). Our goal is to combine these model-centric views of the system with NFR analysis in order to give users a clearer understanding of NFRs in various contexts, and within varying levels of detail.

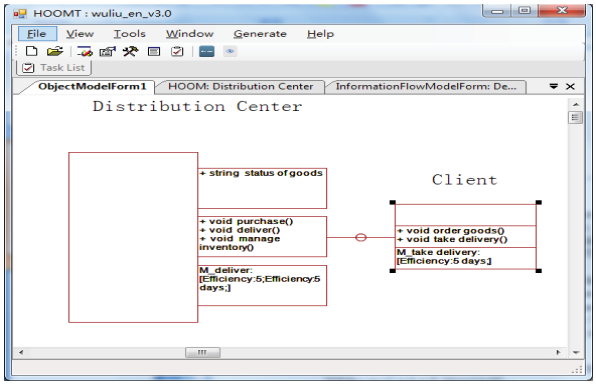


Figure 1: HOOM Object Diagram in CASTAT-HOOMT

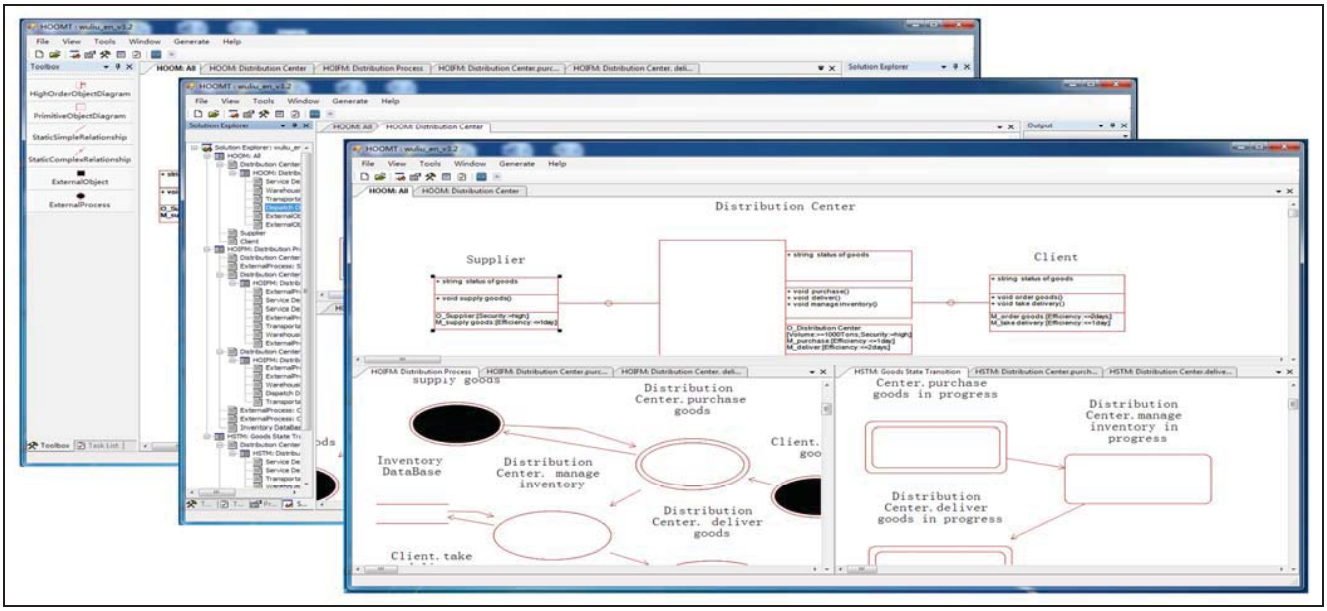


Figure 2. Screenshot of CASAT-HOOMT Enviornment

HOOM is an object model which assists in the analysis of objects with complex structures. It organizes objects hierarchically based on their abstraction level with respect to system requirements. This allows the structural, functional, non-functional and dynamic behaviors of objects at higher abstraction levels to be analyzed based on those of objects at lower abstraction levels. In HOOMT objects are categorized into two types: high order objects and primitive objects.

HOIFM allows for the analysis of functional requirements and their relationships within the system hierarchically in terms of their abstraction levels. The information flow diagram is categorized into two types: primitive processes and high order processes further analyzed through decomposition into primitive processes.

HSTM was developed to analyze dynamic behaviors and their relationships within the system hierarchically, in terms of their abstraction levels. HSTM is a hierarchy of high order state transition diagrams and primitive state transition diagrams. Every attribute in HOOM corresponds to a state in HSTM, which has its own set of NFRs as well as the NFRs of the corresponding HOOM attribute, and transition between state diagrams must also be constrained by NFRs.

III. THE IMPLEMENTION OF NON-FUNCTIONAL REQUIREMENTS MODELING IN CASAT-HOOMT

When an NFR is added to one of the preceding mode elements, its target, factor, operator, value, unit, and chart must be specified. This is done through the interface of the NFR add-and-decompose module, an example of which is given in figure 3.

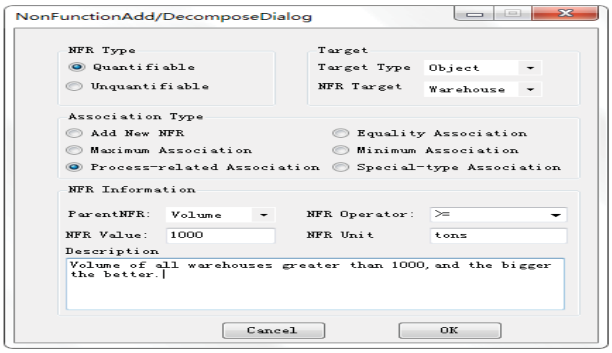


Figure 3: Interface of NFR add-and-decompose module

In CASAT-HOOMT, NFRs are expressed by a 6-tuple: <target, factor, operator, value, unit, chart>. Their descriptions are as follows:

Target: *NFRs do not exist independently from functional requirements, therefore we first identify the functionality the given NFR pertains to. In CASAT-HOOMT, target refers to either an object, method (process) ,or attribute (state).*

Factor: *Used to describe the type of the NFR, such as performance, maintainability, reliability, security, etc. In CASAT-HOOMT, we categorize factor into two types: quantifiable factors and unquantifiable factors.*

Operator: *Comparison operator. In CASAT-HOOMT, we defined four types of operators: "NoMoreThan", "NoLessThan" and "NotEqualsTo", "EqualsTo", respectively indicateds by symbols: "<=", ">=", "≠", and "=".*

Value: *The metric used to describe a particular non-functional constraint in concrete terms. Value can be*

either numeric types or enumerated types. Numeric types correspond to quantifiable NFRs and enumerated types correspond to unquantifiable NFRs.

Unit: *(Optional) Quantifiable NFRs have a unique unit (such as hours for mean time between failures), but unquantifiable NFRs have none.*

Chart: *Represents the relationship between varying performance values and the level of NFR satisfaction.*

The NFR Chart Module is shown in Figure 3. For each quantifiable NFR, an NFR-chart is selected to describe the relationship between a non-functional constraint's value and satisfaction level. In this example, satisfaction of the NFR is at 0% for all volumes from 0 to 500 tons. As volume increases, so does satisfaction, until it maxes out at 1000 tons. Additionally, the Association Type, which describes to the relation of the NFR to its parent in the decomposition process, must be chosen in order to determine how each NFR is decomposed.

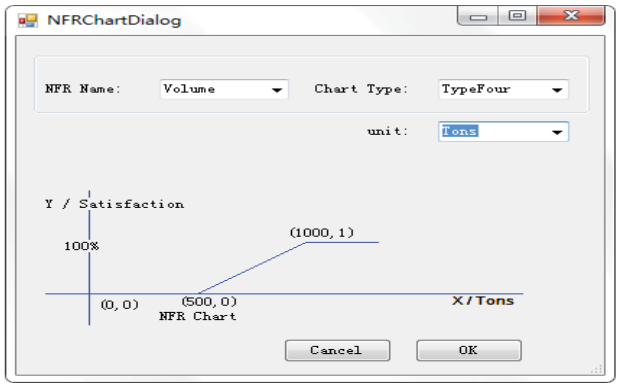


Figure 4: Interface of NFR-chart module

IV. DECOMPOSITION OF NFRS IN CASAT-HOOMT

In the decomposition process of high-order objects, the object, methods, and attributes are decomposed. This process is discussed in greater detail in previous work on HOOMT . In CASAT-HOOMT, NFRs affiliated with an object, method or attribute are decomposed in the same way as their associated element. Because decomposition of objects, behaviors, and attributes is an essential part of the HOOMT method, the decomposition of NFRs must be carried out in the same way in order to maintain compatibility with HOOMT. This allows for an intelligent and accurate analysis of NFRs and how they manifest within the system at varying levels of detail. The mode of this decomposition, however, depends on the NFRs themselves.

In order to make NFR decomposition more convenient, the decomposition relations are categorized into five basic types: Equality Association (security etc.), Maximum Association (throughput, accuracy, etc.), Minimum Association (accessibility etc.), Process-related Association (reliability, response times etc.), and Special-type Association (legibility, legislative requirements etc.). In the decomposition of NFRs, decomposition association is chosen for each NFR, unless it is a top-level NFR with no parent. This is used to check the consistency of NFRs when modeling the system according to the association of the decomposition.

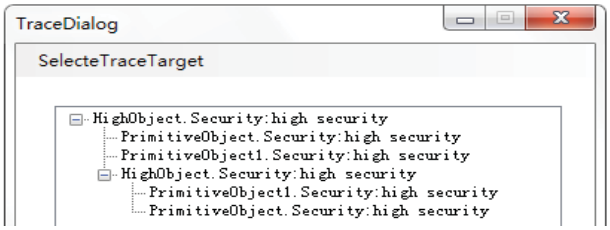


Figure 5: Interface of NFR-trace module

The definition for each association is as follows:

Equality Association: *V_f is the NFR's value for the target function If there is Equality Association between target function and its sub-function, then the non-functional requirement's value of sub-function is V_s, and the relationship between V_s and V_f is: $Vs = V_f$.*

Maximum Association *V_f is the NFR's value for the target function. if there is Maximum Associate between target function FR_f and its sub-function set $FR_{si,}$ ($i = 1, 2, \ldots n$), then the relationship between V_{si} and V_f is: $V_{si} >= V_{f,}$ ($i = 1, 2, \ldots n$).*

Minimum Association: *V_f is the NFR's value for the target function. If there is Minimum Association between target function FR_f and its sub-function set $FR_{si,}$ ($i = 1, 2, \ldots n$), then the relationship between V_{si} and V_f is: $V_{si} <= V_{f,}$ ($i = 1, 2, \ldots n$).*

Process-related* Association: V_f is the NFR's value for the target function. If there is Process-related Associate between target function FR_f and its sub-function set $FR_{si,}$ ($i = 1, 2, \ldots n$), then the relationship between V_{si} and V_f is: $V_f = f(V_{si})$, ($i = 1, 2, \ldots n$), f() is a function which is related to process control structure of functional requirement and the non-functional requirement.*

Special-type Association: *some NFR factors can't depend on the associations above, but need to depend on the definition of non-functional requirements type and domain knowledge. This includes constraints such as legibility.*

V. IMPLEMENATION OF NFR TRACE MODULE

The trace module can track the decomposition process of high-order objects, methods, attributes, and NFRs, and provide the necessary feedback for system modeling and development. In CASAT-HOOMT the Singleton Design Pattern was chosen to design the trace module. The interface of NFRs trace module is shown in Figure 5. In CASAT-HOOMT, NFRs can be traced downwards from objects, methods, or attributes to lower level components. This allows the user to analyze and describe

NFRs at varying levels of detail as needed. In this algorithm, the inputs consist of a high-order object, a method of the high-order object, and an NFR of the method. The output is a tree structure describing the NFR decomposition. Specific steps are shown as follows:

Algorithm1 The Core Algorithm of Non-Functional Requirements Trace Method

```
//Input: a High-order Object Diagram (target-Diagram)
//        a method of the Object (target-Method)
//        a NFR of the method (target-NFR)
//Output: a Trace Tree

    //define a queue to store high-order object diagrams
Queue <High-order Object Diagram> queue = new Queue < High-order Object Diagram > ();
    //define a tree to show trace result
TraceTree traceTree = new TraceTree();
    //add tree root
TreeNode node = new TreeNode(target-NFR);
Tree. addNode(node);
   //used to record the current active node
ActiveNode activeNode = null;
   //used to record the current parent method
NFR currParentMethod = null;
   //used to record the current parent nfr
NFR currParentNfr = null;
   //push the Target High-order Object into the queue
queue.Push(target Diagram);
   //execute the loop until the queue is empty
while (queue.Count > 0) do {
       // remove and return the object at the top of the queue.
   High-order Object Diagram diag = queue.Pop();
    currParentMethod = Find diag's Method which is sub-method
   of target-Method and return;
    currParentNfr = Find currParentMethod's NFR which is sub-
   nfr of target-NFR and return;
    activeNode = Find currParentNfr's treeNode in traceTree and
   return;
        //fetch the child diagram of model node
    foreach (Diagram diagram in diag.modelNode.diagramList) {
     if (diagram is High-order Object) then
        queue.Push(diagram);
     end if;
     foreach(Method method in diagram.methodList) {
      if (method .patent==currParentMethod) then
       foreach (NFR nfr in method.nfrList) {
        if (nfr.parent == currParentNfr) then
           TreeNode node = new TreeNode(nfr);
           activeNode.addNode(node);
       end if;
       }
      end if;
      }
   }
}
```

The core algorithm the NFR trace of an object is similar to the algorithm above. The difference is that NFRs list of the object instead of the NFR list of the method. For the NFR trace of attributes, "method" in the pseudo code above can be replaced by "attribute".

VI. Conclusion and Outlook

In this paper, we proposed CASAT-HOOMT, which combines functional requirements modeling with NFR analysis in an object-oriented context. CASAT-HOOMT combines tools to draw and analyze HOOM, HOIFM, and HSTM diagrams. These three models are implemented hierarchically based on abstraction levels, in order to ensure consistency among individual models. This allows for a unified decomposition of NFRs as they apply to each model type, resulting in a better understanding of how NFRs constrain the system.

Presently, model driven architecture techniques have become increasingly important in software design. Our next works will commit to improve CASAT-HOOMT, and automatically generate system code according to the CASAT-HOOMT tool.

References

[1] Object Management Group, "Unified Modeling Language: Superstructure," Formal Specification, version2.1.2, 2007.

[2] G.Spanoudakis, A. Zisman, "Discovering Services during Service-Based System Design Using UML," Proc. IEEE Transactions on Software Engineering, May-June 2010, pp. 371 – 389, doi: 10.1109/TSE.2009.88.

[3] D. Šilingas, R. Vitiutinas, "Towards UML-Intensive Framework for Model-Driven Development," Lecture Notes in Computer Science, vol. 5082, pp 116-128, 2008.

[4] G. Booch, R. Maksimchuk, M. Engle, B. Young, J. Connallen, K. Houston, "Object-Oriented Analysis and Design with Applications," ACM SIGSOFT Software Engineering Notes archive, vol. 35, September 2008, doi: 10.1145/1402521.1413138.

[5] P. Coad and E. Yourdon, "Object-Oriented Analysis," Prentice Hall, Englewood Cliffs, New Jersey, 2nd ed.,1991.

[6] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy and W. Llorensen, "Object-oriented Modeling and Design," Prentice Hall, Eglewood Cliffs, New Jersey, 1991.

[7] J. Rumbaugh, "OMT Insights," SIGS Publications, New Tork, 1996.

[8] http://www-01.ibm.com/software/awdtools/developer/rose/.2007.

[9] http://www.sy-base.com/products/modelingdevelopment/ power designer, 2012.

[10] http://office.microsoft.com/en-us/visio, 2012.

[11] X. Liu, L. Dong, and H. Lin,"High Order Object Oriented Modeling Technique For Structured Object-Oriented Analysis" International Journal of Computer and Information Science, vol.2, No.2, 2001.

[12] J. Wu, M. Leu, and X. Liu, "A Hierarchical Object-oriented Functional Modeling Framework for Co-Design of Mechatronic Products," Concurrent Engineering, vol.17, No.4, pp 245-256, December 2009.

[13] X. Liu, "HOOM: A High Order Object Model," Proc. of the IASTED International Conference on Software Engineering, San Francisco, CA,No.,1997.

[14] X. Liu,and H. Lin,"High order Object Model Based Software Analysis."Proc. of the 1997 International Conference on Computer Software and Application Software,vol.20, no.6, June 2009, pp.1475-1469, doi: 10.3724/SP.J.1001.2009.03455.

Automated Construction of System Domain Knowledge Using an Ontology-Based Approach

Mohammad Moshirpour
Department of Electrical and Computer Engineering
University of Calgary, Canada
mmoshirp@ucalgary.ca

Armin Eberlein
Department of Computer Science and Engineering
American University of Sharjah
Sharjah , UAE
eberlein@ucalgary.ca

Behrouz H. Far
Department of Electrical and Computer Engineering
University of Calgary, Canada
far@ucalgary.ca

***Abstract*— Lack of central control in distributed systems makes the requirement analysis and design of such systems a challenging task. Literature suggests that detection and removal of defects prior to the deployment of a software system is several times less expensive than the discovery of faults afterwards. Since manual review of software requirements and design artifacts is too inefficient and time-consuming, devising automated methodologies to analyze software requirements is greatly desirable. However automating the process of software analysis is a challenging task because each software system has different domain knowledge. The existing approaches in the literature often require a great deal of input from the system engineer which makes for a time-consuming process. This research suggests the use of scenario-based software engineering to represent system requirements. Scenarios are often depicted using message sequence charts (MSCs). Due to their formal notation, MSCs can be used to analyze software requirements in a systematic manner. This paper presents the use of an ontology-based approach to create the domain knowledge of software systems. This domain knowledge then can be used to analyze software requirements and design artifacts to detect system deficiencies. This methodology is demonstrated using the case study of a real-time fleet management system.**

Keywords - distributed system; ontology; emergent behavior; automated analysis, scenario-based specifications; message sequence charts.

I. INTRODUCTION

Review of requirements and design documents prior to implementation of software systems is an effective and efficient approach to prevent introducing flaws into a system. Literature suggests that detecting unwanted behavior during the design phase is about 20 times cheaper than finding them during the deployment phase [1]. One of the greatest challenges in the design of distributed systems is emergent behavior [2-7]. In general emergent behavior is defined as a specification of behavior that is in the synthesized model of the distributed system, but is not explicitly specified in its specifications. Emergent behavior arises when there exists a state, in which a system component becomes confused as to what course of action to take. Although emergent behavior is not always problematic, there are many cases which emergent behavior is the root cause of system defects [2-7].

Unfortunately, manual review of the requirements and design documents may not efficiently detect all design flaws depending on the scale and complexity of real-life systems. Therefore devising automated methodologies to review and verify system requirements and design artifacts are highly desirable. The first step towards automatic analysis of software requirements is to express them using a precise notation. An effective and efficient way to describe system requirements is using scenario-based specifications. A scenario is a temporal sequence of messages sent between a system and actors using the system. Scenarios are appealing because of their simplicity and expressive power [2]. This research uses Message Sequence Charts (MSC) developed by the International Telecommunication Union (ITU) [8] to represent scenarios.

Several methodologies which systematically analyze software requirements have been introduced in the literature [3-5, 9-11]. Each of these methodologies requires a considerable amount of input from the system engineer which makes the analysis process very time consuming.

However it is very challenging to fully automate the process of software requirement analysis. This is mainly because each software system has its own particular domain knowledge. This research works on devising a comprehensive framework to detect emergent behavior in distributed and multi-agent systems (MAS) [4, 12]. In [12] a methodology to detect emergent behavior was introduced. However this approach was dependent on the construction of the domain knowledge by the system engineer [12]. This paper presents an approach to increase the level of automation of the methodology shown in [12]. This is done by automating the process of building the domain knowledge of a software system using an ontology-based approach.

The rest of this paper is organized as follows: Section 2 contains the case study of the real-time fleet management system. In Section 3 the process of behavior modeling is illustrated. Section 4 contains the process of building the ontology and extracting the domain knowledge of the system. The detection of emergent behavior is discussed in Sections 5 and the conclusions and future work are given in Section 6.

II. CASE STUDY

The Real-time fleet management system provides a multi-purpose solution for transportation companies and their

customers. Such systems enable concise tracking of vehicles which results in accurate scheduling. A prime example of such systems is Real-time city transit information systems which have been employed in a number of large cities. Transit users are able to receive real-time schedules of bus stops and can even register to be notified of bus arrivals by receiving text messages or emails. Due to the diverse technologies used by users, the system must provide support for different platforms such as different browsers, mobile web, as shown in Figure 1.
The transit information system keeps track of the location of each bus using GPS data received. It then attempts to estimate the time remaining until each stop by considering other data such as: weather conditions received from weather network and traffic conditions received from roadside infrastructure. The requirements of this system are defined using scenarios as illustrated in Figure 2-4. These MSCs depict the estimation of bus arrival times by the system. It can be assumed that each category of data (i.e. traffic, weather and location data) reaches the server on set intervals, unless there is a sudden change of conditions. For instance, the traffic data for a given street is reported by the roadside infrastructure every hour.

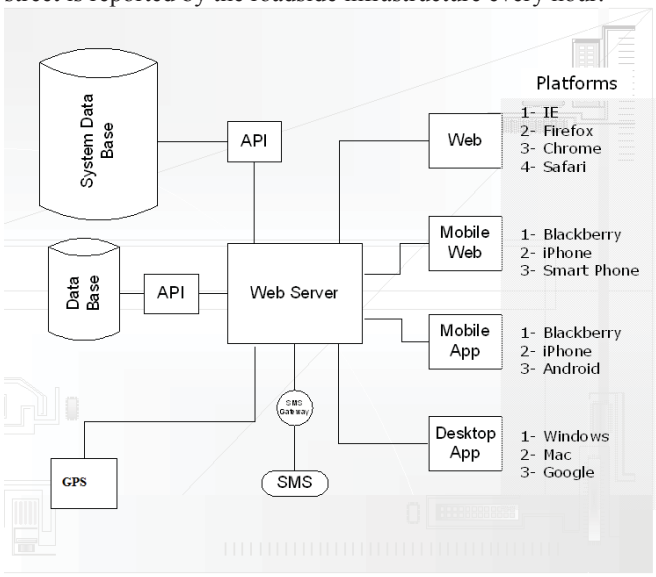


Figure 1. High level design of the fleet management system

However in the event of a sudden change in traffic condition (such as a traffic jam due to an accident) this information is transmitted to the server right away and arrival times are estimated and reported accordingly. Moreover, data can be entered manually to effect bus arrival times, such as mechanical failure (Figure 4).

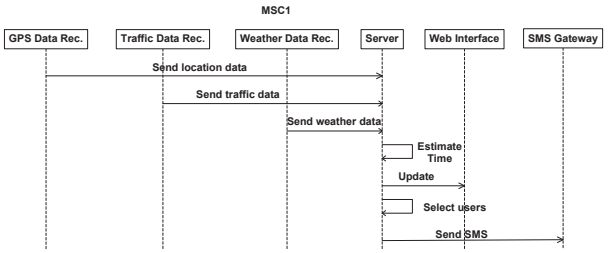


Figure 2. The system calculates bus arrival times and notifies users

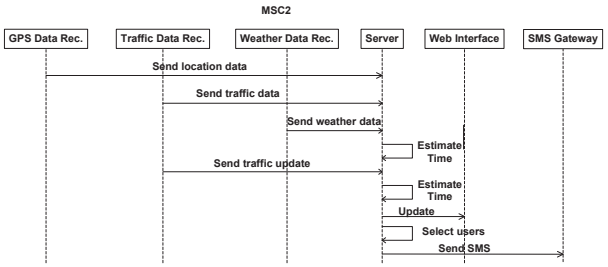


Figure 3. System recalculates bus arrival according to traffic update

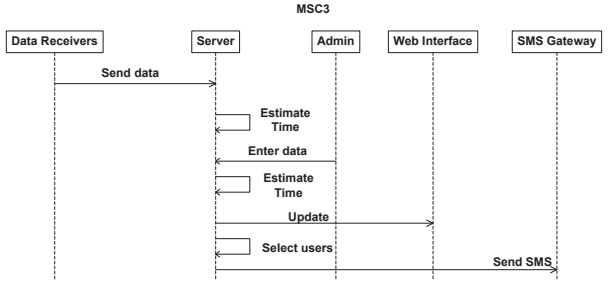


Figure 4. System recalculates bus arrival according to traffic update

III. BEAHVIOR MODELING

The procedure of construction of finite state machines (FSMs) from message sequence charts (MSCs) is referred to as behavior modeling [4]. For any process i of a MSC, an equivalent finite state machine can be constructed. Figures 5 illustrate the eFSM constructed for the infrastructure component in MSCs 1. It is important to note that, regardless of what type of data is received by the system, this data triggers the calculation of bus arrival times. Thus, for the sake of partiality, there is no sense in distinguishing between data types received in the behavior models. Therefore all data received is simply denoted as "Received message" or "Rec. msg" for short.

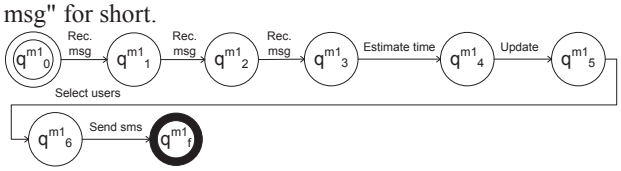


Figure 5. FSM for the Server component in MSC 1

The behavior model for the Server component is obtained by the union of all the individual state machines [4].

IV. CONSTRUCTION OF THE DOMAIN KNOWLEDGE

After the synthesis of behavior models for each system component, the state values for the models are to be calculated. To do this, an invariant property of the system called semantic causality is used:

Definition 1 (Semantic causality): A message $m|_i[j]$ is a semantical cause for message $m|_i[k]$ and is denoted by $m|_i[j] \overset{se}{\rightarrow} m|_i[k]$, if component *i* has to keep the result of the operation of $m|_i[j]$ in order to perform $m|_i[k]$.

For instance, in MSC1 shown in Figure 2, the message "Rec. msg." is the semantic cause for the message "Estimate time". As semantic causality is an invariant property of the system and is part of the system's architecture and the domain knowledge, it is independent of the choices made by the domain experts. In other words, we let the current state of the component to be defined by the messages that the component needs in order to perform the messages that come after its current states.

Following the definition of semantic causality, the domain theory of the system is constructed as follows:

Definition 2 (Domain theory): The domain theory D_i for a set of MSCs M and component $i \in P$ is defined such that for all $m \in M$, if $m|_i[j] \overset{se}{\rightarrow} m|_i[k]$ then $(m|_i[j],\ m|_i[k]) \in D_i$.

Following the above example, both messages "Rec. msg." and "Estimate time" are part of the domain theory as one is the semantic cause of the other. The steps to be followed to build the domain theory using the ontology-based approach are outlined in this section.

A. Constructing the Ontology

This research proposes that two different views of ontologies are built for the system; namely static view and dynamic view.

1) Static View of Ontology

The static view of ontology is much like a tree structure, where the elements are components of the system and are related to each other in this ontology based on their hierarchy within the system. The static view of the ontology is constructed by the domain expert (Figure 6) illustrates the static view of the ontology for the real-time transit information system. In the static view, the domain expert classifies all system components into a finite set of categories.

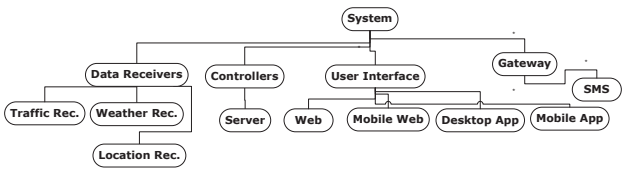


Figure 6. Static view of the ontology

2) Dynamic View of Ontology

The dynamic view of ontology represents the interactions between system components and is constructed by the domain expert. This view describes the aspects of the system which can change with time. Figure depicts the dynamic view of the ontology for the real-time transit information system, which has been constructed using a deterministic finite state automaton. The states of this automaton are the categories which were established in the static view of the ontology (Shown in Figure 7).

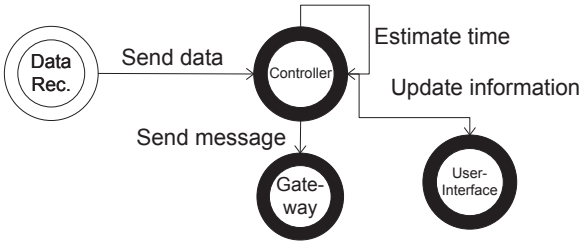


Figure 7. Dynamic view of the ontology

As is the case with state machines, the dynamic view of ontology will have 3 different types of states: start state (e.g. Data receivers in Figure), transition states, and final states (e.g. Gateway in Figure). General definitions for each state type are as follows:

Start State - Resembles a category of components that will only send messages, but do not receive any messages.

Final State - Resembles a category of components in which a type sending or receiving a message results in completing a task.

Transition State - is a state that is determined to be not a start nor a final state.

B. Constructing Tables

To build the domain theory, tables are built for each state to discover semantic causality between messages. For each state, the rows are the transitions before the state, and the columns are the transitions after it. The sender/receiver component for each message is also indicated in the tables as shown in Table 1.

The information required to fill the tables is extracted from the static and dynamic views of the ontology as follows:

- For each row consider the component which has sent/received that message
- Using the static view of the ontology, determine what category it belongs to
  - For each column consider the component which has sent/received that message
  - Using the static view of the ontology, determine what category it belongs to
    - Now consider the dynamic view of the ontology
    - The category of the sender/receiver component of the message in the row is the starting point
    - The category of the sender/receiver component of the message in the column is the end point
    - If there is a path from the start point to the end point AND the end point is a final state
      - The message in the row is a semantic cause for message in the column and the table completion is completed
  - End For
- End For

Table 1. Finding Semantic causality for state q_3^1

	Estimate time **[Server]**	Update **[Web Interface]**	Send sms **[SMS Gateway]**
Rec.msg**[Weather Data Rec]**	X		
Rec. msg **[Traffic Data Rec]**			
Rec. msg **[Location Data Rec]**			

Upon building the domain theory based on semantic causality we proceed to assign state values to the states of the constructed eFSM as explained in Definition 3.

Definition 3 (State value): The state value $v_i|(q_k^m)$ for the state q_k^m in eFSM $A_i^m = (S^m, \Sigma^m, \delta^m, q_0^m, q_f^m)$ is a word over the alphabet Σ_i U {1} such that $v_i|(q_f^m) = m|_i[f-1]$, and for 0 < k < f is defined as follows:

i) $v_i|(q_k^m) = m|_i[k-1]v_i|(q_j^m)$, if there exist some j and l such that j is the maximum index that $m|_i[j-1] \overset{se}{\rightarrow} m|_i[l], 0 < j < k,\ k \leq l < f$

ii) $v_i|(q_k^m) = m|_i[k-1]$ if case i) does not hold but $m|_i[k-1] \overset{se}{\rightarrow} m|_i[l]$, for some k ≤ l < f

iii) $v_i|(q_k^m) = 1$, if none of the above cases hold

For example, to calculate the state value for the state q_4^{m1} the following steps are followed: From the domain theory of Definition 5, it can be deducted that the maximum index *j* for which $m1|_{Server}[j-1]$ is a semantical cause for a message in the transitions after q_4^{m1} is j = 3 for which $m1|_{Server}[j-1] = Rec.\ msg.$ That is to say that for example the message "Rec. msg" is a semantic cause for message "Update".

Therefore, it is concluded that the value of the state q_4^{m1} is obtained using case (ii) of Definition 6 as follows: $v_{Server}|(q_4^{m1}) = m|_{Server}[4-1]v_{Server}|(q_3^{m1})$. This becomes: $v_{Server}|(q_4^{m1}) = (Rec.msg.)\ v_{Server}|(q_3^{m1})$

V. Detection of Emergent Behavior

After calculating state values using the domain theory, the basis for comparing states and consequently discovering identical states is established. Identical states are defined in Definition 7 as follows.

Definition 4 (Identical states): Two states q_j^m and q_k^n of process *i*, (*m* and *n* could be the same) are identical if one of the following holds:

i) j = k for $0 \leq t < j: m|_i[t] = n|_i[t]$

ii) $v_i(q_j^m) = v_i(q_k^n)$

Figure 8. Merging identical states

Thus as shown in Figure 8, S2 is where the system becomes confused as what source of data to use; the received messages or the manual update. That is, there could arise a scenario, where the received data can over-write the manual data entered.

VI. Conclusions and Future Work

Literature suggests that detecting unwanted behavior during the design phase is about 20 times cheaper than finding them during the deployment phase [1]. Therefore it is greatly beneficial to devise methodologies to analyze software requirements and design artifacts in an effective and efficient manner. However, many of the existing methodologies used to analyze system requirements and design documents require a lot of manual input from the system engineer and thus introduce a certain amount of overhead [4]. The goal of this research is to devise a systematic approach to analyze system requirements for emergent behavior, while saving on overhead by replacing ad-hoc methodologies with automated ones [4, 13]. In this paper an approach to extract the domain knowledge of the system from scenarios using an ontology-based approach was presented. Future work would include automatic ontology generation from scenarios and completing a software package based on this methodology [4].

References

[1] R. F. Goldsmith, *Discovering Real Business Requirements for Software Project Success*. Norwood MA: Artech House, Inc., 2004.

[2] *Casual Closure for MSC Languages* 2005.

[3] R. Alur, K. Etessami, and M. Yannakakis, "Inference of Message Sequence Charts," *IEEE Transaction on Software Engineering*, pp. 623-633, July 2003.

[4] M. Moshirpour, "Model-Based Detection of Emergent Behavior In Distributed and Multi-Agent Systems from Component Level Perspective," Master of Science Department of Electrical and Computer Engineering, University of Calgary, Calgary, 2011.

[5] A. Mousavi, "Inference of Emergent Behaviours of Scenario-Based Specifications," PhD Thesis PhD Thesis, Department of Electrial and Computer Engineering, University of Calgary, 2009.

[6] H. Muccini, "Detecting implied scenarios analyzing nonlocal branching choices," presented at the FASE 2003, Warsaw, Poland.

[7] S. Uchitel, J. Kramer, and J. Magee, "Negative scenarios for implied scenario elicitation," presented at the 10th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE 2002), Charleston.

[8] "ITU: Message Sequence Charts. Recommendation, International Telecommunication Union. ," 1992.

[9] I. Kruger, R. Grosu, P. Scholz, and M. Broy, "From mscs to statecharts," in *Franz j. rammig (ed.): Distributed and parallel embedded systems*, ed: Kluwer Academic Publis, 1999.

[10] J. Whittle and J. Schumann, "Generating statecharts designs from scenarios," presented at the ICSE, Limerick, Ireland, 2000.

[11] J. Whittle and J. Schumann, "Scenario-Based Engineering of Multi-Agent Systems," in *Agent Technology from a Formal Perspective*, Third ed London: Springer-Verlag, 2006.

[12] M. Moshirpour, A. Mousavi, and B. H. Far, "Detecting Emergent Behavior in Distributed Systems Using Scenario-Based Specifications," in *International Conference on Software Engineering and Knowledge Engineering*, San Francisco Bay, 2010.

[13] M. Moshirpour, S. Mireslami, A. Eberlein, and B. H. Far, "A method to detect and remove emergent behavior caused by overgeneralization " in *Systems, Man, and Cybernetics (SMC 2012)*, COEX, Seoul, Korea, 2012, pp. 2469 - 2474.

Dynamic Adaptation of Cloud Computing Applications

André Almeida[1,2], Everton Cavalcante[2], Thais Batista[2], Nélio Cacho[2], Frederico Lopes[2], Flavia Delicato[3], Paulo Pires[3]

[1] Federal Institute of Education, Science and Technology of Rio Grande do Norte (IFRN), Parnamirim, Brazil
[2] Federal University of Rio Grande do Norte (UFRN), Natal, Brazil
[3] Federal University of Rio de Janeiro (UFRJ), Rio de Janeiro, Brazil

andre.almeida@ifrn.edu.br, evertonrsc@ppgsc.ufrn.br, thais@ufrnet.br, neliocacho@dimap.ufrn.br, {fred.lopes, fdelicato, paulo.f.pires}@gmail.com

***Abstract*—Cloud-based applications are composed of services offered by distinct third-party cloud providers. As most cloud-related information (i.e. properties of the services such as price, availability, response time, etc.) of the services are dynamic and may change any time during the application execution, it is essential to adapt the application upon the detection of QoS violations that affect the application requirements. In this paper we present a dynamic adaptation approach managed by an autonomic control loop that takes place when a service becomes unavailable or when QoS parameters are degraded. Our dynamic adaptation approach relies on *dynamic aspect-oriented programming* (DAOP) to: (i) encapsulate the dynamic adaptation (removal and/or insertion of services) as an *aspect* that contains *join points* that specify where each aspect must act, and; (ii) easily change the application by dynamically removing a service and inserting a new one.**

Keywords-Cloud Computing, Software Product Lines, Aspect-Oriented Programming, Dynamic Adaptation, Autonomic Computing.

I. INTRODUCTION

Cloud-based applications are inherently dynamic as they rely on a set of services provided by several underlying cloud computing platforms that can suffer from instability and Quality of Services (QoS) fluctuations. Moreover, the major difficulties in terms of developing such applications encompass issues such as the decision of which underlying cloud computing platforms to use, the need of continuously monitoring the dynamic cloud-related information of the very broad variety of services, and the need of adapting the application upon the detection of QoS violations that affect the application requirements. In this context, the *software product lines* (SPL) [1] paradigm is useful for [2]: (i) representing the alternative cloud services to be used by the applications as variabilities; (ii) configuring the application by choosing the proper cloud platform service that fits the application needs, and; (iii) annotating the cloud-related QoS information as properties of each service, thus making it easier the identification of the dynamic information to be monitored.

In this paper we present a dynamic adaptation approach for cloud-based applications that takes place when a service becomes unavailable or when an agreed QoS parameter is violated. Our dynamic adaptation approach relies on *dynamic aspect-oriented programming* (DAOP) [4] to: (i) encapsulate the dynamic adaptation (removal and/or insertion of services) as an *aspect* that contains *join points* that specify where each aspect must act, and; (ii) easily change the application by dynamically removing a service and inserting a new one. In fact, the use of DAOP has been a natural choice for supporting adaptation of dynamic SPL [5] as it supports *late variability* to deal with elements that can change at runtime, such as cloud services. In our scenario, the flexible variability mechanism of DAOP enables cloud services, represented as variabilities, to be woven and unwoven at the application at runtime. In addition, we use our existing SPL-based monitoring strategy [3] to detect when an adaptation is required.

This paper is structured as follows. Section II briefly presents the background of this work. Section III contains the description of our adaptation approach for Cloud Computing applications, and the implementation details. Section IV presents an evaluation of our approach. Section V discusses related works. Finally, Section VI presents final remarks.

II. BACKGROUND

A. Cloud Computing

Cloud Computing is a paradigm that enables ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or interaction with the service provider and are provided in a pay-per-use way to the user. There are three fundamental models of cloud service providers: (i) IaaS (*Infrastructure as a Service*) platforms, which often provide physical resources such as virtual machines, servers, networks, etc.; (ii) PaaS (*Platform as a Service*) platforms, which typically provide an underlying infrastructure to develop, deploy, execute, and manage cloud-based applications, and; (iii) SaaS (*Software as a Service*) platforms, which provide application software on the cloud and users can access them by using a browser.

Building cloud-based applications is a challenging task as they are significantly more complex due to the intrinsic complexity of using third-party cloud providers. The major difficulties encompass issues such as the decision of which underlying Cloud Computing platforms to use, and the need of tracking pricing policies of services provided by different clouds platforms, thus hampering the development of applications using different cloud services [2]. Furthermore, the particular nature of Cloud Computing applications creates specific requirements that also demand changes in terms of the development of such applications, encompassing methodologies and techniques for requirements elicitation, architecture, implementation, deployment, testing, and evolution of software. In such context, this

paper strives to make easier the development of applications using different cloud services, so that these applications can be monitored at runtime and adapted under dynamic conditions that may affect their requirements. To achieve these goals, our approach uses two paradigms, namely *software product lines* [1] and *aspect-oriented programming* [7], as briefly introduced in Section II.B.

B. Software product lines and Aspect-Oriented Programming

SPL [1] enable the creation of a *family* (or *product line*) of similar products by identifying *commonalities* (similarities) between all members of the family as well as characteristics that vary among them, the *variabilities*. Commonalities and variabilities between products of a family are typically modeled in terms of *features*, which may be a requirement, a function, or a non-functional feature depending on the interest of such stakeholder involved in the application development. Features can be [6]: (i) *mandatory*, i.e. the feature must be included in a product; (ii) *optional*, i.e. the feature may or may not be included if the feature from which it derives is selected; (iii) *or-inclusive*, i.e. among the set of related features at least one of them must be selected, and; (iv) *alternative*, i.e. among the set of related features exactly one of them must be selected.

AOSD [7] emerged as an approach to promote the modularization of *crosscutting concerns*, which are usually spread over several modules in a software system. Without proper means for separation and modularization, crosscutting concerns tend to be scattered over a number of modular units and tangled up with other concerns, thus resulting in lower cohesion and stronger coupling between modular units, and reduced degrees of understanding, evolvability, and reusability of software artifacts. In AOSD, crosscutting concerns are modularized as *aspects*, which are abstractions used to encapsulate crosscutting concerns that are associated with a set of classes and/or objects affected by such aspects. In turn, *basic concerns* of a software system are non-crosscutting concerns that can be modularized as conventional classes and objects, so that a mechanism called *weaver* is responsible for composing (*weaving*) the code regarding basic and crosscutting concerns. In aspect-oriented programming (AOP) [7] the major mechanisms to modularize crosscutting concerns in terms of aspects are *join points*, *pointcuts*, and *advices*. *Join points* are well-defined points in the application code (e.g. method calls) that specify how classes and aspects are related. Each aspect defines one or more expressions called *pointcuts*, which are used to select the join points that will be affected by the aspect's crosscutting behavior. Finally, when the program execution reaches a join point selected by some *pointcut* expression, a piece of code called *advice* attached to a *pointcut* can be executed *before*, *after* or *around* it: (i) a *before* advice runs whenever a join point is reached and before the current computation proceeds; (ii) an *after* advice runs after the method body that has run and just before returning the execution control to the caller, and; (iii) an *around* advice runs whenever a join point is reached and has explicit control whether and when the computation under the join point is allowed to run.

In the synergic relationship between AOP and SPL [8], recent research has pointed out that AOP promotes better modularity and changeability of SPL than conventional variability mechanisms [9]. Besides modularizing crosscutting concerns and managing variabilities in an SPL [10, 11], AOP is able to improve its evolvability and stability upon dynamic scenarios [11]. Aspects can contribute to modularize variabilities and facilitate their addition or removal according to the product (application) configuration, which is usually based on the selection of a set of features [8, 12]. For these reasons, we exploit such relationship between AOP and SPL in our approach for supporting adaptation of an SPL as it supports late variability to deal with elements that may change at runtime [5], such as cloud services. As we present hereafter, the flexible variability mechanism of dynamic AOP enables cloud services, represented as variabilities, to be woven and unwoven at the at runtime, thus supporting the adaptation of the application [4].

C. Running example:HW-CSPL

In our previous work [2], we have proposed a seamless adaptation of the SPL-based development to support specificities of cloud-based applications by adopting an extended feature model in order to introduce attributes to the features, in which an attribute is any characteristic of a feature that can be measured. Similarly, we also ground on this idea of introducing attributes to features in the feature model with the notion of *properties*, which have the form of <*name*, *type*, *value*> triples regarding a feature. Thus, a property can represent *cloud-related information* such as pricing, elasticity support and QoS parameters. In addition, the feature model becomes more expressive in order to represent important characteristics of the cloud services such as pricing model, availability, and response time. In order to illustrate such approach, we have developed HW-CSPL (*Health Watcher Cloud Software Product Line*), an SPL developed from the *Health Watcher* (HW) [13] real Web-based system. HW enables citizens to consult information about the public health system of a city and to register complaints in terms: (i) ingestion of contaminated food; (ii) mistreatment of animals or diseases transmitted by contaminated animals, and; (iii) other cases, e.g. hygiene problems in restaurants, sewage leaks, etc. The commonalities were proposed from the requirements and features in the original HW system and the different service facilities provided by cloud platforms led to the features that represent the variabilities.

Fig. 1 illustrates the HW-CSPL extended feature model. It contains mandatory features representing commonalities: (i) *Persistence*, the persistence mechanism of the application, and; (ii) *Log System*, the infrastructure used for storing log information. Such model also contains one optional feature, *File Storage*, which defines how files (e.g. images related to the application data) are managed in the application. Each one of these top-features has properties regarding the services represented by their alternative feature groups. For instance, the *Persistence* feature has three dynamic properties (*price*, *availability*, and *responseTime*) and offers two options for application's data persistence, respectively represented by the *Relational Amazon RDS* feature, which is related to the Amazon RDS database service provided by Amazon Web Services (AWS) [18], and by the *Relational HP Cloud*, which is related to the relational database service provided by the HP Cloud platform [19].

III. DYNAMIC ADAPTATION OF CLOUD APPLICATIONS

QoS parameters and other dynamic-kind information regar-

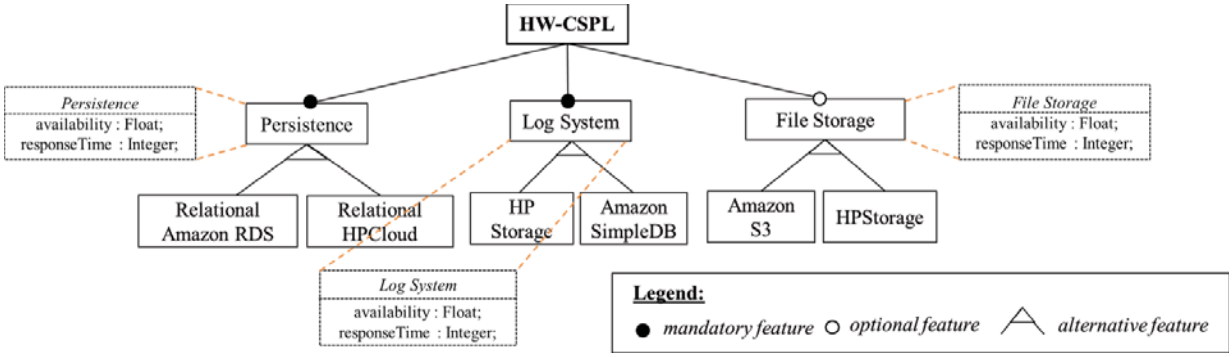


Figure 1. HW-CSPL feature model.

ding the used cloud services may change over time, thus affecting the deployed applications that make use of such services. In this perspective, our previous work [3] introduced a strategy that enables to continuously monitor the dynamic properties of the cloud services that are required/used by an application. In this work we extend our previous approach by using the MAPE-k loop [25], as illustrated in Fig. 2. In the *Monitoring* phase the values gathered by the *Feature Monitoring Agent* are stored in a database managed by the *Knowledge* component, which is currently responsible for storing all information used in our strategy to achieve the adaptation of cloud applications. In the *Analysis* phase, the *Product Generator and Evaluator* generates the *product description*, which stands for the configuration of the product to be deployed. This is achieved by parsing the *feature model* and evaluating the *product selection criteria*, described in more details in our previous work [3]. Afterwards, the generated *product description* is stored in the *Knowledge* component and serves as input for the *Planning* phase, in which the *Aspect Composer* component parses the *product description* in order to generate a *pointcut/advice model* to be used by the *Dynamic Weaver* component in order to reconfigure the application in the *Execution* phase. All of these elements are conceptually described in the following subsections. Although Cloud Computing offers several models, we concentrate our approach on IaaS platforms, more specifically the AWS and HP Cloud platforms, but our strategy is generic and can be used with other platforms. As we are working with dynamic adaptation using QoS information, it is fair to compare services provided by platforms that follow the IaaS model.

A. Feature model

In our previous work [3], the feature model regarding the SPL was extended in order to enable the user to annotate the features with dynamic properties to be monitored. Now, such feature model was extended again by adding two new elements: (i) the points of interest (represented as *pointcuts*) that describe which parts of the application are susceptible to adaptation; (ii) the code responsible for implementing the variability (represented as an *aspect*) and how they are bound to the *pointcuts*.

Fig. 3 shows a fragment of the XML representation regarding to the *FileStorage* feature in HW-CSPL. In lines 2 to 6 in Fig. 3, the *pointcuts* tag contains the declaration of the *pointcut pc01* associated to the *FileStorage* feature (lines 3 to 5). In line

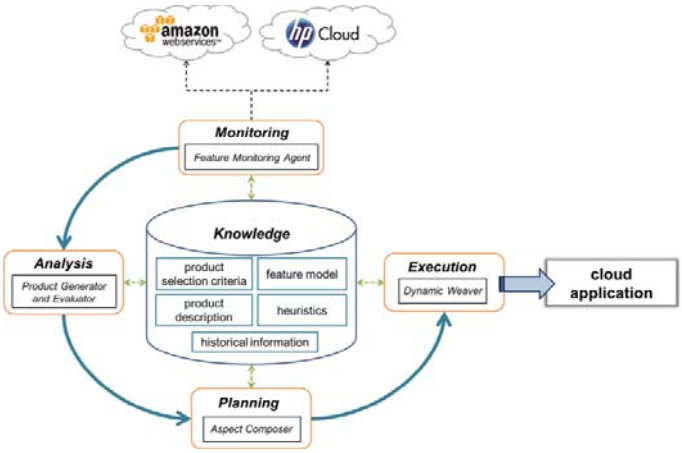


Figure 2. Overview of the proposed dynamic approach

4, the *pointcut* expression `execution(hw.Storage->store(*))` means that the interception must happen when the *store* method regarding the *Storage* class is executed even if it has multiple signatures (as represented by the wildcard *). In addition, it is possible to define how such *pointcuts* are associated to the aspects/advices that implement the variabilities of the feature model by using the *bindings* tag, as shown in lines 9 to 12 of Fig. 3. Advice types are the same used on traditional AOP: in a *before* advice a specific behavior must be executed before reaching the *pointcut*, in an *after* advice, a specific behavior must be executed after reaching the *pointcut*, and in an *around* advices replaces the current execution of the *pointcut*. As shown in lines 10 and 11, the *pointcut pc01* is related to an *around* advice named *store* and that is implemented by the class *hw.aspects.storage.HPStorage* class.

B. Product Generator and Evaluator

The extended feature model and the product selection criteria serve as inputs to the *Product Generator and Evaluator* component, which was extended from our previous work [3]. This component evaluates the product selection criteria by using the monitored values of the feature attributes and then generates the *product description*, a XML description of the selected product according to such criteria and that will be deployed/adapted. Such product description consists in speci-

```
<alt abstract="true" name="FileStorage">
    <pointcuts>
        <pointcut name="pc01">
           execution(hw.Storage->store(*))
        </pointcut>
    </pointcuts>
    <feature mandatory="true" name="HPFileStorage"
        featuremonitoringbean="monitoring.MonitoringHPFileStorage">
        <bindings>
            <around pointcut="pc01"
               aspect="hw.aspects.storage.HPStorage"
               name="store" />
        </bindings>
    </feature>
    <feature mandatory="true" name="AmazonS3"
        featuremonitoringbean="monitoring.MonitoringS3FileStorage">
        <bindings>
            <around pointcut="pc01"
                aspect="hw.aspects.storage.S3Storage"
                name="store" />
        </bindings>
    </feature>
    <properties>
        <property name="availability" type="double" />
        <property name="responseTime" type="double" />
    </properties>
</alt>
```

Figure 3. XML representation regarding the *FileStorage* feature in HW-CSPL.

fying the features that compose the product and contains information about the *pointcuts* and a reference to the implementation of the variabilities that compose the product, as well as *where* aspects must be weaved into the application, thus encompassing the definition of *pointcuts* expressions and their associated *advices* and *aspects*.

Fig. 4 shows a fragment of the XML representation of the *product description* that uses the *RelationAmazonRDS* variability associated to the *Persistence* feature. In lines 3 to 13, the *pointcut* and aspect/advice are described for the *RelationalAmazonRDS* variability. Due to space restrictions, this description contains only the information needed for reconfiguring the application, so that the other details specified in the XML representation of the feature model (see Fig. 3) were removed.

```
<product>
    <feature name="RelationalAmazonRDS">
        <pointcuts>
            <pointcut name="pper">
               execution(lib.persistence.
               PersistenceMechanism->
               getCommunicationChannel())
            </pointcut>
        </pointcuts>
        <bindings>
            <before pointcut="pper"
               aspect="hw.aspects.persistence.AWSPersistence"
               name="connect" />
        </bindings>
    </feature>
    ...
```

Figure 4. XML representation of the product description.

C. Aspect Composer

The *Aspect Composer* component is responsible for parsing the product description in order to generate a *pointcut/advice model*, which is a representation of the product to be adapted with *pointcuts*, aspects, and advices and described how the application must be reconfigured by using the interface provided by the *AOP Handler* component, which provides an interface to receive the *pointcut* and aspect/advice models. If it is the first deployment of the application, the *AOP Handler* is accessed to define the *pointcuts* and bind the aspects/advices to such *pointcuts*. If the application has already been deployed, it is necessary to first unbind the current aspects associated to the application and then bind the new aspects that are described in the product description.

The *AOPHandler* component is responsible for interacting with the *Dynamic Weaver* implementation, so that it is necessary to know how it works. Our proposal is to provide a flexible solution in order to enable the developer/user to change the *Dynamic Weaver* component. For this purpose, we defined an *IAOPHandler* interface to be used by the *Aspect Composer* to register/unregister the *pointcuts*, aspects, and advices. Our current implementation uses the JBoss AOP framework [16] for implementing the *Dynamic Weaver* component, so that the *AOPHandler* component knows how to deal with the provided API. For example, if the developer/user wants to change from JBoss AOP to the Guice injection framework [17], the provided implementation must know how to: (i) register the defined *pointcuts* by interpreting the *pointcut* grammar, and; (ii) bind/unbind *pointcuts* with the defined aspects/advices.

D. Dynamic Weaver

The *Dynamic Weaver* component supports not only the AOP programming model, but it also has the capability of dynamically *weaving* code into the application. In traditional AOP, the developer must specify, at design time, the *pointcuts* and aspects (crosscutting concerns) that must be weaved into the application. In traditional AOP, the weaving of aspects is typically done only at compilation time. On the other hand, *dynamic weaving* of applications stands for the capability of dynamically introducing aspects within the application at runtime. To add new functionalities to components only available at the binary format, it is necessary to rely on *code instrumentation*, which means changing the bytecodes of compiled classes of Java applications, for example.

For dynamic weaving capabilities, our *Dynamic Weaver* component must: (i) provide a *grammar* that enables to define *pointcut* expressions or the capability for adding such expressions; (ii) provide an *API* that supports the runtime definition of aspects and how *pointcuts* are related to advices/aspects, and; (iii) use a *code instrumentation strategy* to support dynamic weaving of aspects. The *Dynamic Weaver* implementation used in this work is the JBoss AOP framework [16], which supports static and runtime weaving. It enables to insert aspects by using: (i) annotations or XML annotations for static changes, and; (ii) the JBoss AOP API for runtime changes. Our *pointcut* description uses the full grammar of JBoss AOP, thus enabling the user to define a rich feature model since the feature implementation (variability) may be scattered in multiples point of the code. To support the runtime binding/unbinding of aspects, JBoss AOP relies on *code instrumentation*, thus incurring in the manipulation of the generated Java bytecode. Many other solutions for dynamic weaving, such as PROSE [22] and AspectWerkz, rely on the modification of the Java Virtual Machine (JVM), which may be a prohibitive issue in the context of Cloud Computing applications. However, a drawback of JBoss AOP is that it must be loaded with the JVM used by the container/application server, which is required by the Java programming language to instrument the Java code. Since it requires more access to the JVM, using such solution for a PaaS

environment, which may have restrictions in terms of JVM manipulation, is not feasible. The Guice solution [17] is a dependency injector that does not require such access, but it lacks of support of full features for AOSD because it is necessary to implement all *pointcut* expressions and it only supports around advices. In future works, we intend to implement a solution that makes use of the Guice framework and evaluate our proposal with PaaS platforms.

IV. EVALUATION

We have evaluated the proposed approach by using the case study described in Section II.C. It was deployed in an OpenStack private cloud environment running a single cloud server instance (Tiny - 512 RAM and 1 VCPU) and using JBoss as its application server and Linux as base operating system. We have purchased an Amazon RDS instance and HP Compute instance with a MySQL database to evaluate the *Persistence* feature and loaded them with a set of random data. In order to evaluate the *FileStorage* feature, we have used the Amazon S3 and HP Cloud Storage services, and for the *Logging System* feature we have used the Amazon SimpleDB and HP Cloud Storage since the HP Cloud platform does not offer a non-relational database service like Amazon SimpleDB., In order to measure the performance of the whole adaptation process for adapting from a product to another, we have measured the time spent by the adaptation of each feature of the case study. As the product is selected based on the monitored values gathered from the purchased services, we have decided to provide a given set of feature monitoring values in order to trigger the adaption and to evaluate all possible scenarios (i.e. all the variabilities presented in the feature model). We have measured 1000 product changes by measuring the time for each feature takes to be adapted. In Table 1 we present for each feature of the SPL the maximum, minimum, and average times (in milliseconds) spent by the adaptation process.

TABLE 1. EXECUTION TIME FOR THE DYNAMIC ADAPTATION PROCESS.

Feature	Minimum (ms)	Maximum (ms)	Average (ms)	Standard Deviation (ms)
Persistence	2968.36	3631.92	3240.19	194.67
File Storage	14.05	19.88	16.98	1.96
Log System	15.73	22.86	20.20	2.20

As shown in Table 1, the *Persistence* feature stands out when comparing with the other features due to the implementation of such feature. Since the HW system uses a pooling control for database connections, the adaptation process for this feature consists in releasing the connection pool and creating a new one according to the changed configuration (i.e. new URL, user and password for accessing the new database). The other two features are directly implemented to use the services (e.g. store an image by using the *FileStorage* feature or register a log entry by using the *LogSystem* feature), and since we are interested in evaluating the time for reconfiguring the application instead of the use of such services themselves, we consider such difference as a normal issue. Considering the set of changes and the observed standard deviations, for the *Persistence* feature we have only 27% of the 1000 test cases outside the standard deviation, and considering that lower times presents a better response, only 1/3 of the 27% (90 test cases) are above the average execution. For the *FileStorage* feature, the mentioned percentage is around 21% of the 1000 test cases. In this perspective, we consider that our solution presents a stable performance, considering the proposed scenario and feature model. It is important to notice that the measured time is added to the application execution time, thus meaning that the overhead due to the use of our strategy is represented by the times presented in Table 1. However, the process of adaptation is only triggered when changes in the deployed product occur, thus meaning that once a product is deployed and reconfigured, the application execution is not affected by our solution. It is also important to highlight that depending on the complexity of the features/variabilities, the overhead will be increased, as we mentioned before, but this time is counted for the feature implementation and not by our solution itself.

V. RELATED WORK

The idea of using SPL for supporting adaptation has been used by various works. Gomaa and Hashimoto [20] describe a dynamic software adaptation approach and environment for service-oriented product lines. This approach uses a dynamic feature model for a family of service-oriented architectures (SOA), in which a member of such family can be dynamically adapted to a different member of the family at runtime. The decision to adapt is based only on a set of software adaptation patterns, while our approach considers the *Product Selection Criteria* and the monitored values of the feature attributes. More similar to our approach, Baresi et al. [21] use the Common Variability Language (CVL) to augment BPEL (Business Process Execution Language) processes with variability. This makes it possible to generate a dynamic SPL and use an aspect-oriented based version of BPEL to manage and run the SPL. This approach is more suitable for self-adaptive SOA systems which are usually self-contained and loosely coupled. In contrast, our approach combines an expressive *Product Selection Criteria* with an extended feature model, which allows our approach to support a more fine-grained adaptation than the previous approach.

More recently, some approaches have been using SPL to manage software variability of cloud applications. In the Mietzner et al.'s work [14], variability techniques are used to support the management of variabilities in SaaS applications. Application templates describe the variability through variability descriptors. Likewise, FraSCAti [15], an adaptive and reflective middleware for multi-cloud systems, uses SPL to enable developers to select the configuration of the SaaS platform that matches the application needs. SPL is used only to represent the features and their constraints, which captures all possible configurations. In our work we go a step further by using SPL and aspects to implement an adaptation mechanism of cloud applications. This is an important difference of our work since our solution improves the adaptation and modularization of the application.

In addition, there has been much previous research using monitoring of QoS attributes to support dynamic adaptation. The Dai et al.'s work [23] is based on prediction of performance failures to support adaptation. The decision to adapt is based on the performance of a single service, while our approach considers the *Product Selection Criteria* and the moni-

tored values of each feature. In the Leitner et al.'s work [24], the PREvent approach is described to support prediction and prevention of SLA violations in service compositions based on event monitoring and machine learning techniques. The prediction of violations is calculated only at defined checkpoints in a composition based on regression classifiers prediction models. In contrast, our approach supports adaptations every time a product is chosen according to its *Product Selection Criteria*. Despite the similarities, none of these AOP approaches take the feature model and the *Product Selection Criteria* into account to support software adaptation.

VI. Conclusion and Future Work

The development of cloud-based applications that are composed of services offered by distinct cloud providers is a hard task due to the inherent heterogeneity of cloud environments. The selection of the proper cloud services that fit the application needs is based on cloud-related information, which is used to triggering an adaptation process. In our previous work [3], we introduced a strategy that enables to continuously monitor the dynamic properties of the cloud services that are required/used by an application. If there is any change on the values of such properties that affects the requirements of the product already deployed, a *dynamic adaptation process* is triggered in order to make the application redeployment. In this paper we presented an adaptation strategy that relies on DAOP to encapsulate the dynamic adaptation (removal and/or insertion of services) and easily change the application by dynamically removing a service and inserting a new one. The MAPE-K control loop enables to better define the phases of our strategy and a centralized knowledge management provides proper inputs to such phases. In addition, we are able to describe, in the feature model, the points in the application that must be intercepted to support the adaptation process. Moreover, it is necessary to provide a way for dynamically weaving aspects into the application and then changing it, based on the monitored values. For this purpose, we used the JBoss AOP framework, which has a rich *pointcut* expression grammar.

In future works, we intend to improve the algorithm that selects the best product to be deployed since it still generates and evaluates all possible products, which can be prohibitive in case of a larger SPL. Another important issue is to consider historical information in our strategy, for instance, how the cost of a specific adaptation affects a future adaptation. Finally, we also intend to evaluate other dynamic strategies and frameworks for handling dynamic weaving and dependency injection since the JBoss AOP project is not suitable for environments with runtime restrictions.

Acknowledgments

This work was partially supported by Brazilian Academic and Research Network (RNP) through the AltoStratus Project, and by the National Institute of Science and Technology for Software Engineering (INES)[1] funded by CNPq under grant 573964/2008-4. Thais Batista is partially supported by CNPq under grant 485935/2011-2, and Flavia Delicato and Paulo Pires are also partially supported by FAPERJ and CNPq (grants 311363/2011-3, 470586/2011-7, 310661/2012-9).

[1] http://www.ines.org.br

References

[1] P. Clements and L. Northrop, Software product lines: Practices and patterns. USA: Addison-Wesley, 2001.

[2] E. Cavalcante et al., "Exploiting software product lines to develop Cloud Computing applications", Proc. of the 16th International Software Product Line Conference, vol. 2. USA: ACM, 2012, pp. 179-186.

[3] A. Almeida et al., "Towards an SPL-based monitoring middleware strategy for Cloud Computing applications", Proc. of the 10th Int. Workshop on Middleware for Grids, Clouds, and e-Science. USA: ACM, 2012.

[4] T. Würthinger et al., "Improving aspect-oriented programming with dynamic code evolution in an enhanced Java virtual machine", Proc. of the 7th Workshop on Reflection, AOP and Meta-Data for Software Evolution , 2010.

[5] T. Dinkelaker et al., "A dynamic software product line approach using aspect models at runtime", Proc. of the First Workshop on Composition and Variability. USA: ACM, 2010.

[6] K. Kang et al., Feature-oriented domain analysis (FODA) feasibility study. Technical report, Software Engineering Institute, Carnegie Mellon University, USA, 1990.

[7] R. Filman et al., Aspect-Oriented Software Development. USA, Addison-Wesley, 2005.

[8] M. Mezini and K. Ostermann, "Variability management with feature-oriented programming and aspects", Proc. of the 12th ACM SIGSOFT Int. Symp on Foundations of Software Engineering.: ACM, 2004, pp. 127-136.

[9] E. Figueiredo et al., "Evolving software product lines with aspects: An empirical study on design stability", Proc. of the 30th Int. Conf. on Software Engineering. USA: ACM, 2008, pp. 261-270.

[10] J. Oldevik, "Can aspects model product lines?", Proc. of the 2008 AOSD Workshop on Early Aspects. USA: ACM, 2008.

[11] S. Apel and D. Batory, "When to use features and aspects? A case study", Proc. of the 5th Int. Conf. on Generative Programming and Component Engineering. USA: ACM, 2006, pp. 59-68.

[12] J. Zhang et al., "The role of aspects in software product lines", Proc. of the 2008 Int. Conf. on Computer Science and Information Technology. USA: IEEE Computer Society, 2008. pp. 588-592.

[13] S. Soares et al., "Distribution and persistence as aspects", Software: Practice & Experience, vol. 36, no. 7, 2006, pp. 711-759.

[14] R. Mietzner et al., "Variability modeling to support customization and deployment of multi-tenant-aware Software as a Service applications", Proc. of the 2009 ICSE Workshop on Principles of Engineering Service Oriented Systems. USA: IEEE Computer Society, 2009, pp. 18-25.

[15] L. Seinturier et al., "A component-based middleware platform for reconfigurable service-oriented architectures", Software: Practice and Experience, vol. 42, no. 5, 2012, pp. 559-583.

[16] JBoss AOP Framework: http://www.jboss.org/jbossaop

[17] Google Guice: http://code.google.com/p/google-guice/

[18] Amazon Web Services (AWS): http://aws.amazon.com/

[19] HP Cloud Services: http://www.hpcloud.com/

[20] H. Gomaa et al, "Dynamic software adaptation for service-oriented product lines", Proc. of the 15th International Software Product Line Conference, vol. 2. USA: ACM, 2011.

[21] L. Baresi et al., "Service-oriented dynamic software product lines", Computer, vol. 45, no. 10, 2012, pp. 42-48.

[22] A. Vasseur, "Dynamic AOP and runtime weaving for Java: How does AspectWerkz address it?", Proc. of the 2004 Dynamic Aspects Workshop, pp. 135-145.

[23] Y. Dai et al., "QoS-driven self-healing Web service composition based on performance prediction", Journal of Computer Science and Technology, vol. 24, no. 2, 2009, pp. 250-261.

[24] P. Leitner et al., "Monitoring, prediction and prevention of SLA violations in composite services", Proc. of the 2010 IEEE Int. Conf. on Web Services. USA: IEEE Computer Society, 2010, pp. 369-376.

[25] An architectural blueprint for Autonomic Computing, 4th ed. IBM, 2006 (Autonomic Computing White Paper).

A Machine Learning Based File Archival Tool

Robert Carreras, Du Zhang and Jinsong Ouyang
Department of Computer Science
California State University
Sacramento, CA 95819-6021, USA

***Abstract*— Managing a cloud file share is time consuming and costly. The task of routinely archiving stale files from the share is very difficult for IT professionals. In this paper, we report a machine learning based tool called MLFAT for file archiving. MLFAT provides a mechanism to make the task of identifying and archiving files easy and convenient through the following process: (1) an IT professional assigns categories with associated retention values for files and folders in the network share; (2) a knowledge-based bootstrap engine then propagates the category information for all the files according to the hierarchical structure, paving the way for supervised learning; (3) a support vector machine learner generates a decision function through train data on a subset of the file system; (4) the decision function is then utilized to generate archive or not archive labels for files/folders; and (5) IT professional will make the final decision based on the classification results. MLFAT is not based on any specific hardware system, can be deployed in a variety of file systems than current archival tools, and provides the ability to run discovery queries on files. Preliminary experimental results of the tool are promising.**

Keywords- file archiving, cloud file share, knowledge-based bootstrap, support vector machine.

I. INTRODUCTION

In this paper, we report a machine learning based tool for file archiving. The tool is called MLFAT (for Machine Learning File Archival Tool). The primary goal of MLFAT is to reduce an organization's monthly cost of renting or maintaining a large cloud file share. MLFAT uses a support vector machine learner to generate archival labels (archive, not archive) for files and folders in an organization's file share, and an IT professional of the organization will determine if the archive labels are correct. End users will be able to access archived files via a client-server model.

Reducing the size of an organization's network share will decrease the monthly rental cost for the file share. For a moderate sized Enterprise level cloud file share (500 GB) the monthly cost is $339 [5]. A low cost file storage medium like Amazon S3 with capacity of one TB filled to 500 GB costs $70 a month with an additional charge of $41.50 for file redundancy [1]. The difference in monthly cost is due to the access methods and allowances of the storage mediums. The high cost storage allows two TB of downloads per month, which is included in the file share cost. Amazon's S3 charges per file request and for data transfers are beyond the first gigabyte.

The primary focus of this paper is on describing the development of the knowledge-based ingest engine, the machine learning component for generating archival labels, and the graphical user interface (GUI) component. Due to space limitation, we will not discuss issues pertaining to file storage, file data collection, and file retrieval.

The rest of the paper is organized as follows. Section II gives some background information on related file archiving tools. In Section III, we describe the design of MLFAT. Section IV provides some initial performance evaluation of the tool. Finally Section V concludes the paper with remarks on future work.

II. RELATED WORK

File archival tools primarily focus on reducing the complexity and increasing the ease of identifying and archiving files from a company file system. Major components in archival tools often include an ingest engine (a mechanism used to assign pertinent meta data to files entering the system), a classification engine (a mechanism tasked with determining the archival label), and a discovery tool (a mechanism focused on providing specific information related to litigation issues). End to end systems also include mechanisms for retrieval of archived files by end users.

Full featured commercial archival tools are available on the market. Some of them include IBM Smart Archive [6], Symantec Enterprise Vault [11], and Dell's file archiving software [4]. The end-to-end archival products made available through IBM, Symantec and Dell all provide the full infrastructure for a company to seamlessly access archived data using fully developed enterprise tools.

IBM Smart Archive makes use of Content Collector software that manages an organization's e-mail, file systems, SharePoint content and other file data [6]. The Content Collector software also provides deduplication functionality. The Content Collector acts as a common ingest point for an organization's data, making classification and archival decisions easier. The Smart Archive system makes use of the Watson DeepQA technology to provide an analytics-based discovery and archival procedure [6].

Symantec Enterprise Vault File Archival System provides Content Control, Classification, Archiving, Retention management and Discovery [11]. It manages the types of content on a file server by maintaining a list of allowed file extensions and deciding if the file type can be saved or archived. File classification occurs during file creation. When a user creates a file in the system, its classification is determined by the retention policy of its parent folder [11]. In Symantec's system all data resides in a hierarchical folder structure, allowing archival labels to be assigned. Retention management makes use of a special folder type called retention folder,

which allows the creator to specify the length of file retention. The system can be used to identify files for legal discovery, external requests, or internal investigation. The discovery mechanism is provided by indexing files. The archiving process in the system uses a filter system based on attributes such as file type, last access time, age, size, or free space on volume, which IT staff determine [11].

Dell's E-mail and File Archival solution provides a company with the option to maintain unalterable data archives [4]. Another advanced Archival system integrates a global parallel file system and a standard backup/archive product with an innovative parallel software code to construct a scalable and parallel archive storage system [3].

Compared with the related work, MLFAT provides an ingest engine in the form of the knowledge-based bootstrap engine, a classification engine through a support vector machine (SVM) learner, and a discovery mechanism with the combination of the output files from the bootstrap engine and the underlying relational database. With similar functionality as the current major commercial products used for file archival, MLFAT is not based on any specific hardware system, and can be deployed in a variety of file systems.

III. DESIGN OF MLFAT

MLFAT is a modular tool to determine the archive label of files and folders. Figure 1 shows its structure. The core components are comprised of the MLFAT Learner, MLFAT Knowledge-based Bootstrap Engine, and the graphical user interface (GUI). Additional components include: File Crawler, Client, File Record Server, and Storage Server. In the rest of this section, we focus our attention on the core components and the file record database.

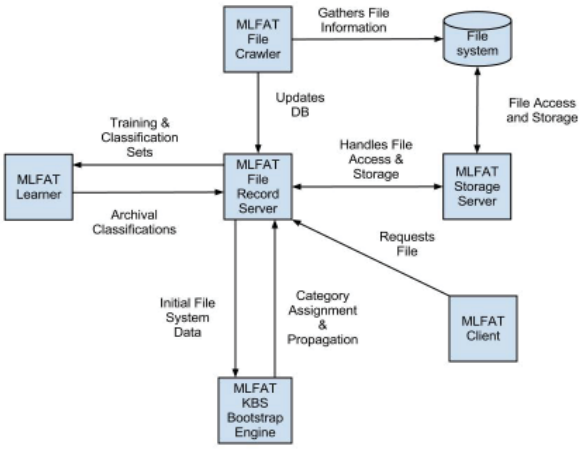


Fig. 1. MLFAT structure.

A. MLFAT File Record Database

The MLFAT File Record Server uses five database tables to maintain all current data and statistics.

The *file_system* table stores file and folder information that includes: id, path, name, type, size, last modified, archived status, parent, file extension, and label source. The *category table* contains the name and retention value for each respective category created by the user and has three attributes, *idcategory*, *cat_name*, and *retention_age*. The *file_category* table includes the assignment of a category entry to a file_system entry. It contains three attributes, idfile_category, idcat, and idfile.

The *learner_stats* table holds the statistical results of each machine learner instance. It contains six attributes learner_id, true_positive, false_positive, true_negative, false_negative, and date. Finally the *file_scratch* provides a scratch table that temporarily holds backup values from file_system. The backup attributes are idfile_system, archived, and label_src.

B. MLFAT Knowledge-Based Bootstrap Engine

The knowledge-based bootstrap engine (KBBE) allows for the information on file category and retention limit for that category, specified by the IT professional, to be properly disseminated across the file system. Special categories "root" and "subroot" will set inheritance start points. KBBE uses SWI-Prolog's engine [12] as its inference engine and take advantage of a runtime generated Prolog file as its Knowledge Base and Working Memory. The bootstrap mode in KBBE is comprised of three phases, Fact Gathering, Inference, and Propagation.

Fact Gathering – This phase consists of an IT professional creating categories and assigning them to file system entries. Assignments also include the special categories Root and Subroot. These assignments are stored in tuples associated with the specific file or folder they correspond with in a relational database.

Inference – This phase will consist of applying the inference rules to the facts generated in fact gathering and saving the outcomes in a relational database. There are following inference rules:

- root(X). This rule determines the root of the file tree represented in the knowledge base. An example root would be the parent directory of an entire file system, like C:\.
- subroot(X). This rule determines the locations within the file tree that should not inherit categories from their ancestors.
- parent(X, Y). This rule specifies the parent relationship, with X being the parent of Y. The parent relationship specifies that X is the parent folder in which Y resides. An example parent relationship: *parent*: C:\My_Folder, *child*: C:\My_Folder\My_File.txt.
- cat(L, C). This rule specifies which category, L, is assigned to a child, C. C can either be a folder or a file in the file system.
- ancestor(A, C) :– parent(A, C), not(subroot(C)), not(root(C)). This ancestor rule is used to determine if a child, C, should inherit the categories that have been assigned to A, its Ancestor. This rule first checks if A is the parent of C. If A is the parent of C, it makes sure that C is not a subroot or a root because subroot and root entries do not inherit from their parents and have no ancestors.
- ancestor(A, C) :– parent(P, C), ancestor(A, P), not(subroot(C)), not(subroot(P)). This rule allows the determination of an ancestor to C from any number of prior generations (folder levels).
- cat(L, A, C) :– ancestor(A, C), cat(L, A). This rule allows

for the inheritance of a category, L, from any ancestor, A, of the child C.

Propagation – In this phase, the rules generated in inference propagate the categories throughout the file system. This step entails the inheritance of every category assigned to a child's ancestors. After KBBE finishes the propagation, the category information is stored for each affected file or folder.

C. MLFAT Learner

Once files receive their category labels through KBBE, the SVM learner component is used to generate a decision function that predicts if a file f needs to be *archived* or *not archived* based on when f was last modified/accessed and the average retention value for the file category of f. The learner has two phases, Training and Classifying. Based on the size of the file_system table, the learner will determine the sample size for the Training phase. Both phases use the third party library, DLIB, a C++ machine learning library for SVM [7]. A DLIB matrix provides the container for each individual sample used in Training and Classifying. Sample initialization occurs by assigning negative one, to all locations.

For each category that is associated with a sample, the value one is written into the location that matches its idcategory plus the offset of the initial attributes.

Diff_time is calculated by subtracting the last_modified value from the current date. diff_time is rounded to whole years, and avg_ret is the average retention_age of all categories associated with the particular sample. All of the samples are stored in a vector of DLIB matrixes. For each sample, the archival label is stored in a separate vector.

Training – This is the first phase of the machine learner. It makes use of the sample and label vectors. The training phase begins with the sample passing through a normalization function, which normalizes using the mean and standard deviation [7]. The sample and labels are then randomized to remove any ordering that might have occurred during sampling. An SVM nu trainer, which makes use of the sequential minimal optimization algorithm (SMO) [7] as implemented in LIBSVM [2], takes both vectors and outputs a decision function.

Classifying – This is the second phase of the machine learner. It makes use of the decision function to a classification set that is built using the same method as the training set. Each instance of the classification set is passed into the decision function, resulting in the output of a real number. Negative values denote a false classification, while positive denote a true classification. The outputs are transformed to exactly positive one (Archive) or negative one (Not Archive) by the MLFAT learner component. The results of the decision function are presented to the user who analyzes them for correctness. The file_system entries for the classification set are then updated to match the new archival labels and label source information.

D. MLFAT Graphical User Interface

The GUI is a tabbed program, with the following 5 tabs. The first tab lists all the categories in the system, with their associated retention ages in tabular form. It also provides the mechanism to create new categories. Special categories, root and subroot are included due to them being stored in the same database table as the user generated categories. Currently this tab only provides the mechanism to create categories, but not modify them. This tab covers part of the Fact Gathering phase, by allowing the creation of Categories and persisting them to the relational database. The second tab lists all files and folders in the system in tabular form. It provides a check box interface for root and subroot, and a combo box that lists all available categories that have not been assigned to a particular file or folder. The user is able to select root or subroot and select an entry from the combo box, when the user clicks the Save File-Category button, the selected categories will be persisted to the relational database. This tab completes the Fact Gathering Phase. The third tab is about KBBE that provides two tabular views and two buttons, Build KBS and Propagate KBS. The first view is a dynamically generated list of categories that apply to a selected row in the other view, which displays all files and folders in the system. To use this tab, the user must first click the Build KBS button, which queries the relational database for all root(C), subroot(C), cat(L, C) and parent(P, C) data, and then appends the ancestor and other cat rules into a file. After the Build KBS button is clicked the Propagate KBS button becomes available, upon clicking, the Prolog engine consults the built file. After the file is loaded into the Prolog Engine, all new cat(L, C) facts are persisted to the database, using both the KBS and queries on the relational database to determine if the cat(L, C) facts are new to the relational database. This tab implements both the Inference and Propagation phases.

The fourth tab, the learner, provides three tabular views and three buttons. The buttons consist of Run Learner, Commit Changes, and Clear Changes. The first view is a dynamically generated list of all the file_system entries in the classification set. The second view presents the pre 'run learner' values for archived and label source for a particular row of the first view. The third view displays all categories and their retention ages for a particular row of the first view. To use this tab, the user must first click the Run Learner button. This calls and runs the MLFAT Learner component. Once the learner is complete, the three views are useable and the user may correct the archived and label source values of the first view. Once the user completes the required changes, they can click Commit Changes, which stores the new values and stores the statistical information for the learner instance. Instead of clicking Commit Changes, the user may click Clear Changes, to revert the system to the prior state. The fifth tab, Learner Statistics, provides one tabular view, one graphical view, and one button. The provided button is Refresh Statistics, this allows the tabular view of learner statistics to contain the results of learner instances generated while the program is open. The graphical view is a graph of the receiver operating characteristic (ROC) curve of the selected learner instance. QCustomPlot, a Qt plotting widget, generates the graphical view and plots the ROC point for the selected instance [14].

IV. PERFORMANCE EVALUATION

Performance of the Archive label is based on four possible confusion matrix values: true positive, true negative, false positive, false negative [13]. All of the confusion matrix values will be determined and recorded for each learner instance. Recording these statistics allows the ROC curve and precision to be calculated for each instance of the learner. The current

system stores all four confusion matrix values, but only plots the ROC curve, with the True Positive rate on the y-axis and the True Negative rate on the x-axis.

Learner instances run against the initial development sample data, with average retention age as the sole retention policy factor, have consistently plotted above the fifty percent line of the ROC curve. The learner instances tend to contain approximately one third true positive and two thirds true negative, which are indicative to the sample data set. Instances run using ad-hoc retention policies initially have decreased accuracy, but additional learner instances have increased accuracy as the sample data begins to reflect the ad-hoc retention policy. Each learner instance executes in a reasonable amount of time, with the response time being a few seconds.

Performance of the knowledge-based bootstrap engine is evaluated by response time. When it processes small datasets, the response time of the application is excellent, with no noticeable delay. However, larger datasets have noticeable response time when propagating the categories throughout the file system. There also appears to be redundant calculations done by the bootstrap engine when it searches for all categories that a child can inherit from its ancestors. In order for a child to inherit a category, only a single ancestor must possess the category, but the bootstrap engine exhaustively checks all the ancestors of the child to see if they have the category. However, in small datasets, this redundant checking does not reduce performance

The machine learner sample is based on the total file system size, but in the case of small file systems, it trains on two thirds and classifies one third. In the case of large file systems, the sample size chooses a small subset to prevent the classification set from being overly large. The current date determines the relationship between average retention age and the difference of current date and last modified. The sample learner instance trains on the training set, and then classifies the classification set.

The Machine Learner GUI tab presents the classification results once the learner has been run. Each entry in the classification set is displayed with the learner's prediction of Archive (1) or Not Archive (–1). When the user selects an individual entry, the initial archival state and associated categories are displayed. This display helps the user decide if the learner is correct. The user then makes the final determination on archival status and corrects the entries where the learner is incorrect. The manual process of correcting the learner enables the tracking of the confusion matrix values.

The user would then compare the average retention age, or any other retention policy, with the last modified date and the current date to determine if the learner's prediction is correct. The user can correct errors to match the retention policy.

V. CONCLUSION

The task of managing a network file share is time consuming and costly. The task of routinely archiving stale files from the share is very difficult for information technology professionals. This task is not simple and products that assist in it often require special hardware, special file systems, or advanced learning technology, which all come at a cost. Additionally most of these commercial systems require specific system setup conditions.

In contrast, MLFAT provides a mechanism to make the task of identifying and archiving easy and convenient. Using a combination of knowledge-based approach and machine learning, MLFAT is able to assist IT professionals in this task, regardless of the physical hardware their network runs on. Sample learning instances of MLFAT indicate that it provides accurate labels in a reasonable response time.

Future work includes: Category Tab needs improvement, mostly, with the addition of the ability to change the retention age associated with a given category. The MLFAT should also provide the ability to setup the relational database, and its user on first run. It currently requires the database to be configured separately. MLFAT should also provide the ability to load datasets. A discovery tool should also be included to allow users to select a particular category and have it output a list of all files associated with the category.

REFERENCES

[1] Amazon.com Simple Storage Service (S3). Oct. 10, 2011. http://aws.amazon.com/s3/#pricing

[2] Chang, C. and Lin, C.. LIBSVM: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1--27:27, 2011

[3] Chen, H., Grider, G., Scott, C., Turley, M., Torres, A., Sanchez, K., Bremer, J., "Integration Experiences and Performance Studies of a COTS Parallel Archive System," *IEEE International Conference on Cluster Computing*, 2010, pp.166-177.

[4] Dell. "E-Mail and File Archive Solution: Satisfy retention requirements and improve storage optimization", http://i.dell.com/sites/content/business/solutions/brochures/en/Documents/satisfy-retention-requirements.pdf, 2012

[5] Drive Headquarters: The First Cloud IT Solution Provider. Oct. 10, 2011, http://www.drivehq.com/premium/enterprisePricing.aspx.

[6] B. Dufrasne et al, IBM System Storage Solutions for Smarter Systems, July 2011, http://www.redbooks.ibm.com/redbooks/pdfs/sg247945.pdf.

[7] King, D. Dlib-ml: A Machine Learning Toolkit, *Journal of Machine Learning Research*, Vol.10, 2009, pp. 1755-1758.

[8] Microsoft: Visual Studio. Oct. 5, 2012, http://www.microsoft.com/visualstudio/.

[9] PostgreSQL: The world's most advanced open source database. Oct. 5, 2012. http://www.postgresql.org/

[10] Qt. Oct. 5, 2012. http://qt.digia.com/

[11] Symantec. Jan 18, 2012. "Symantec Enterprise Vault™ File System Archiving: Data Sheet: Data Protection", http://www.symantec.com/content/en/us/enterprise/fact_sheets/b-ev_for_fsa_DS_21197947.en-us.pdf.

[12] SWI-Prolog. Oct. 5, 2012. http://www.swi-prolog.org/

[13] Witten, I. and Frank, E., *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd Edition, Morgan Kaufmann, 2005.

[14] Works like Clockwork. Oct. 5, 2012, http://www.workslikeclockwork.com/index.php/components/qt-plotting-widget/

Modeling and Analyzing Attack-Defense Strategy of Resource Service in Cloud Computing

Huiqun Yu[1], Guisheng Fan[1,2], Liqiong Chen[3], Dongmei Liu[1]
[1]Department of Computer Science and Engineering
East China University of Science and Technology, Shanghai 200237, China
[2] State Key Laboratory for Novel Software Technology, Nanjing University 210093, China
[3] Department of Computer Science and Information Engineering
Shanghai Institute of Technology, Shanghai 200235, China

***Abstract*—In this paper, we propose a stochastic game theoretic method to model and analyze the attack-defense process in cloud computing. A stochastic game model (SGM) is proposed based on combining Stochastic Petri nets with game theory. The attack-defense behavior and their attributes are modeled by using SGM, thus forming the attack-defense game. On this basis, the *nash equilibrium* of attack-defense process of physical machine is computed in order to get the best defense strategy. The related theories of Petri net and the reachable states of attack-defense game are used to formally verify the correctness of proposed method. A case study shows that the proposed method adapts quickly to the changes in cloud application and thus improves the security of cloud computing.**

Index terms: Cloud computing, security, Game Theory, Stochastic Petri nets, attack-defense.

I. Introduction

Cloud computing sets a new paradigm for infrastructure management by offering unprecedented possibilities to deploy software in distributed environments [1]. Its goal is to share resources among the cloud service consumers and cloud vendors in the cloud value chain [2]. However, security issues present a strong barrier for users to adapt into cloud computing systems. First, cloud computing is inherently complicated, the provider must ensure that their clients' data and applications are protected while the customer must ensure that the provider has taken the proper security measures to protect their information. Secondly, the infrastructure that exists between the client and the cloud computing server, presents opportunities for other types of specifically targeted attacks. Finally, most attempts to validate security mechanisms and strategies, however, have been qualitative by showing that the process employed to construct a system is secure. But it can not be practically feasible to construct perfectly secure mechanisms and strategies, in face of complex and various attack behaviors in cloud computing.

Game theoretic methods allow to gain an in depth analytical understanding of the attack-defense process, which has been successfully applied to diverse problems such as Internet pricing and networking [3]. According to the Nash equilibrium, we could get the best defense strategy to against the attack strategy. However, some essential properties of game theory prevent it from further using in the field of cloud computing. For example, game theory has not enough modeling ability to describe interaction relations in the complex structure [4]. Therefore, formal methods can be used to analyze game process, thus increasing the semantic constraints. Petri net is a mathematically based technique for modeling and verifying software artifacts [5]. Which can be used to model and analyze attack-defense process of cloud computing.

In order to solve the security of cloud computing, some approaches [6,7] proposed enhancements for process models to express security requirements on a more accessible level. The approaches in [8] focus on algorithmic solutions in resource management. A game-theoretic model of an Infrastructure-as-a- Service (IaaS) cloud market, covering dynamics of pricing and usage, is targeted by [9]. An agent-based simulations on dynamic software pricing are proposed in [10] The work in [11] provides a discussion on advantages, drawbacks, and the future direction of using game theory. The authors in [12] proposes a scalable method for availability analysis of large scale IaaS cloud using analytic models. However, most of the current works on network security focuses on system models and their equilibrium analysis, the verification process is generally ignored. And they cannot capture the probabilistic nature of the state, which is more useful in analyzing attack-defense process in cloud computing. Furthermore, they don't consider the different defense strategies in face of various attack behaviors, the proposed method in this paper can prevent or to counteract attacks more efficiently.

In this paper, we propose a stochastic game model (SGM), which is used to describe the hierarchical structure and probabilistic nature of attack-defense behavior in cloud computing. The isomorphic *Markov chain* is computed by analyzing the relations between the states in attack-defense game model. The benefit of attack-defense strategy is also computed based on the actual mapping of state, the security attributes and cost of attack-defense behavior. The related theories of Petri net and the reachable states of attack-defense game model are used to formally verify the correctness of proposed method, which can make the cloud computing dynamically select the defense strategy to against attack behavior as quickly as possible.

The rest of this paper is organized as follows. In Section 2, we analyze the attack-defense requirements of cloud

computing. The attack-defense strategy is given in Section 3. In Section 4, we prove the correctness of proposed method. In Section 5, we show the execution process and simulation results by an actual example. Finally, Section 6 presents some related works while Section 7 is conclusion.

II. Attack-defence requirements of cloud computing

A. Attack-defence requirements

The attacker has a specific purpose when it takes any attack behaviors. In this paper, we use the preference of attacker on the security attribute to define the type of attack behavior, thus describing the purpose of attacker. Generally, the attacker has a strong attack purpose, it is denoted by the weight of security attribute. Physical machine can work normally only when it owns the confidentiality, integrity and availability, but the resource service may have the different security attributes. The loss of target resource can also be described by using security attribute. The cost of security attribute can be evaluated by the hazards that the attacker do to the target resource, which can also be denoted by the loss of three security attributes. First, we need characterize the requirement of attack-defense game in cloud computing.

Definition 1. *The requirement of attack-defense game in cloud application is a triple:* $\Xi = (BNS, RL, PRB)$*:*

(1) BNS *=*$\{RS, IT, AT\}$*, where RS, IT, AT are the finite set of physical machines, attack behaviors and defensive behaviors. Stop attacking "Null" is also an attack behavior.* AT *is the set of available defensive strategies. The system doesn't take any defensive behavior* (α) *is also a strategy.*

(2) $RL : C \times C \rightarrow \{>, +, \|, n\}$*, where* $>, +, \|, n$ *are the sequence, choice, parallel and loop relation between behavior. Let* $\forall x \in IT \cup AT, ForT(x) = \{y|RL(y, x) => \}, BacT(x) = \{y|RL(x, y) =>\}$

(3) $PRB = \{AW, Sq, AP, ITe\}$ *is the attribute function of cloud computing,* $AW : RS \cup IT \rightarrow [0, 1] \times [0, 1] \times [0, 1]$ *is the attribute function of physical machine and attack behavior,* $AW(rs_i) = (C_1, C_2, C_3)$ *is the preference of physical machine on the integrity, confidentiality and availability, and* $C_1 + C_2 + C_3$ *= 1.* $AW(it_i)$ *is the weight of attack behavior on each security attribute.* $Sq : IT \times AT \rightarrow [0, 1]$ *is used to describe the blocking rate that defensive strategy dose to the attack behavior.* $AP : IT \cup AT \rightarrow [0, 1]$ *is the cost that the system invokes the attack or defensive behavior, the total attack capacity of attacker is 1, similarly, the total defensive capacity of defender is 1.* $AP(Null) = AP(\alpha) = 0$*.* $ITe : IT \rightarrow (0, 1]$ *is the average arrival rate of attack behavior.*

B. Syntax and semantics of stochastic game model

Petri net is a formal language for describing the concurrency system. We will propose a stochastic game model based on Petri net, and introduce the related concepts, the remaining concept can refer to [5].

Definition 2. *A five tuple* $\Sigma = (PN, IO, tr, pr, M_0)$ *is called a Stochastic Game Model(SGM): (1)* $PN = (P, T, F, W)$ *is a basic Petri net.* P, T, F, W *are the finite set of places, transitions, arcs and weights. (2)* $IO \subset P$ *is a special type of place, which is the interface of* Σ *and denoted by dotted circle. (3)* $tr : T \rightarrow R \times R \times R \times R$ *is the attribute function of transition,* $tr(t_i) = (\lambda_i, \pi_i, \xi_i, r_i)$ *represents the response rate, the cost, benefit and priority of* t_i*, the default value is (*∞*, 0, 1, 0). (4)* $pr : p \rightarrow [0, 1] \times [0, 1] \times [0, 1]$ *is the weight of transition. (5)* M_0 *is the initial marking of* PN*.*

$\forall x \in (P \cup T)$, we denote ${}^\bullet x = \{y|y \in (P \cap T) \wedge (y, x) \in F\}$ and $x^\bullet = \{y|y \in (P \cap T) \wedge (x, y) \in F\}$. Let $S = (M, UD, IC, AC)$ is a state of SGM, where M is a marking, UD is the weight of place under M. IC, AC are the attack cost and defense cost. $UD(S, p_i)$ is the weight of place p_i under S. The initial state $S_0 = (M_0, UD_0, IC_0, AC_0)$, $UD_0 = \{pr(p)\}, IC_0 = AC_0 = 0$. $\forall t_i \in T, \forall p_j \in^\bullet t$, if $M(p_j) \neq \varnothing$, then transition t is enable under S. All the enabled transitions under S are denoted by set $ET(S)$. For $t_i \in ET(S)$, if t_i meets the condition: $r_i \leq min(r_j)$, where $t_j \in ET(S)$, then the firing of t_i under S is effective. All effective firing transitions under S are denoted by $FT(S)$.

Let $S = (M, UD, IC, AC)$ be a state of Σ, $\forall t_i \in FT(S)$, where λ_i, π_i are the response rate and cost of t_i, the system will reach a new state $S = (M', UD', IC', AC')$ by firing t_i, denoted by $S[t_i > S'$, S' is called a reachable state of S. M', UD' are computed based on the following rules:

(1) Computing the marking: $\forall p_j \in^\bullet t_i \cup t_i^\bullet, M'(p_j) = M(p_j) - W(p_j, t_i) + W(t_i, p_j)$

(2) If $\lambda_i = \infty$, then:$\forall p_j \in t_i^\bullet, UD'(p_j) = UD(p_j) - \xi_i$, $IC' = IC$, $AC' = AC + \pi_i$

(3)$\lambda_i \neq \infty$, then:$\forall p_j \in t_i^\bullet, UD'(p_j) = UD(p_j) \times \xi_i$, $IC' = IC + \pi_i$, $AC' = AC$

The reachable state is dynamically changing according to the feasible replacement of transition in the model *SGM*. If there is a firing sequence $t_1, t_2, \ldots, t_k$ and state sequence $S_1, S_2, \ldots, S_k$, which make $S[(t_1, \omega_1) > S_1[(t_2, \omega_2) > M_2 \ldots S_{k-1}[(t_k, \omega_k) > S_k$, then S_k is a reachable state from S. All the possibly reachable states of S are denoted by $R(S)$ and $S \in R(S)$. $\delta(S_i, S_j)$ is the firing sequence from S_i to S_j.

III. Attack-defense strategy

A. Modeling cloud computing

The attack-defense model is the foundation of security management in cloud computing.

(1) Modeling the basic components:

The attack behavior it_j and defensive behavior at_j is extracted as $t_{j,h}^i$, $t_{j,h}^a$, where i and a explain that the transition is an attack or defensive behavior. We introduce the subscript h to describe the hth invocation of it_h or at_h. The response rate of $t_{j,h}^i$ is equal to the average arrival rate of it_h, the cost of transition is mapped into the actual cost of behavior.

(2) Modeling the basic relation, the corresponding model is shown in Fig.1. Let $RL(it_k, it_r) = +, RL(it_k, it_w) = \|$:

For the sequence relation between the attack behavior and defensive attack, if $RL(it_k, at_{f1}) = RL(it_k, at_{f2}) = \ldots = RL(it_k, at_{fn}) =>$, if the system can reach position $p_{k,j}^i$ after

the jth execution of attack behavior it_k. The specific model is shown in Fig.1(a). For the sequence relation between the attack behavior, if $RL(it_k, it_h) =>$, and the system can reach the set of defensive position $\{p^a_{f1,g}, p^a_{f2,l}, \ldots, p^a_{fn,t}\}$ when it takes the defensive behavior $\{at_{f1}, at_{f2}, \ldots, at_{fn}\}$ to against the attack behavior it_k. The specific model is shown in Fig.1(b).

The specific model for choice and parallel relation between the attack behavior are Fig1(c)-(d).

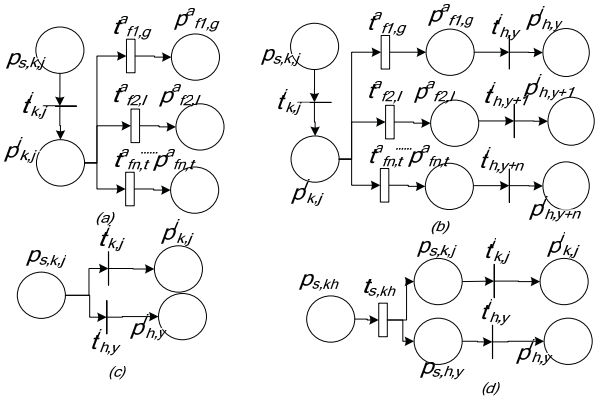


Fig. 1. The offensive and defensive model of basic relation

Let the adopted strategy of attacker and defender on physical machine rs be $sp^i_j(rs) = \{it_{j1}, it_{j2}, \ldots, it_{jm}\}$ and $sp^a_k(rs) = \{at_{k1}, at_{k2}, \ldots, at_{kn}\}$. The modeling process of attack-defense model $\Omega_{rs}(sp^i_j, sp^a_k)$ for rs is:

(1) If the physical machine is in the initial position, and $\exists it_{jy} \in sp^i_j(rs)$, $\forall it_{jh} \in sp^i_j(rs)$, $RL(it_{jh}, it_{jy}) \neq >$.

(2) The model library is used to model the set of behavior $\{it_{j1}, it_{j2}, \ldots, it_{jm}, at_{k1}, at_{k2}, \ldots, at_{kn}\}$, then construct the attack-defense model of physical machine according to the relations between behaviors and the interface of model(The system will merge the same place).

(3) The system will set the parameters for the place and transition in the model according to the attributes of behavior.

Let the requirements of attack-defense game be $\Sigma = (BNS, RL, PRB)$, and the adopted strategies of attacker and defender in the attack-defense process of cloud computing are $sp^i_j = \{sp^i_j(rs_1), sp^i_j(rs_2), \ldots, sp^i_j(rs|RS|)\}$ and $sp^a_k = \{sp^a_k(rs_1), sp^a_k(rs_2), \ldots, sp^k_a(rs|RS|)\}$, the modeling process of attack-defense model in cloud computing is: The system will construct the attack-defense model for each physical machine, and merge the attack-defense model of physical machine, then set the initial parameters for cloud computing.

B. Computing the benefit of physical machine

Definition 3. *Let $Markov$ chain MC be isomorphic to SGM model Ω, $|R(M_0)| = n$, $n \times n$ order matrix $Q = [q_{i,j}]$ is called the transfer matrix, where:*

$$q_{i,j} = \begin{cases} -\sum\limits_{t_k \in FT(M_i)} & \lambda_k: \quad if i = j \\ & \lambda_k: \quad else if \exists t_k \in T, M_i[t_k > M_j \\ & 0: \quad otherwise \end{cases}$$

$q_{i,j}$ describes the probability from state s_i to state s_j. The transfer matrix describes the reachable relation between the states. Let n stable states in MC be a row vector $X = (x_1, x_2, \ldots, x_n)$. We can get the stability probability $P[s_i] = x_i$ of each reachable marking. Stability probability can be used to predict the probability that the system reaches each state.

In order to accurately evaluate the cost of attack, we must quantify the type of attack and the extent of loss that the attack do for target resource.

Definition 4. *Let s_k be a stable state of $G_{rs}(sp^i_j, sp^a_k) = (S, E, pt, \Gamma)$, the physical machine rs may reach the states after firing the attack it_i under state s_k, $AT(it_i) = at_h$, then:*

(1) When rs reaches s_f by firing t_i, the possible loss $L(s_f)$ of security attribute is equal to:

$$L(s_f) = P[s_f] \times (v(s_f) + \sum_{s_j \in backS(it_i, s_f)} v(s_j) p[s_j])$$

$v(s_f)$ is the attribute function of rs under s_f, $backS(it_j, s_f) = \{sk | \exists t_j \in backT(it_i, s_f), s_f[t_j > s_k\}$.

(2) The revenue of $IR(it_i, s_f, rs)$ is equal to the benefits got by firing a successful attack. If $it_i = Null$ then $IR(it_i, s_f, rs) = 0$, else $IR(it_i, s_f, rs) = (AW(rs) - L(s_f)) \times (1, 1, 1)^T$. Where $(AW(rs)$ is the weight of physical machine on security attribute.

(3) The cost of $IC(it_i, s_f)$ is the required software and hardware when the attacker fires an attack and the corresponding legal sanctions after detecting the attack (which is a negative value), $IC(it_i, s_f) = -AP(it_i, s_f)$.

(4) The benefit of attack $IC(it_i, s_f)$ is the getting benefit (which is a positive value) when the system takes the defensive strategy at_h on physical machine rs, we can set $AR(at_h, s_f, rs) = IR(it_i, s_f, rs) \times sq(it_i, at_h)$.

(5) The benefit of attack $IN(IT_a, AT_h, rs)$ represents the benefit when the attack takes a series of attack behaviors $IT_a = \{it_{a1}, it_{a2}, \ldots, it_{am}\}$ on rs and the defender reaches the state set $R(IT_a)$ after takeing the set of defensive behavior AT_h: $IN(IT_a, AT_h, r_s) = \sum\limits_{s_f \in R(IT_a)} IR(it_{am}, s_f, rs) - IC(s_f)$. In the similar way, the benefit of defensive behavior is: $AN(IT_a, AT_h, r_s) = \sum\limits_{s_f \in R(AT_a)} AR(at_{am}, s_f, rs) - AC(s_f)$.

Definition 4 gives the benefit of a single attack behavior and defense behavior according to the actual meaning of the SGM's state. However, the best benefit of a single behavior can not guarantee the best benefit of cloud computing, otherwise, it is necessary to consider the cooperative relation between the behavior, the defense capability of defender.

C. Attack-defense game

The set of player's strategy $SP = (SP_i, SP_a)$ is the strategy of attacker P_i and defender P_a. $SP_i = (sp^i_1, sp^i_2, \ldots, sp^i_m)$ is the set of strategy of player P_i. SP is the tools and means used by the player in the game, each strategy set at least has two different strategies. Each strategy contains the deployment of

the player on all physical machines, the paly's behavior on the physical machine rs is denoted by set $SP(rs)$, such as, $sp_1^i = \{\{it_1, it_2\}, \{it_5, it_6, it_7\}\}$ is the invoked attack behaviors it_1 and it_2 on the physical machine rs_1. While the invoked attack behaviors of sp_1^i on physical machine rs_2 are it_5, it_6, it_7, and the system will not take any behaviors on the remaining physical machines.

In the process of attack-defense game, the utility of player is equal to the total benefit of all physical machines. UP_i is a function of $SP_i \rightarrow R$, which is the utility function of player and R is the value of utility. The benefit of strategy is equal to the total benefit of the strategy on all physical machines. From analyzing the computation formula of the benefit of strategy, we can get that the benefit of attacker is not only related with the attack strategy, but also is related with the defense strategy of defender. $\forall sp_f^i \in SP_i$, $\forall sp_j^a \in SP_i$, $U_i(sp_f^i, sp_j^a), U_a(sp_f^i, sp_j^a)$ are the utility of attacker and defender if the system takes the defense strategy sp_j^a for the attack strategy sp_i^i. $U_i(sp_f^i, sp_j^a) = \sum_{rs_k \in RS)} IN((sp_f^i, rs_k), (sp_j^a, rs_k), rs_k)$,

$U_a(sp_f^i, sp_j^a) = \sum_{rs_k \in RS)} AN((sp_f^a, rs_k), (sp_j^a, rs_k), rs_k)$.

The player's strategy in the attack-defense process is denoted by set $SP = (SP_i, SP_a)$, the utility of attacker and defender can be described by a matrix $U(SP) = [U_i(sp_f^I, sp_j^a), U_a(sp_f^i, sp_j^a)]$, the row of the matrix represents the possible attack strategy that may be used by attacker, and the column of the matrix represents the defense strategy that may be used by defender.

We use the zero and game to describe the game relation between attacker and defender. In the attack-defense game model, the attack and defensive strategy (sp^i*, sp^a*) is a *Nash equilibrium* if and only if strategy sp^i* is the best strategy for player i to against another player: $\forall sp^i \in SP_i$, $U_i(sp^i*, sp^a*) \geq U_i(sp^i, sp^a*)$ and $\forall sp^a \in SP_a$, $U_a(sp^i*, sp^a*) \geq U_i(sp^i*, sp^a)$.

The attacker and defender in game process can select their behavior according to the strategy (sp^i*, sp^a*) in *Nash equilibrium*. The best strategies of attacker in cloud computing and their corresponding defensive strategy can be computed by solving the following equation:

(1) $MinU_i(sp_f^i, sp_j^a)$

(2)$MaxU_a(sp_f^i, sp_j^a)$

(3) $\sum_{it_k \in sp_f^i)} AP(it_k) = \sum_{at_k \in sp_a^j)} AP(at_k) = 1$

The main purpose of computing and analyzing stochastic game model is to predict the stability probability of attack behavior in cloud computing.

IV. Analysis techniques

The attack-defense model in cloud computing is mainly used to model the attack-defense game process of all physical machines, we can analyze the correctness of SGM model by analyzing the properties of each physical machine. However, the size of state space can affect the analysis process of constructed model. Therefore, we will firstly analyze the state space of constructed model.

Because the transition in the attack-defense model can be executed after getting the required resource. Therefore, the attack-defense model of physical machine doesn't have the deadlock, the state space of $\Omega_{rs}(sp_j^i(rs), sp_k^a(rs))$ is finite. Similarly, the state space of attack-defense model of cloud computing is finite. So we can use the attack-defense model to analyze the related properties. $\forall at_g \in sp_k^a(rs)$, the corresponding transition of it_j, at_g on $\Omega_{rs}(sp_j^i(rs))$, $\Omega_{rs}(sp_k^a(rs))$ are denoted by set $T(it_j)$, $T(at_g)$.

Theorem 1. *The attack-defense model of physical machine rs is $\Omega_{rs}(sp_j^i(rs), sp_a^k(rs))$, S_0 is the initial state of model. $\forall it_j \in sp_j^i(rs)$, then $\forall t \in T(it_j) \cup T(at_g)$, $\exists S \in R(S_0)$, which makes $t \in FT(S)$.*

Proof: $\forall t \in T(it_j) \cup T(at_g)$, from the modeling process of SGM model $\Omega_{rs}(sp_j^i(rs), sp_k^a(rs))$, we can get that t is an execution of a behavior. We will prove the theorem by using the mathematical induction.

If $\forall it_k \in sp_j^i(rs)$, $> \notin RL(it_k, it_j) \cap RL(it_k, at_g)$, then $\forall S \in R(S_0)$, there is $t \in FT(S)$.

Otherwise, if $\exists it_k \in sp_j^i(rs) > \in RL(it_k, it_j) \cap RL(it_k, at_g)$, the forward attack and defensive transition of t are denoted by set $Fork(t) = \{t_{j1}^i, \ldots, t_{jm}^i, t_{j1}^a, \ldots, t_{jm}^a\}$, t_{jm}^i, t_{jm}^a are the forward and final attack or defensive behavior.

We can assume $\forall t_h \in Fork(t)$, $\exists S' \in R(S_0)$, which makes $t_h \in FT(S')$ establish. We will prove that $t \in FT(S)$ establishes when $\exists S \in R(S_0)$.

Therefore $\exists S' \in R(S_0)$, which makes $t_{jm}^a \in FT(S')$.

According to the operation semantics of SGM, we can get $\exists S_1 = (M_1, UD_1, IC_1, AC_1) \in R(S'), S'[t_{jm}^a > S_1$.

Because $t_{jm}^{a\bullet} = p_{jm}^a$, we can get $M_1(p_{jm}^a) = 1$.

According to the relation between t and the remaining behavior, it can be divided into four cases: (1)$\forall t_g \in sp_j^i(rs) \cap sp_k^a(rs)$, $RL(t_{jm}^i, t_g) \neq >$, and $RL(t_{jm}^i, t_g) =>$. (2)t is an attack behavior and $\exists it_t \in sp_j^i(rs)$, $RL(it_t, t) = +$. (3)t is a defensive behavior and $\exists at_g \in sp_a^k(rs)$, $RL(t_{jm}^i, at_g) =>$. (4) t is an attack behavior and $\exists it_t \in sp_j^i(rs)$, $RL(it_t, t) = \|$.

Case 1: Because $RL(t_{jm}^i, t) = RL(t_{jm}^i, t_{jm}^a) =>$, according to the modeling process of basic relation, we can get $p_{jm}^{a\bullet} = t \wedge^\bullet t = p_{jm}^a$, so $t \in FT(S_1)$, because $S = S_1$, the proposition establishes. In the similar way, we can get the proposition also establishes in other cases.

In summary, $\forall t \in T(it_j) \cup T(at_g)$, $\exists S \in R(S_0)$, which makes $t \in FT(S)$.

Theorem 1 illustrates that all behaviors are possible to be invoked in the attack-defense model $\Omega_{rs}(sp_j^i(rs), sp_k^a(rs))$ of physical machine. Similarly, the attack-defense model can also describe the choice, parallel and sequence relation.

Theorem 2. *The attack-defense model of physical machine rs is $\Omega_{rs}(sp_j^i(rs), sp_a^k(rs))$, S_0 is the initial state of model, $\forall it_j \in sp_j^i(rs)$, $\forall at_g \in sp_k^a(rs), RL(it_j, at_g) =>$, then $\forall t_{j,k}^i \in T(it_j)$, $\exists S \in R(S_0)$, which makes $t_{j,k}^i \in FT(S)$,*

then $S' \in R(S)$, $FT(S') \cap T(at_g) \neq \varnothing$.

Proof: Reduction to absurdity, $FT(S') \cap T(at_g) = \varnothing$. We can assume that $t^a_{g,h} \in (t^{i\bullet}_{j,k})^\bullet \cap T(at_g)$.

Because the transition of it_j, at_g on Ω are denoted by set $T(it_j), T(at_g)$, then $\forall t \in T(it_j) \cup T(at_g), \exists S \in R(S_0)$, which makes $t \in FT(S)$. Because $M(RS_{i,k\bullet}p_{ac}) = 1$, we can get $(RS_{i,k\bullet}t_c) \in ET(S)$.

Because the firing of transition will not take the time. Therefore, $\exists S_1 \in R(S)$, which makes $VET(S, RS_{i,k}) = RS_{i,k\bullet}t_c$.

Because $(RS_{i,k\bullet}t_c)^\bullet = \{RS_{i,k\bullet}p_{ch}, RS_{i,k\bullet}p_{cac}\}$, $^\bullet(RS_{i,k\bullet}t_c) = \{RS_{i,k\bullet}p_{uac}\}$.

According to the value of $M(RS_{i,k\bullet}p_c)$, there are two cases:

Case 1: $M(RS_{i,k\bullet}p_c) = 0$. Therefore, $M(RS_{i,k\bullet}p_{ch}) = ST_{i,j}$, and $\exists S_2 \in R(S_1)$, which makes $M_2(RS_{i,k\bullet}p^{IO}_{ou}) = 1$, that is $M_2(TK_{i\bullet}P^{i,k}_{ou}) = 1$. Because $TS(TK_{i\bullet}p_{tm}) > M(RS_{i,k\bullet}p_c), TS(TK_{i\bullet}p_{ac}) = 1$, we can get $M_2[TK_{i\bullet}t_{ou,k} >$. Because $^\bullet(TK_{i\bullet}t_e) = TK_{k\bullet}p_{we}$, we can get $\exists S_4 \in R(S_3)$, which makes $M_4(TK_{k\bullet}p_e) = 1$. Because $^\bullet(TK_{i\bullet}t_e) = TK_{k\bullet}p_{we}$, we can get $\exists S_4 \in R(S_3)$, which makes $M_4(TK_{k\bullet}p_e) = 1$. We can set $S' = S_4$, the proposition establishes.

In the similar way, we can get $M(RS_{i,k\bullet}p_c) \neq 0$, the proposition is also proven.

In summary, $\exists S' \in R(S), FT(S') \cap T(at_g) \neq \varnothing$.

Theorem 2 explains that the attack-defense model of physical machine can correctly describe the relation between the attack behavior and defense behavior.

V. Example and experiment simulation

We use a network topology to simulate the attack and defense process. The machine taking the attack behavior is in the external network, cloud application has 20 physical machines (10 public Web servers (rs_1 - rs_{10}), five file servers(rs_{11} - rs_{15}), five internal database servers (rs_{16}- rs_{20})). The firewall is used to separate cloud application from the external network. The rules of firewall are as follows. When target machine is Web server, service is HTTP or FTP then all source host's access strategy is allow. When target machine is file server, service is FTP then all source host's access strategy is allow. When target machine is database server, service is Oracle and FTP then Web and file source host's access strategy is allow. We can use the vulnerability scanning software to analyze the vulnerability of all physical machines, the host information and weaknesses. Vulnerability Attribute functio Web server, Apache chunked Enc,Wu-Ftpd SockPrintf(), (0.3,0.2,0.5) ,File server, FTP, rhost overwrite, (0.4,0.4,0.2),Database server, Oracle TNS Listener,Local buffer overflow (0.4,0.5,0.1)

The physical machine has the vulnerability and its dependency relation, the attacker can take the atomic attack.at1,Delete the threat of account and restart the httpd service,0.2 at2,Restore the node and delete the threat of account, 0.2 at3,Clear the virus and delete the threat of account, 0.1 at4,Install sniffer to monitor the program,0.2 at5,Clear the monitoring program of sniffer,0.1 at6,Delete the threat of account and re- start ftpd service,0.3 at7,Clear the threat of sniffer,0.2 at8,Null,0 The available defensive behaviors of defender are: httpd attack(it_1),ftpd attack(it_2),continue to attack(it_3),destroy the node and leave(it_4),install sniffer(it_5),get the access rights of information center(it_6) and network nodes(it_7),steal data(it_8),close node(it_9),dos(it_10),null(it_11). Their attributes are (0,0,0.1), (0,0,0.05), (0,0,0.05), (0.1,0,0.1), (0,0.1,0), (0,0.1,0), (0,0.15,0), (0,0.3,0), (0,0,0.2), (0.3,0,0), (0,0,0). Their cost are 0.2,0.2,0.1,0.2,0.2,0.2,0.2,0.3,0.2,0.5,0. Firing rate of these behaviors are 1,1.5,1.5,2,2,1.5,1.5,2,2,1.5,∞.

The relations between behaviors are: $ForT(it_3) = \{it_1, it_2\}$, $BacT(it_3) = \{it_4, it_5\}$, $BacT(it_8) = \{it_9, at_7\}$, $BacT(it_5) = ForT(it_8) = \{it_6, it_7\}$, $BacT(at_1) = \{it_1\}$, $ForT(at_2) = \{it_4, it_9, it_10\}$, $BacT(it_2) = \{at_4, at_6\}$,, $RL(it_1, it_2)$ = $RL(it_4, it_5)$ = $RL(it_6, it_7) = +$. $Sq(it_1, at_1)$ = $Sq(it_4, at_2)$ = $Sq(it_{10}, at_2)$ = $Sq(it_9, at_2)$ = $Sq(it_2, at_6)$ = $Sq(it_2, at_4)$ = $Sq(at_4, at_5)$ = $Sq(it_8, at_7)$=0.95,

According to the characteristics of attack behavior and defensive behavior, we can construct the attack-defense model $\Omega_2(IT, AT)$, which is shown in Fig.2. From analyzing the state space of Ω_2, we can get: (1) The state space of Ω_2 is finite, the state space of physical machine's model is also finite under a certain attack-defense strategy. (2) The model can correctly describe the attack behavior and defensive behavior and their relation. For example, $RL(it_9, at_2) =>$, the mapping of it_9 on Ω_2 is $\{t^i_{9,1}, t^i_{9,2}, t^i_{9,3}, t^i_{9,4}\}$, if Ω_2 reaches state S by firing transition $t^i_{9,3}$($M(t^i_{9,3})$=1), then the firing of transition $t^a_{2,2}$ under state S is effective.

We will randomly generate 15 attack-defense strategies. Because the generated strategy is large, we only give the attack strategy of rs_1, rs_2, rs_{11}, rs_{16}. First, we will analyze the change rules of different attack strategies on security attribute, which is shown in Fig.3, the X axis is the steps, Y axis is the revenue. From analyzing the figure, we can get that the change rules of security attribute on the different attack strategies are different for the same physical machine. The reason is that the revenue of attack behavior on security attribute is different. For example, the attack behavior $sp^i_{15}(rs_1) = \{2, 3, 5, 7, 8, 9\}$, so the revenue of attack behavior sp^i_{15} on rs_1 is (0,0.25,0.3). The change rules of the confidentiality of rs_1 with the time changing are shown Fig.4, the X axis is the strategy, Y axis is the revenue of security attribute.

VI. Conclusions

As a new distributed computing mode, cloud computing is different from the traditional distributed computing: loose organization, high scalability, heterogeneity, and so on. And all these make the attack-defense process under the cloud environment different from that under the traditional distributed computing environment. In this paper, we have proposed a stochastic game theoretic method to select attack-defense of resource service in cloud computing. A Stochastic game model is defined as a unified formalism to describe different components of cloud computing. The physical machine, the basic relation between the attack behavior and defense behavior are took into account in the modeling process, thus we can

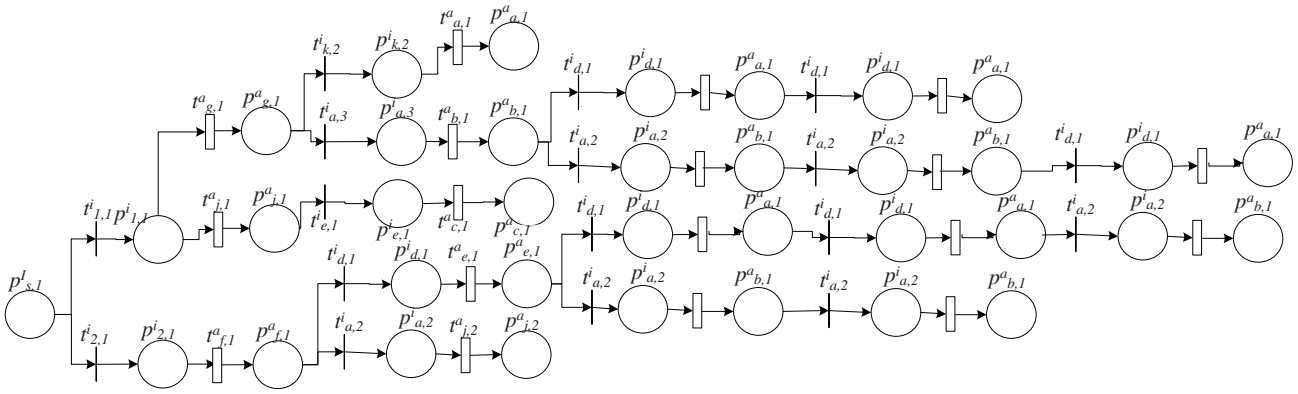


Fig. 2. The attack-defense model $\Omega_2(IT, AT)$

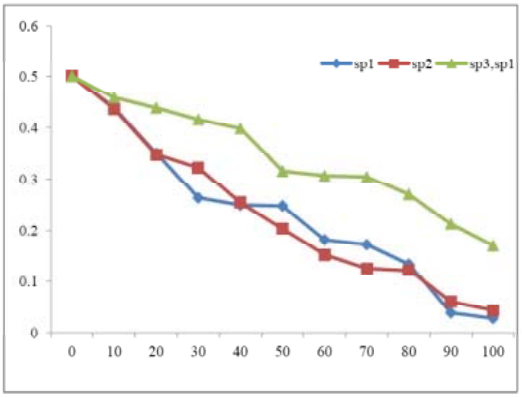


Fig. 3. Analyzing the security attributes of rs_1

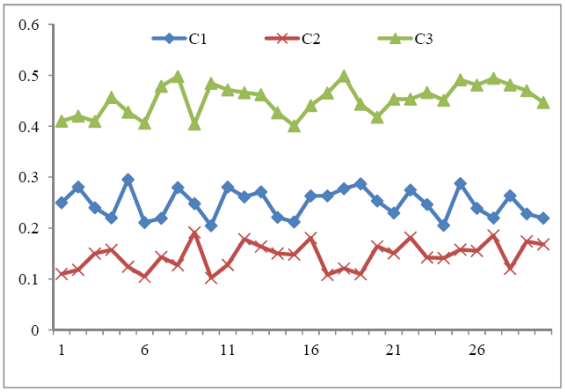


Fig. 4. Analyzing the confidentiality of rs_1

characterize precisely the attack-defense process. An attack-defense strategy is proposed, which is used to dynamically select the defense strategy to against the attack behavior according to the actual situations. The operational semantics and related theories of Petri nets help prove the effectiveness of attack-defense game process. Experiment results show that the proposed can effectively model and analyze the attack-defense in cloud computing.

ACKNOWLEDGMENT

The work is partially supported by the NSF of China under grants No. 61173048. Innovation Program of Shanghai Municipal Education Commission under Grant No.12YZ166. Construction Program of Shanghai Science and Technology under Grant No.12510503800. The Fundamamental Research Funds for the Central Universities.

REFERENCES

[1] S. Marstona, Z. Lia, S. Bandyopadhyaya, et al. Cloud computing-the business perspective. Decision Support Systems. 2011, 51(1): 176-189.
[2] C. Vecchiolaa, R. N. Calheirosa, D. Karunamoorthya, et al. Deadline-driven provisioning of resources for scientific applications in hybrid clouds with Aneka. Future Generation Computer Systems. 2012, 28(1): 58-65.
[3] E. Altman, T. Boulogne, R. El-Azouzi, et al. A survey on networking games in telecommunications. Computers and Operations Research. 2006, 33(2): 286 - 311.
[4] Y. Z. Wang, M. Yu, J. Y. Li, et al. Stochastic game net and applications in security analysis for enterprise network. International Journal of Information Security. 2012, 11(1): 41-52.
[5] M. TADAO. Petri nets: properties, analysis and application. Proceedings of the IEEE. 1989, 77(4):540-581.
[6] J. Juerjens. *UMLsec: Extending UML for Secure Systems Development*. Proceedings of the 5th International Conference on The Unified Modeling Language. LNCS 2460. 2002: 412-425.
[7] C. Wolter, A. Schaad. Modeling of Task-Based Authorization Constraints in BPMN. Processing of the 5th International Conference on Business Process Management. LNCS 4714, 2007:64-79.
[8] G. Wei, A. Vasilakos, Y. Zheng, et al. A aame-theoretic method of fair resource allocation for cloud computing services. The Journal of Supercomputing. 2009, 54(2):1-18.
[9] K. Jorn, H. Karl. A game-theoretical approach to benefits of cloud computing. Proceedings of the 8th international conference on Economics of Grids, Clouds, Systems, and Services. Springer-Verlag, 2011: 148-160.
[10] J. Rohitratana, J. Altmann. Agent-based simulations of the software market under different pricing schemes for Software-as-a-Service and perpetual software. Economics of Grids, Clouds, Systems, and Services. Springer-Verlag, 2010: 62-77.
[11] M. Manshaei, Q. Y. Zhu, Quanyan, T. Alpcan, et al. Game theory meets network security and privacy. ACM Computing Surveys. 2012, 45(3):1-45.
[12] F. Longo, R. Ghosh, V. K. Naik, et al. A scalable availability model for Infrastructure-as-a-Service cloud. Processing of the 2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks. IEEE Computer Society, Washington, DC, USA, 2011: 335-346.

Proposal and Validation of a Feasibility Model for Information Mining Projects

Pablo Pytel
PhD Program on Computer Science, UNLP - GISI UNLA - GEMIS UTN-FRBA, Buenos Aires, Argentina.
ppytel@gmail.com

Paola Britos
Information Mining Research Group. National University of Rio Negro at El Bolson, Argentina.
paobritos@gmail.com

Ramón García-Martínez
Information Systems Research Group, National University of Lanus, Remedios de Escalada, Argentina.
rgarcia@unla.edu.ar

***Abstract*— Information Mining projects are a special type of Software Engineering projects with the objective of extracting non-trivial knowledge from available data repositories. Information mining projects share similar problems with Software Engineering projects. Most of these problems should be handled at the initial activities of the project. But there is no model to analyze and evaluate the project feasibility which could be used to predict the main project risks. In this context, the objective of this paper is to propose and validate an ad-hoc model that can be used at the beginning of an Information Mining project in order to analyze its feasibility.**

Keywords-Feasibility Model; Information Mining; Software Engineering; Small and Medium-sized Enterprises.

I. INTRODUCTION

Information Mining projects are a special type of Software Engineering projects with the objective of extracting non-trivial knowledge which is located (implicitly) in the available data from different sources [1]. Commonly, instead of developing specific software, available software tools are used which include the necessary techniques and algorithms [2]. As a result, the features of Information Mining projects are different from Traditional Software Engineering projects and, also, from Knowledge Engineering projects, even though the algorithms are based on artificial intelligence methods [3]. The most used methodologies for Information Mining projects are CRISP-DM [4], SEMMA [5] and P^{3}TQ [6]. These methodologies are considered as proven by the community, but they exhibit problems when trying to define the phases related to project management [7]. Some elements of project management are mixed with project development process and tasks (such as project monitoring, verification and measurement) and others are not considered in the referenced methodologies.

In this context, it is detected the lack of a model to analyze and evaluate the project feasibility which could be used to predict the project risks. Therefore the objective of this paper is to propose and validate an ad-hoc model that can be used at the beginning of an Information Mining project in order to analyze its feasibility and then identify its strong and weak points. First, the problem is identified by analyzing the reasons of project failures (section II). Then the corresponding solution is proposed by specifying the feasibility model (Section III) which is validated using real project information (Section IV). Finally the main conclusions are presented (Section V).

II. ANALYSIS OF PROJECT FAILURE

Most Traditional Software Engineering projects can be considered (at least) partial failures because few projects meet all their cost, schedule, quality, or requirements objectives [8]. From the challenged or canceled projects, the average project was 189 percent over budget, 222 percent behind schedule, and contained only 61 percent of the originally specified features. In 2005, it has been considered that from 5 to 15 percent of projects were abandoned before or shortly after delivery as hopelessly inadequate [9]. In other words, few projects truly succeed. The most important reasons are, among others: inability to manage the project complexity, poorly defined system requirements, sloppy development practices, commercial pressures, unrealistic / unarticulated project goals and unmanaged risks.

Information Mining projects share similar problems. Conducted studies about Information Mining projects have detected that not all projects are successfully completed [10], ending most in failure [11]. In 2000, it has estimated that 85% of the projects have failed to achieve its goals [12]. In other words, this means that in average only 15 projects had been successfully completed from 100 developed ones. After five years working, the community has been able to decrease this project failure rate to approximately 60% [13]. Hence it can be said that the community is working on the right lane but there are still project elements that should be enhanced.

Most of these problems have to be detected in the initial activities of the project. Before starting any Traditional Software project, the organization normally decides whether it is appropriate performing it or not. Making such decisions is complex and depends on multiple factors as it is necessary to know both the impact of the software on the organization and the developing associated risks [14]. The project features are required to be analyzed by assessing the technical and economic feasibility of the project (commonly known as feasibility study). In Expert System development projects, something similar happens. As the initial specifications for these systems are often uncertain, incomplete, and inconsistent, it is necessary to develop several prototypes for coherently define the system functionality, performance, and interfaces

[15]. Since Knowledge Engineering (KE) projects use more resources than Traditional Software development projects [16], their feasibility study is highly more important in order to identify the risks that should be monitored and controlled during the project.

Information Mining project's initial tasks ought to be similar to a Traditional Software Development or KE projects. By early detection of the risks, its effects could be reduced during the project development. However, as the features of Information Mining projects are different from Traditional Software and KE projects, the models to study the feasibility cannot be reused for this type of projects and it is necessary to propose specific ones. In this domain, several studies have been conducted to identify success criteria [17-20], but there is no comprehensive model to analyze if the project is achievable or not. While in [21] a model that uses a fuzzy expert system has been proposed to measure the project success level from the quality used in each of its phases of CRISP-DM, this study may be performed once the project is completed because it requires the quality level applied in each phase to be known. On the other hand, in [22] a Bayesian analysis is used to determine whether the company is qualified to implement a data mining project (i.e. the enterprise characteristics are valued to decide whether data mining can be applied or not), but it does not consider important topics such as the business problem. Furthermore, this analysis does not make the classification of feasibility in different dimensions, considering it as a whole.

III. Proposed Feasibility Model

The proposal of the feasibility model for Information Mining projects requires the identification of the main conditions for considering a project as feasible (subsection A). Such a task is dependent on the project features which can be known in the initial stages. However, it is not usually easy to answer these conditions by answering 'yes' / 'no' questions (or by giving a numerical value). Thus the proposed feasibility model should be able to handle a range of linguistic values to answer each condition. From such values, and by applying a pre-defined process, it would be possible to determine the overall project feasibility (as detailed in subsection B).

This model has been based on the feasibility test defined for KE projects defined in [15, 16] which has been adapted and validated using actual Information Mining projects. These projects have been provided by researchers from the following research groups: GISI-DDPyT-UNLa, GEMIS-FRBA-UTN and GIEdI-UNRN. Be advised that all these projects had been performed by applying the CRISP-DM methodology [4] within Small or Medium-size Enterprises (SMEs). Therefore, the models can be considered reliable only for small and medium-sized Information Mining projects developed with this methodology.

A. Conditions

The main conditions are identified based on [17-24] and classified into three groups (or dimensions):

- *Conditions that determine the plausibility of the project* include the factors that make it possible to perform the information mining project. A project can be performed if the following conditions are met: the available data repositories have current and representative data of the business problem to be solve, the business problem is understood, and the team has a minimum knowledge about the information mining process.
- *Conditions that determine the adequacy of the project* include the factors that determine whether information mining is the appropriate solution for the identified business problem (i.e. it is the best solution for the problem). It is appropriate to apply information mining when the following conditions are met: the available data repositories have digital format (they are not only in paper), the business problem cannot be solved by using traditional statistical techniques, the business problem will not change during the project, and the data quality is good. The following metrics are used for assessing the data quality:
  - Quantity of attributes and records (measuring the availability of enough data to apply the data mining process).
  - Degree of credibility of the data (measures of how much you can trust on the data accuracy depending on the source and nature).
- *Conditions that determine the success of the project* include the factors ensuring the project accomplishment. An information mining project will be successful if the following conditions are met: data repositories are implemented with technologies allowing easy data access and manipulation (i.e. integration, cleaning, and formatting tasks), the project stakeholders (either high level managers, mid-level managers or end-users) support the project, it is possible to perform the project planning considering best practices with necessary required time, and the team has experience in similar projects.

B. Proposed Procedure

The following five steps are proposed to assess the project feasibility:

Step 1: Determining the value of each project features.
Looking for characterizing an Information Mining project and evaluating its feasibility, 13 features are used which are specified in Table I. Such features, based on the conditions identified in subsection A, should be answered from the interviews conducted with the organization stakeholders at the beginning of the 'Business Understanding' phase of CRISP-DM methodology. They should be valued by using one of the following words: 'nothing', 'little', 'regular', 'much', and 'all'.

For each feature the following attributes are defined:

- *Category*: used only to group the features according to what or who is concerned.
- *ID*: indicates a code to uniquely identify the property and the dimension to which it belongs (Plausibility, Adequacy, or Success).
- *Condition*: describes the question associated to the feature to be identified for characterizing the project.
- *Weight*: indicates the relative importance of each feature in the global model.

Note that features related to *plausibility* and *adequacy* dimensions must have a value equal to or bigger than 'little', otherwise the project can be considered as non-feasible. For *success* features, there is no minimum threshold (they can be valued with 'nothing').

TABLE I. PROJECT FEATURES EVALUATED BY THE MODEL.

Category	ID	Condition	Weight
Data	P1	How much actual is considered the data from the repositories?	*8*
	P2	How representative is considered the data in the repositories in order to solve the business problem?	*9*
	A1	How much the data repositories have digital format?	*4*
	A2	How many attributes and records are available in the data repositories?	*7*
	A3	How much credibility has the available data?	*8*
	S1	In which degree the repository technology supports the manipulation of the data?	*6*
Business Problem	P3	How much the business problem is understood?	*7*
	A4	In which degree the business problem cannot be solved by traditional statistical techniques?	*10*
	A5	How stable is considered the business problem during the project?	*9*
Project	S2	How much the stakeholders support the project?	*8*
	S3	In which degree the project plan considers the required time to perform best practices during the project?	*7*
Project Team	P4	How much knowledge has the team about information mining?	*6*
	S4	How much experience has the team in similar projects?	*6*

Step 2: Converting feature values into fuzzy intervals.
Once the linguistic values have been defined for each feature of Table I, they should be translated into numeric values to calculate the project feasibility. This transformation process is based on Fuzzy Expert systems [25]. For each word value, a fuzzy interval is specified that is expressed by four numbers (ranging from zero to ten) representing the breakpoints (or corner points) of the corresponding membership function. These intervals with the graphic representation of each membership function are shown in Figure 1.

Step 3: Calculating the value of each dimension.
To calculate the project dimension values, the fuzzy intervals (obtained in the previous step) are balanced considering their corresponding weight (already defined in Table I). The interval representing the value of each dimension (I_d) is calculated with the Formula #1 of Table II. This formula is formed by the combination of the harmonic mean and the arithmetic mean of the set of intervals (thus the influence of low values is reduced when calculating the dimension value). As a result of the formula, another fuzzy interval is achieved. To convert this interval into a single numeric value (V_d) the arithmetic average is used as specified in Formula #2 of the same table.

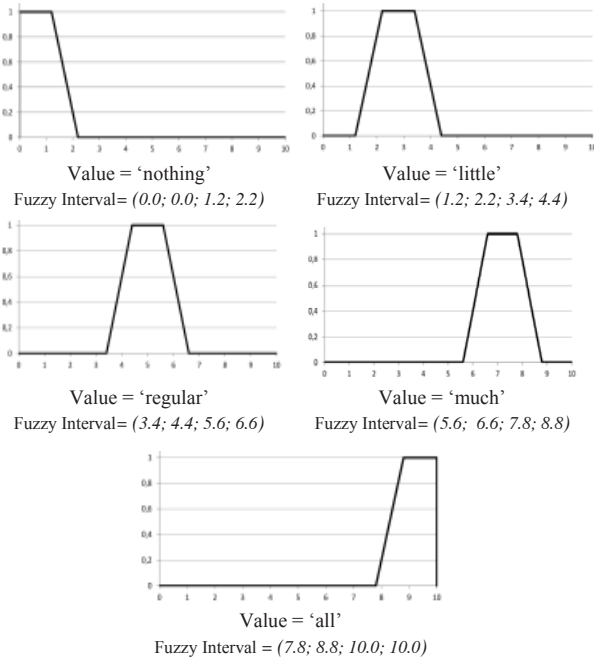


Figure 1. Membership function graphical and fuzzy interval assigned.

TABLE II. FORMULAS USED BY THE MODEL.

#	Formula
1	$$I_d = \left(\frac{1}{2} \cdot \frac{\sum_{i=1}^{n_d} W_{d_i}}{\sum_{i=1}^{n_d} \left(\frac{W_{d_i}}{F_{d_i}} \right)} \right) + \left(\frac{1}{2} \cdot \frac{\sum_{i=1}^{n_d} (W_{d_i} \cdot F_{d_i})}{\sum_{i=1}^{n_d} W_{d_i}} \right)$$ where: I_d: represents the fuzzy interval calculated for the dimension d (using '$_P$' for plausibility, '$_A$' for adequacy, and '$_S$' for success). W_{di}: represents the weight of the feature i for the dimension d. F_{di}: represents the fuzzy interval that has been assigned to the feature i for the dimension d. n_d: represents quantity of features associated to the dimension d.
2	$$V_d = \frac{\sum_{i=1}^{4} I_{d_i}}{4}$$ where: V_d: represents the numeric value calculated for the dimension d. I_{di}: represents the value of the position i of the fuzzy interval calculated for the dimension d .
3	$$OV = \frac{8 \cdot V_P + 8 \cdot V_A + 6 \cdot V_S}{22}$$ where: OV: represents the overall project feasibility value. V_P: represents the value calculated for dimension plausibility. V_A: represents the value calculated for dimension adequacy. V_S: represents el value calculated for dimension success.

Step 4: Calculating the overall project feasibility.
The numerical values calculated in the previous step for each dimension (V_d) are combined by using a weighted arithmetic mean (Formula #3 of Table II) obtaining the overall project feasibility value (OV).

Step 5: Interpreting the results.
Finally, the numeric values for each dimension and the overall project feasibility value (already calculated in steps 3 and 4 respectively) are analyzed. As a way to interpret the results of the feasibility of each dimension, it is recommended to plot the corresponding membership function of the obtained fuzzy interval (I_d). The feasibility of a dimension can be considered as accepted if it exceeds the range of '*regular*' value (that has been shown in Figure 1). Examining the numeric value of the dimension is another way to do it. If the dimension value (V_d) is greater than 5, the dimension can be considered as accepted.

On the other hand, for analyzing the feasibility of the project, the following criteria can be used: whenever the three dimensions are accepted and the overall project feasibility (OV) is greater than 5, then the project is considered as feasible. Otherwise, it is not feasible. In both cases, the project weaknesses to be strengthened should be recognized by identifying the dimensions with lower values.

IV. Validation of the Proposed Model

In this section the validation of the model proposed in Section III is performed using the data of 25 collected information mining projects. The first twenty-two projects (i.e. *P1* to *P22*) have been satisfactory finished (with some minor problems) but the last three (i.e. *P23*, *P24* and *P25*) have been cancelled before completion.

To perform this validation the projects values calculated by the proposed model are compared to an appraisal provided by researchers who can be considered experts in the domain. On one hand, the projects have been characterized by the paper authors using the model's features and applying the corresponding steps to calculate the project dimensions and the project feasibility value. Because of the paper's strict length, the results of applying the model in each project cannot be reproduced here but it can be found in [26]. A summary of these results is included in Table III.

On the other hand, a survey has been issued to each researcher to assess one project. The researcher had examined the project information (including the plan, meeting notes, status reports among other things) and indicate a value between 1 and 10 (where 1 is the lowest value and 10 the biggest) to appraise each project dimension (i.e. plausibility, adequacy, success). Then the global feasibility has been calculated as the average of them. The obtained values are shown in Table IV.

As soon as the previous values has been collected, the chart graphs of Figure 2 have been prepared to show graphically the comparison between the values appraised by the researchers (shown with a light gray line) and the values calculated by the model (dark gray bar) per dimension. As it can be seen the values are very similar but not totally equal. This can also be noted in the boxplot graphs included in Figure 3. These graphs reflect the behavior of the values assigned by the researchers (locate in the left part of the graph) and the ones calculated by the model (right part) indicating the minimum and maximum values (thin line), standard deviation range (thick line) and average value (marker).

TABLE III. PROJECT VALUES CALCULATED BY THE PROPOSED MODEL.

#	Value of Plausibility (V_P)	Value of Adequacy (V_A)	Value of Success (V_S)	Overall Project Feasibility (OV)
P1	*7.20*	*6.11*	*5.25*	*6.27*
P2	*6.87*	*5.07*	*5.25*	*5.77*
P3	*5.90*	*5.67*	*5.31*	*5.65*
P4	*5.12*	*6.95*	*4.12*	*5.51*
P5	*5.12*	*7.82*	*6.81*	*6.56*
P6	*5.45*	*5.61*	*5.25*	*5.45*
P7	*5.45*	*5.56*	*5.42*	*5.48*
P8	*6.45*	*5.80*	*5.18*	*5.87*
P9	*7.20*	*5.61*	*5.57*	*6.18*
P10	*5.85*	*5.34*	*5.57*	*5.59*
P11	*6.22*	*6.56*	*5.42*	*6.14*
P12	*7.67*	*7.35*	*6.45*	*7.22*
P13	*5.93*	*5.09*	*7.05*	*5.93*
P14	*6.20*	*6.59*	*5.69*	*6.20*
P15	*8.72*	*6.89*	*7.66*	*7.77*
P16	*6.45*	*6.43*	*5.64*	*6.22*
P17	*6.14*	*5.83*	*5.42*	*5.83*
P18	*6.00*	*5.31*	*5.42*	*5.59*
P19	*7.01*	*6.89*	*5.58*	*6.58*
P20	*8.24*	*6.75*	*5.52*	*6.96*
P21	*8.05*	*6.45*	*5.25*	*6.70*
P22	*6.45*	*5.81*	*6.54*	*6.24*
P23	*4.66*	*5.34*	*3.25*	*4.52*
P24	*4.66*	*3.46*	*4.21*	*4.10*
P25	*4.63*	*2.81*	*3.01*	*3.52*

TABLE IV. PROJECT APPRAISAL PROVIDED BY RESEARCHERS.

#	Plausibility Value	Adequacy Value	Success Value	Global Feasibility Value
P1	*8.00*	*7.00*	*4.00*	*6.33*
P2	*7.00*	*6.00*	*5.00*	*6.00*
P3	*8.00*	*5.00*	*6.00*	*6.33*
P4	*6.00*	*6.00*	*4.00*	*5.33*
P5	*6.00*	*8.00*	*7.00*	*7.00*
P6	*6.00*	*5.00*	*5.00*	*5.33*
P7	*5.00*	*5.00*	*5.00*	*5.00*
P8	*6.00*	*5.00*	*6.00*	*5.67*
P9	*7.00*	*6.00*	*6.00*	*6.33*
P10	*6.00*	*5.00*	*6.00*	*5.67*
P11	*8.00*	*5.00*	*6.00*	*6.33*
P12	*7.00*	*8.00*	*7.00*	*7.33*
P13	*7.00*	*5.00*	*6.00*	*6.00*
P14	*7.00*	*7.00*	*6.00*	*6.67*
P15	*9.00*	*7.00*	*8.00*	*8.00*
P16	*7.00*	*6.00*	*5.00*	*6.00*
P17	*6.00*	*5.00*	*5.00*	*5.33*
P18	*5.00*	*5.00*	*6.00*	*5.33*
P19	*8.00*	*7.00*	*7.00*	*7.33*
P20	*9.00*	*7.00*	*5.00*	*7.00*
P21	*8.00*	*6.00*	*5.00*	*6.33*
P22	*7.00*	*6.00*	*6.00*	*6.33*
P23	*3.00*	*4.00*	*3.00*	*3.33*
P24	*5.00*	*3.00*	*2.00*	*3.33*
P25	*4.00*	*2.00*	*1.00*	*2.33*

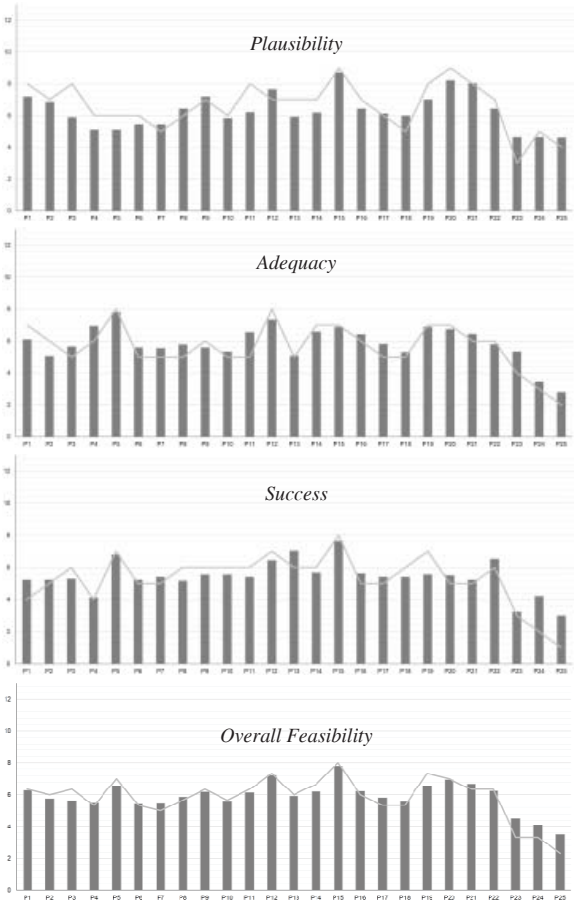


Figure 2. Comparison graph for each dimension

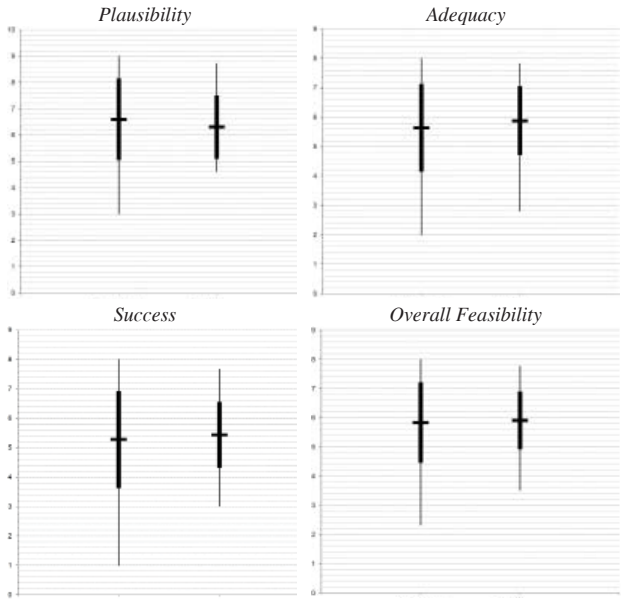


Figure 3. Boxplot graph for each dimension

As seen in the boxplot graphs of Figure 3, the model tends to be more conservative because the total range is shorter than the one assigned (particularly for the minimum values). But the standard deviation range and average values are almost the same (the bigger difference is lower than 0.30 for plausibility). Thus from this preliminary analysis it can be said that the model seems to be valid.

In order to assess the model, the Wilcoxon signed-rank test is applied [27]. This non-parametric statistical test allows to compare two related samples and define whether their population means differ (i.e. it is a paired difference test). It is an alternative to the paired Student's t-test when the population cannot be assumed to be normally distributed but there is a symmetric distribution of the differences around the median. In this test, each project dimension is handled independently. This means that for each dimension the values provided by the researchers are tested against the calculated by the model. The used null and alternative hypotheses are:

H_0: the valued assigned by the researchers and the values calculated by the model for each dimension have a median difference of zero (in other words, there are no meaningful differences between the researchers and the model values and they can be considered equivalent)

H_1: the median difference is not zero (i.e. the researchers and the model values are not equivalent)

The sums of signed-ranks generated by the application of the Wilcoxon test are shown in Table V for each dimension (where W^+ is the sum of all positive ranks and W^- is the sum of all negative ranks). All the auxiliary tables used in this test are available in [26].

TABLE V. RESULTS OF WILCOXON SIGNED-RANK TEST

Dimension	Sum Ranks$^+$ (W^+)	Sum Ranks$^-$ (W^+)	Quantity of non-zero pairs
Plausibility	*97*	*228*	*25*
Adequacy	*227*	*98*	*25*
Success	*175*	*150*	*25*
Overall Feasibility	*181*	*144*	*25*

The null hypotheses (H_0) is accepted or rejected based on comparison of the minimum sum of ranks (W) and a critical value extracted from the statistical reference table corresponding to the quantity of non-zero pairs and a level of significance. If W is lower than or equal to the critical value then the null hypotheses can be rejected (so in this case, it means that the model is not equivalent to the assessment of the researcher's appraisal). Otherwise the null hypotheses can be considered as valid (and, in this case, the model can be considered as equivalent). For all dimensions, there are not any zero-value pair, so the total of pairs is 25 (n=*25*). As a 0.01 level of significance is selected, the critical value is equal to 68. This value is then compared to the minimum sum of ranks for each dimension:

- For *Plausibility*, the minimum sum of ranks (W) is equal to *97* because W^+ is lower than W^-. As 97 is bigger than 68, the null hypotheses is not rejected and then it can be said that there is no meaningful differences between the

researchers and the model plausibility values and they can be considered equivalent.

- For *Adequacy*, the minimum value is of $W^- = 98$ which is also bigger than 68. This means that H_0 is not rejected and the model adequacy values are also valid.
- For *Success* happens something similar: $W = W^- = 150$ is also bigger than 68. This means that the success values are also significant.
- Finally, the *Project Overall Feasibility* values calculated by the model value can also be considered equivalent because $W = W^- = 144 > 68$.

Therefore it is confirmed that the proposed model has calculated values equivalent to the appraisal performed by the experts.

V. Conclusions

Information Mining projects are a special type of Software Engineering projects with the objective of extracting non-trivial knowledge from available data repositories. Conducted studies for these projects have detected that not all projects are successfully completed, ending most in failure.

This paper has the objective of proposing an ad-hoc model to be used at the beginning of Information Mining project in order to analyze its feasibility. Thirteen projects features (based on the project conditions) are defined and utilized in a procedure to calculate the project feasibility with three dimensions: *plausibility*, *adequacy* and *success*. As it is difficult to assign the features values at the beginning of the project, the proposed procedure considers using fuzzy intervals to calculate the project overall feasibility.

The proposed model has been validated using the information of 25 projects which have been appraised by expert researchers. A preliminary statistical comparison and the Wilcoxon signed-rank test have been applied. As a result it is found that the proposed model can estimate correctly the *plausibility*, *adequacy*, *success* and *overall feasibility* of the project in the initial steps.

Acknowledgements

The research reported in this paper has been partially granted by Research Projects 33A105, 33B102 and 33A167 within National University of Lanus, Research Project 40B133 within National University of Rio Negro, and Research Project EIUTIBA11211 within UTN at Buenos Aires.

References

[1] J. Schiefer, J. Jeng, S. Kapoor and P. Chowdhary, "Process Information Factory: A Data Management Approach for Enhancing Business Process Intelligence", Proceedings IEEE International Conference on E-Commerce Technology, pp. 162-169, 2004.

[2] R. García-Martínez, P. Britos, P. Pesado, R. Bertone, F. Pollo-Cattaneo, D. Rodríguez, P. Pytel and J. Vanrell, "Towards an Information Mining Engineering", Software Engineering, Methods, Modeling and Teaching, Sello Editorial Universidad de Medellín, pp. 83-99, 2011, ISBN 978-958-8692-32-6.

[3] R. Garcia-Martinez, P. Britos, F. Pollo-Cattaneo, D. Rodriguez and P. Pytel, "Information Mining Processes Based on Intelligent Systems", Proceedings of II International Congress on Computer Science and Informatics (INFONOR-CHILE 2011), pp. 87-94, 2011, ISBN 978-956-7701-03-2.

[4] P. Chapman, J. Clinton, R. Keber, T. Khabaza, T. Reinartz, C. Shearer and R. Wirth, "CRISP-DM 1.0 Step by step BI guide", Edited by SPSS, 2000. http://tinyurl.com/crispdm

[5] SAS. "SAS Enterprise Miner: SEMMA", 2008. http://tinyurl.com/semmaSAS

[6] D. Pyle, "Business Modeling and Business intelligence", Morgan Kaufmann, 2003.

[7] J. Vanrell, R. Bertone, R. García-Martínez, "Un Modelo de Proceso de Operación para Proyectos de Explotación de Información", Proceedings Latin American Congress on Requirements Engineering and Software Testing, pp. 46-52, 2012, ISBN 978-958-46-0577-1.

[8] L.J. May, "Major causes of software project failures", CrossTalk: The Journal of Defense Software Engineering, 11(6), pp. 9-12, 1998.

[9] R.N. Charette, "Why software fails", Spectrum, IEEE, 42(9), pp. 42-49, 2005.

[10] H.A.Edelstein and H.C. Edelstein, "Building, Using, and Managing the Data Warehouse", Data Warehousing Institute, Prentice-Hall PTR, EnglewoodCliffs (NJ), 1997.

[11] M. Strand, "The Business Value of Data Warehouses - Opportunities, Pitfalls and Future Directions". Ph.D. Thesis, Department of Computer Science, University of Skovde, 2000.

[12] U.M. Fayyad, "Tutorial report". Summer school of DM. Monash University (Australia), 2000.

[13] J.E. Gondar, "Metodología del Data Mining". Number 84-96272-21-4. Data Mining Institute S.L., 2005.

[14] R. Pressman, "Software Engineering: A Practitioner's Approach", Editorial Mc Graw Hill, 2004.

[15] R. García Martínez and P. Britos, "Ingeniería de Sistemas Expertos". Editorial Nueva Librería, 2004, ISBN 987-1104-15-4.

[16] A. Gómez, N. Juristo, C. Montes and J. Pazos, "Ingeniería del Conocimiento", Centro de Estudios Ramón Areces. S.A. (Madrid), 1997.

[17] J. Sim "Critical success factors in data mining projects". Ph.D. Thesis, University of North Texas, 2003.

[18] H.R. Nemati and C.D. Barko, "Key factors for achieving organizational data-mining success". Industrial Management & Data Systems, 103(4), pp. 282-292, 2003, doi:10.1108/02635570310470692.

[19] T.H. Davenport, "Make Better Decisions", Harvard Business Review, (November), pp. 117-123, 2009.

[20] U.Bolea, J. Jakličb, G. Papac and J. Žabkard, "Critical Success Factors of Data Mining in Organizations", Ljubljana, 2011.

[21] A. Nadali, E.N. Kakhky and H.E. Nosratabadi, "Evaluating the success level of data mining projects based on CRISP-DM methodology by a Fuzzy expert system", Electronics Computer Technology (ICECT), 3rd International Conference on Kanyakumari, Vol. 6, pp. 161-165, IEEE, 2011, doi:10.1109/ICECTECH.2011.5942073.

[22] G. Nie, L. Zhang, Y. Liu, X. Zheng and Y. Shi, "Decision analysis of data mining project based on Bayesian risk", Expert Systems with Applications, 36(3), pp. 4589-4594, 2009.

[23] L.L. Pipino, Y.W. Lee and R.Y. Wang, "Data quality assessment", Communications of the ACM, 45(4), pp. 211-218, 2002.

[24] N. Lavravc, H. Motoda, T. Fawcett, R. Holte, P. Langley and P. Adriaans, "Introduction: Lessons learned from data mining applications and collaborative problem solving", Machine learning, vol. 57, n.º 1, pp. 13-34, 2004.

[25] J.S.R. Jang, "Fuzzy inference systems", Upper Saddle River, NJ: Prentice-Hall, 1997.

[26] P. Pytel, "Datos Recopilados para Validación del Modelo de Viabilidad de Proyectos de Explotación de Información", Reporte Técnico GISI-TD-2011-01-RT-2012-02, Grupo de Investigación en Sistemas Información. Departamento de Desarrollo Productivo y Tecnológico, Universidad Nacional de Lanús (Argentina), 2012. http://tinyurl.com/valViab

[27] F. Wilcoxon, "Individual Comparisons by Ranking Methods", Biometrics 1, 80-83, 1945.

Decision Support for Re-planning of Software Product Releases

S. M. Didar-Al-Alam
Dept. of Computer Science
University of Calgary,
Calgary, Alberta, Canada
smdalam@ucalgary.ca

Guenther Ruhe
Dept. of Computer Science
University of Calgary,
Calgary, Alberta, Canada
ruhe@ucalgary.ca

Dietmar Pfahl
Institute of Computer Science
University of Tartu,
Tartu, Estonia
dietmar.pfahl@ut.ee

***Abstract*— In presence of ongoing change, re-planning of software release plans aims at making the plan a better fit with changing reality. Re-planning is potentially as complex as the original planning. Decisions to be made in this context are about when the re-planning should happen and which features should potentially be eliminated, added or replaced. This paper presents a decision support approach which combines the strengths of analytical methods with the intuition of human experts. Inspired by the idea of software process control, the approach assumes continuous monitoring of the amount of functionality implemented, the consumed effort, and the number of defects detected and corrected. The actual performance is compared with the planned one. In case of above-threshold deviation, a re-planning analysis is initiated for how to react in a best way on the changed situation.**

The decision support framework has been instantiated by existing methods for generating optimized roadmaps and their operationalization. The result is an operational method called Dyna-H2W. The method has been applied in a case study investigating the planning and re-planning for 50 features of a commercial word-processing software product. The results initially confirm the hypothesis that Dyna-H2W helps to create more valuable products.

Keywords-component; Software release planning, re-planning strategies, decision support, operational planning, strategic planning, case study.

I. INTRODUCTION

Release planning is the process of assigning features to different releases considering technological and business objectives and constraints. Re-planning is the process of modifying an existing release plan to better adjust the existing plan towards changing factors. In software development, nothing is as common as change. Re-planning is required to accommodate various types of changes in software development.

Re-planning is potentially as complex as the original planning which is considered a wicked problem [1]. Decisions to be made in this context are about when the re-planning should happen, how to perform the re-planning, and which features should potentially be eliminated, added or replaced by others. These decisions are potentially impacted by multiple factors. But neither human expertise nor model-based analytical methods in isolation are expected to be able to address the wicked nature and complexity of the problem. The paradigm of software engineering decision support is applied. It has been proven successful in situations when human decisions have to be made in complex, uncertain and/or dynamic environments.

Inspired by the idea of Statistical Process Control [2], we monitor the process to detect an 'out-of-control' situation and analyze re-planning actions. To support the actual re-planning decision to be made by the product or project manager, we propose a method called Dyna-H2W which utilizes optimization-based planning methods for generating optimized roadmaps and their operationalization. The method gives an answer to the *W*hen to re-plan question and suggests *H*ow and *W*hat to re-plan actions. Besides presenting design and giving justification of the new method, we instantiate the method with an illustrative case study demonstrating how Dyna-H2W detects an out-of-control situation requiring re-planning along with suggesting actions for improved re-planning decisions.

This paper is organized as follows. Section II presents related work. Section III discusses the background of this research. Section IV formulates the research questions and the solution approach. In Section V, the results of a case study are reported. Section VI presents a discussion of results and threats to the validity. Finally, Section VII presents conclusions along with plans for future work.

II. RELATED WORK AND RESEARCH QUESTIONS

A. Related Work

Release planning methods typically do not consider re-planning [3]. Thus, the standard way to handle feature change requests is to freeze change requests for one release period and consider them again at the beginning of the next release period [4]. However, McConnell identified nine deadly sins related to project planning [5]. These sins point towards the importance of initiating re-planning at appropriate time.

In the context of release planning, two major categories of re-planning methods exist, i.e., static and dynamic. While static re-planning considers re-planning in pre-defined intervals, e.g., when a new iteration starts, dynamic re-planning continuously monitors feature change requests and project development, and triggers re-planning only when certain conditions are met. While the *when to re-plan?*-

question has a predetermined value in static re-planning, it changes dynamically in dynamic re-planning.

AlBourae et al. proposed a static lightweight re-planning method [6] which compares old features with new incoming features using AHP (Analytical Hierarchy Process), and then selects the most promising features using a greedy approach. DynaReP [7] offers a re-planning solution at operational level through staff re-allocation.

The H2W-pred method [8] is the predecessor of the proposed method. The emphasis of H2W-pred is on the application of predictive models. H2W-pred is a static re-planning method exclusively considering effort and quality factors. In contrast to all the formerly mentioned re-planning methods, Dyna-H2W facilitates dynamic re-planning and utilizes multiple factors to initiate and control re-planning. While based on computationally efficient optimization components, the results are offered as decision support, thus guiding the decision maker and acknowledging that additional aspects and concerns can influence the actual decision to be made.

B. Research Questions

In this paper, we address three research questions (RQs). Their respective answers constitute the key structure of the proposed new decision support method Dyna-H2W:

- RQ1: How to provide decision support on detecting the appropriate time to initiate re-planning process?
- RQ2: How to provide decision support on performing dynamic re-planning in consideration of multiple factors?
- RQ3: How to integrate predictive models into the decision support provided under RQ1 and RQ2?

III. Background Methodology

Our proposed decision support method for performing re-planning of software product releases is based on the idea of performing process control in order to trigger the analysis of re-planning scenarios. In this section, we give a short description of the key underlying methodological components.

A. Strategic Release Planning

Strategic release planning (optimized feature selection) is the problem to decide about which set of features should be implemented in which of the upcoming release. We consider a set F of features. F consists of n features abbreviated as f(1), …, f(n). The strategic release planning process defines the assignment of features to at most one of the k (≥ 1) upcoming releases.

Definition 1: A *(strategic) release plan* defines the assignment of features to releases. It can be expressed by a vector x = (x(1) …x(n)). In its i-th component, the vector specifies for feature f(i) to which release it is assigned, with

$x(i) = k$ if feature f(i) is offered in release k (1)

$x(i) = 0$ otherwise (2)

In our decision support approach, we utilize an existing planning approach called EVOLVE II [9] and its implementation in a proprietary system called ReleasePlanner™ [10].

B. Operational Release Planning

Once the assignment of features to releases has been defined operational release planning can be done. Each feature is considered the result of performing a sequence of tasks. A specified pool of human resources, i.e., the developers, is available to perform different tasks. Solving the operational release planning problem means to assign developers to the tasks to be performed and to determine the appropriate scheduling of tasks in a way to minimize release time.

Definition 2: For a given (strategic) plan x_i, and for a given pool of developers, an (operational) release plan $y_i(x_i)$ defines the order of implementation of all tasks of the features in the subset F_1 of features of F assigned to the first release (k=1). In addition, it defines the assignment of developers to perform the implementation tasks creating the feature.

In instantiating Dyna-H2W, we utilize an existing planning approach called RASORP [11] and its implementation in ReleasePlanner™ [10].

C. Prediction Models

For applying the notion of process control, we are applying predictive models for both effort and number of defects. There is a variety of methods available for this purpose [12]. In terms of effort prediction, we have applied effort prediction method AQUA+ [13] and its customization towards release planning as described in [9].

Different methods are available for defect or fault prediction [14]. For defect prediction, we applied the case based reasoning approach presented in [15] and which has been empirically evaluated comprehensively.

Definition 3: For the implementation of a (operational) plan $y_i(x_i)$, $effort_{pred}(t,r_i,y_i)$ and $effort_{act}(t,r_i,,y_i)$ refer to the predicted (most recently) respectively actual effort consumption for resource r_i (i = 1... m; m being the number of different resource types in consideration) at time t.

Definition 4: For the implementation of a (operational) plan $y_i(x_i)$, $defects_{pred}(t,y_i)$ and $defects_{act}(t,y_i)$ refer to the predicted respectively actual number of detected and corrected defects at time t.

D. Re-Planning

Re-planning incorporates elements of both strategic and operational release planning. After the application of the strategic planning, a set of features is defined for development in the upcoming release. In the course of their development, different types of changes may occur. The question becomes to revise the existing plan in the best possible way to accommodate these changes.

IV. DECISION SUPPORT METHOD DYNA-H2W

A. Overview

The key structure of the proposed decision support method is illustrated in Fig. 1. Further details of the method are presented in the subsequent sections. A case study utilizing the method is presented in Section V.

The human experts in charge of making re-planning decisions are supported by continuously monitoring actual plan implementation. The plan and potential re-plans are generated from two optimization components. The project data base is continuously updated. New features arriving, changed actual efforts and deviations in the predicted quality are accommodated that way.

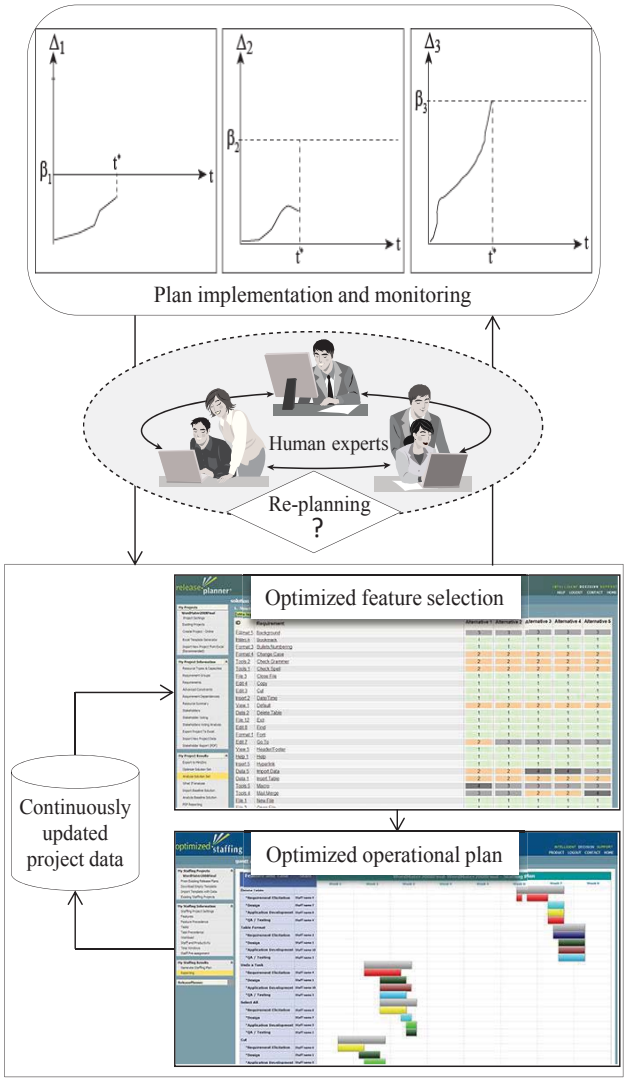


Figure 1. Workflow of Decision Support System Dyna H2W.

B. Plan Generation

Selection (and scheduling) of features has been formerly modeled as an optimization problem [9] The objective of this planning is to maximize a function representing the overall stakeholder satisfaction in consideration of existing resource, budget, time and technological constraints. As a result of this process, a set of five optimized alternative solutions is proposed. Out of the five optimized and maximally diversified solutions, human experts can select the one that in a best possible way addresses also their implicit concerns. The situation is illustrated in Figure 1 (upper part in the lower method component) where five alternatives are listed on the right-hand side as columns in the screenshot. The different numbers correspond to the assignments of the features (corresponding to rows) to releases 1, 2 or being postponed.

Once the feature set for the upcoming release are decided, operational planning of its implementation is initiated. Again, this process is formulated as an optimization process. As a result, an operational plan visualized by a Gantt chart is generated.

In case of any reason to consider re-planning (see Sections IV.E and IV.F for further details), both optimization steps are re-executed with updated data reflecting the changes in the actual project execution.

C. Plan Implementation & Monitoring

Three process parameters Δ1(t), Δ2 (t) and Δ3(t) (further explained below) are continuously monitored. These parameters refer to the deviation between actual process and predicted process performance related to amount of functionality (Δ1(t)), effort consumption (Δ2(t)) and defects detected and fixed (Δ3(t)), respectively. As soon as one of the factors indicates an out-of-control situation (in Figure 1 illustrated at t = t* for Δ3(t)), running re-planning analysis is initiated. As a result, and finally depending on the human expert, decisions are made whether things are changes and if yes, which ones. Details of the monitoring process are discussed next.

D. Out-of-control Situation

One of our major contributions is utilizing multiple factors in alarming out-of-control situation and deciding "When to re-plan". 'Out-of-control' situations are considered those situations that fall outside the expected/planned values (after considering common variation) and indicate existence of an assignable cause of variation. Three factors used in the proposed framework are described below.

1) Change request in functionality

For a given time interval [0, T] corresponding to the release period in consideration, the system is monitored for the relative amount of feature change requests. h(t) represents the percentage of features that are allowed to be replaced by new features at time t ($0 < t < T$). Openness to change is defined in Eq. 3, with α used as an adjustment parameter [9].

At each point in time t, the percentage of new features relative to the number of existing ones is expressed by function g(t). F_{curr} denotes the current set of features selected for the first release. $F_{new}(t)$ denotes the set of new features having arrived at time t ($0 < t < T$). In Eq. 4, card(F) denotes the number of elements (cardinality) of set F.

$$h(t) = (1- t/T)\ \alpha \text{ with } 0 < \alpha < 1 \quad (3)$$

$$g(t) = card(F_{new}(t)) / [card(F_{new}(t)) + card(F_{curr})] \quad (4)$$

Re-planning criterion RC-1: Out-of-control situation is triggered at time t = t*, if for the percentage of new features g(t*) becomes greater than the openness to change h(t*). For all such notifications experts will decide if re-planning is necessary.

$$\Delta 1(t^*) = g(t^*) - h(t^*) > 0\ (= \beta 1) \quad (5)$$

2) Effort

The operational plan $y_1(x_1)$ represents a schedule and an assignment to tasks for the development process of the current release period [0, T]. For each resource type r_i, a continuous monitoring process checks whether the absolute difference between actual development $effort_{act}(t,r_i)$ and predicted development effort $effort_{pred}(t,r_i)$ exceeds tolerance level β2.

Re-planning criterion RC-2: Out-of-control situation is triggered at time t = t* if

$$\Delta 2(t^*) = abs(effort_{pred}(t,ri) - effort_{act}(t, ri)) > \beta 2 \quad (6)$$

3) Quality

Similar to effort, process monitoring is done related to the number of defects detected and fixed. The assumption here is that a (reliable) initial defect estimation measure exists and that defect detection and correction can be monitored in a continuous way.

Re-planning criterion RC-3: Out-of-control situation is triggered at time t = t* if

$$\Delta 3(t^*) = abs(\ defects_{pred}\ (t) - defects_{act}\ (t)\ ) > \beta 3 \quad (7)$$

E. Re-planning

Once conducting re-planning, the final decision about whether a re-plan is improving the former plan is left for the human expert decision. Moreover, the human expert also has the flexibility to choose re-plan tools/techniques. This leaves three categories of choices with different intensity open to the expert i.e. (a) Manual re-plan, (b) Lightweight re-plan [6] (c) Optimization-based re-plan (with analytical planning tools like the one used in Dyna-H2W).

Project data and repositories are updated continuously. Human experts are permitted to re-adjust planning goals and acceptance thresholds β1, β2, and β3. Once re-planning has been performed, the whole process is applied again with the potential for another re-plan.

F. Detailed Process Steps

Finally, for detailed description of our method, and based on all notations introduced, we present the individual steps of how to proceed in Dyna-H2W. The assumptions are the same as for the overview description presented in Section IV.A. The whole process is subdivided into 10 steps as described below:

METHOD DYNA-H2W

Step 1: Start (t=0, i=1)
Step 2: Optimized feature selection (strategic planning)
Step 3: Selection of the most appropriate plan x_i (out of a set of optimized and diversified plans generated in Step 2)
Step 4: From the selected plan x_i, generate optimized operational plan $y_i(x_i)$ for the upcoming release.
Step 5: During implementation at any time t, monitor in respect of process parameters Δ1(t), Δ2 (t) and Δ3(t)
Step 6: As long as t < T and one of the re-planning criteria RC-1, RC-2 or RC-3 is fulfilled, then re-planning analysis is triggered. Go To Step 7. Otherwise Go To Step 10.
Step 7: Update project data in terms of features, actual effort consumption $effort_{act}$ (t, r_i, y_i) and actual number of defects $defects_{act}(t,y_i)$ detected & fixed.
Step 8: Apply predictive models for determining $effort_{pred}$(t, r_i, y_i) and $defects_{pred}(t,y_i)$
Step 9: i = i + 1 and Go To Step 2
Step 10: Stop

Expert involvement is of key relevance for performing the above process steps. In particular, they are involved in the decisions made in steps 3, 4, 6, 7, and 8.

V. CASE STUDY

A. Case Study Set-up

For illustration and initial evaluation purposes, a case study is described. For that, a real world text editor software development project is considered having 50 candidate features. The planning horizon is two releases. Each feature to be implemented is considered being the result of performing a set of related tasks. These tasks are devoted to:

- Requirements elicitation
- Design
- Application development
- Third party development and
- Quality assurance.

Ten developers were supposed working on this project. All the developers were characterized by a competence profile. These profiles reflect the degree of experience and expertise to perform the different candidate tasks. The estimated level of productivity of a developer is determined based on existing work experience. One way to support this evaluation process is to apply pair-wise comparison between developers for each competence area as outlined by the Analytic Hierarchy Process [16]. The detailed project data are available online at [17].

B. Re-planning Steps

Initially, a strategic plan x_1 is considered with a corresponding operational plan $y_1(x_1)$. EVOLVE II generates five optimized and diversified alternative plans (Step 2). Human experts choose plan x_1 suggesting the implementation

of 27 features in the upcoming release (Step 3). The fixed release length is T = 35 days.

Six new features are assumed to arrive randomly within this time period. From the selected plan x_1, an optimized operational plan $y_1(x_1)$ is generated for the upcoming release with aid of RASORP tool (Step 4). Effort distribution follows the optimized operational plan. For sake of simplicity, quality is reduced to defects. We further assume that the number of defects detected and fixed is directly proportional to the effort spent on quality assurance.

To control the conformance to the planned process, we continuously monitor re-planning criteria RC-1 to RC-3 during the release plan implementation process (Step 5). Tolerance levels are context specific and defined by domain expert as $\beta_1 = 0$, $\beta_2 = 0.4$ and $\beta_3 = 20$. In addition, for criterion RC-1, $\alpha = 0.2$ is defined by domain expert. It means according to Eq. 3 initially (t=0) openness to change is 0.2 (20% features could partially be replaced by new ones). However, with time this percentage will decrease.

In the course of re-planning, data related to the set of features might change (e. g., effort). In addition, the feature set itself might change. As a notational convention, D_0 denotes the project data for (i) effort consumed, (ii) defects detected & fixed, and (iii) accumulated feature value, as it was defined at time t = 0. Whenever a re-planning takes place (Step 6), then D_{t^*} denotes the set of (updated) project data at time $t = t^*$ (Step 7 resp. 8). Human experts play the key role here. With direct involvement of human expert in Steps 3, 4, 6, 7, and 8, the final decision is completely made by them.

C. Monitoring Development and Triggering Re-Plan

The details of running through the different re-planning steps are available from [17]. In what follows, we just report the key results. We consider RC-2 for each different resource type individually. At time t = 11, RC-2 related to *Requirements Elicitation* is violated (Step 6). This represents the first out-of-control situation.

Even though three new features arrived, the openness to change is too small as to violate RC-1. With the updated project data (Step 7, 8), revised strategic and operational plans x_2 (Step 3) and $y_2(x_2)$ (Step 4) are determined from using the respective optimization components of ReleasePlanner™.

At point in time t = 22, re-planning criterion RC-3 is violated. This results in another re-planning iteration. Strategic and operational plans x_3 and $y_3(x_3)$ are generated, respectively. At point in time t = 29, RC-3 is again violated. However, due to unavailability of sufficient resources, the human expert preferred to continue with current plan instead of re-planning.

D. Case Study Results

To evaluate the usefulness of the proposed method, we have performed a comparative analysis between the applications of Dyna-H2W against two other re-planning strategies being

i. dynamic re-planning with single factor consideration, and

ii. static re-planning with single factor consideration.

Considering *no re-planning* as the baseline, relative improvements of other strategies are presented in Figure 2. The comparison is in terms of the total release value achieved under the different re-planning strategies. Dyna-H2W achieved the maximum overall release (satisfaction) value. On day 22, substantial decrease in Dyna-H2W value compared to other strategies is observed. This is because some features completed their implementation with lower quality than the target value at this point in time. Although strategies (i) and (ii) considered these features as completed, Dyna-H2W could not do so because of the quality target being not fulfilled. Thus it shows a lower value at day 22, but by the end of the project Dyna-H2W achieved the maximum release (satisfaction) value along with higher quality compared to other strategies.

As can be seen from Table 1, Dyna-H2W was able to utilize the optimized resource allocation to detect more defects than all the other re-planning strategies (including baseline) as well.

TABLE I. COMPARISON OF RE-PLANNING STRATEGIES IN TERMS OF QUALITY

Number of defects detected and fixed			
Baseline (No re-planning)	**Dynamic re-planning with single factor**	**Static re-planning with single factor**	**Dyna-H2W**
412	407	399	422

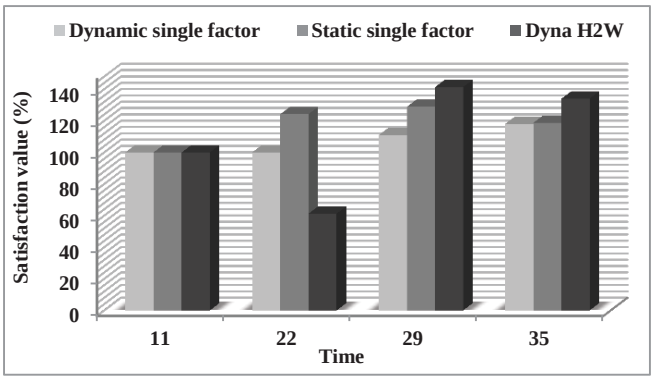


Figure 2. Comparison of three re-planning strategies in terms of value.

VI. DISCUSSION & THREATS TO VALIDITY

Our case study served as a proof-of-concept for the proposed approach. While running the case study, we have made some observations.

- We observed deviation in quality results in minimizing the overall release value. Features demand additional quality assurance resource to minimize quality deviation. Thus quality assurance becomes a bottleneck of the project and results in low release value.
- In such trade-off human expert needs to achieve balance between value and quality of the release.
- Additionally we witnessed dependency among multiple monitoring factors. Re-planning initiated by either one of the factors brings all deviations back to normal.

Thus it impacts the regular growth of deviation for all process parameters.

- Finally, we observe that Dyna-H2W achieves the best overall release (satisfaction) value compared to other strategies.
- From Table I, we observe Dyna-H2W performs best in achieving the quality target as well. Dyna-H2W detects & fixed highest amount of defects compared to all other strategies.

Case study results visibly illustrate achievements of our framework in terms of both value and quality. We cannot claim external validity of our results due to the simplifications made about the underlying assumptions. Main limitations are:

- We assumed that just quality assurance (QA) effort would be responsible for defect detection and debugging. In reality, additional effort might be required.
- The accuracy of Dyna-H2W depends on the accuracy of the prediction data and experts knowledge. While the effort, value and strategic plan data were taken from a real world data set, the QA data had to be synthetically generated.
- The tolerance levels of the re-planning criteria are considered constant for the release period. In a realistic situation, many reasons may require experts to re-adjust the tolerance levels.

VII. Conclusions and Outlook on Future Research

Accommodating different types of change is a vital issue in release planning. Instead of handling the related decisions in an ad hoc manner, we propose a new and systematic framework that integrates both human intuition and analytical methods to provide decision support. Proposed method ensures utilizing a perfect blend of human knowledge and software engineering tools. Initially, it has shown success in detecting appropriate timestamps when out-of-control situation occurred and re-planning decisions. A more comprehensive evaluation of the approach is still pending. In particular, we are planning to evaluate the approach under industry settings.

Another direction of future research is utilizing different combinations of factors to initiate out-of-control situations. It will allow experts to get the flexibility to choose factors of his choice & need from a set of available factors. Altering the definitions of the re-planning criteria, varying the threshold parameters used for initiating re-planning, and discovering the best combination of factors for utilizing proposed method are other areas of future research.

False alarms in monitoring process are still handled manually by human expert. Providing a decision support to recognize false alarms would be another future research. Finally, the overall effort of the re-planning needs to be improved. Currently, switching between strategic and operational planning is done manually, which is planned to be automated in a future release of the method implementation.

Acknowledgment

This research was supported by an Alberta Innovates Technology Future fellowship of the first author and the Natural Sciences and Engineering Research Council of Canada, NSERC Discovery Grant 250343-12 of the second author of the paper.

References

[1] H. Rittel and M. Webber, "Planning problems are wicked problems", Developments in design methodology, 1984, pp 135–144.
[2] W. Florac and A. Carleton, “Measuring the software process,” Addison Wesley, 1999.
[3] M. Svahnberg, T. Gorschek, R. Feldt, R. Torkar, S. B. Saleem, and M. U. Shafique, “A systematic review on strategic release planning models,” Information and Software Technology, vol. 52, 2010, pp. 237-248.
[4] K. E. Wiegers, “Software requirements,” Redmond, Microsoft Press, 2003.
[5] S. McConnell, "The nine deadly sins of project planning," IEEE Software, Vol. 18, 2001, pp. 5-7.
[6] T. Al-Bourae, G. Ruhe, and M. Moussavi, “Lightweight re-planning of software product releases,” in Proceedings IWSPM, 2006, pp. 27–34.
[7] A. Al-Emran, D. Pfahl, and G. Ruhe, “DynaReP: A discrete event simulation model for re-planning of software releases,” Proc. ICSP 2007, LNCS 4470, pp. 246–258.
[8] A. Al-Emran, A. Jadallah, E. Paikari, D. Pfahl, and G. Ruhe, “Application of re-estimation in re-planning of software product releases,” Proceedings ICSSP, 2010, pp. 260–272.
[9] G. Ruhe, “Product release planning: methods, tools and applications,” CRC Press, 2010.
[10] https://www.expertdecisions.com/
[11] A. Ngo-The, G. Ruhe, “Optimized resource Allocation for software release planning,” IEEE Transactions on Software Engineering, Volume 35, 2009, pp. 109-123.
[12] M. Jørgensen and M. Shepperd, "A systematic review of software development cost estimation studies," IEEE Transactions on Software Engineering, vol. 33, 2007, pp. 33-53.
[13] J. Li, G. Ruhe, “Analysis of attribute weighting heuristics for analogy-based software effort estimation method AQUA+,” Empirical Software Engineering Vol. 13, 2008, pp. 63-96.
[14] T. Khoshgoftaar and N. Seliya, “Fault prediction modeling for software quality estimation: Comparing commonly used techniques,” Empirical Software Engineering, Vol. 8, 2003, pp. 255–283.
[15] E. Paikari, M. M. Richter, G. Ruhe, “Defect prediction using case-based reasoning: An attribute weighting technique based upon sensitivity analysis in neural networks”, IJSEKE, Vol. 22, 2012, pp. 747-768.
[16] T. Saaty,“The analytic hierarchy process, planning, priority setting, resource allocation”, McGraw-Hill, New York, 1980.
[17] https://sites.google.com/site/didar522/data-repository.

A Non-Intrusive Process to Software Engineering Decision Support focused on increasing the Quality of Software Development

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

***Abstract* — The lack of quality in the production process of software development isn't attributed only to the techniques and technologies, but also to the lack of process that *management decisions*. Thus, this paper presents a process model for *Software Engineering Decision Support* focused on improving the quality of software development. Its preparation was based on areas and expected results of the process *Decision Management* present in the *Reference Model for Brazilian Software Process Improvement* (MR-MPS)[1]. In order to contribute to its understanding and use, it is presented a comparative study with other models present in the literature and identifies its benefits and problems with an application in two software development projects. The result of this process was a 78% reduction in rework and a 22% increase in performance of the team.**

Keywords - Decision Support; Analytic Hierarchy Process; Historical Database; Increase Quality of Software Development.

I. INTRODUCTION

During the software development lifecycle we can find a set of decisions that should be taken in order to *increase* product quality and / or respect any project restrictions imposed [1, 3, 6, 14]. Some of these restrictions can be seen in Fig. 1. But (*i*) what are the decisions that must be taken throughout the software development lifecycle? (*ii*) How these decisions affect the later stages and final product quality? (*iii*) How to make structured and tracked decisions throughout the software development lifecycle? (*iv*) And how to make these decisions not intrusive to the existing software development process?

Figure 1. Some restrictions that must be balanced in a project [1].

We will examine these issues in greater depth starting from the issue (*i*):

[1] Modelo de Referência para Melhoria do Software Brasileiro (MR-MPS)

A. *What are the decisions that must be taken throughout the software development lifecycle?*

Consider a software development process such as *Rational Unified Process* (RUP) [2] shown in Fig. 2. Several decisions must be made along each disciplines and iterations. For instance, on the discipline "Business Modeling" decisions as (*i*) which processes are the most urgent? (*ii*) What processes are at greatest risk? (*iii*) What are the core and support processes? And others may emerge early in the software development process. Its results will affect the other phases of the process [1]. This leads us to the second issue:

B. *How these decisions affect the later stages and final product quality?*

The next steps of the process will be affected since they use up the results of previous decisions to plan their executions [1]. Regarding the quality of the final product, the result will be a *very strong* relationship to the quality of the process [3]. Since decisions were made erroneous so there is a greater probability of final product to be a poor quality.

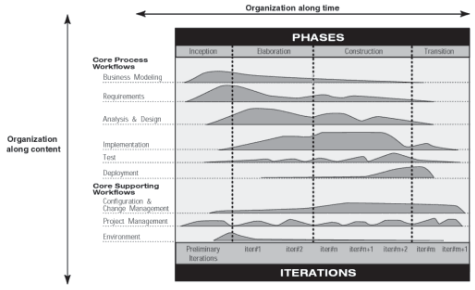


Figure 2. The Rational Unified Process (RUP)

C. *How to make structured and tracked decisions throughout the software development lifecycle?*

In an engineering work where most decisions are techniques [5], it should be structured and stored in a *Historical Database* (HDB). The decisions created and stored in HDB can be accessed and / or reused in the future, making the HDB in an organizational asset [1]. Last but not least:

D. How to make these decisions not intrusive to the existing software development process?

Despite the engineering, software projects are creative [6]. Extra *bureaucracy* can reduce the creativity of developers and / or create unnecessary overhead.

Considering what was previously exposed, the aim of this work is to present a model of process focused on the increase of the quality of the software development process. For its elaboration we based on the expected areas and results of the *Decision Management Process* presents in the *C* maturity level of MR-MPS [14]. For this we propose a non-intrusive process to support decision making in software engineering (NIPSEDS) using the method *Analytic Hierarchy Process*[2] (AHP) and a Historical Database (HDB) to address the issues *A*, *B*, *C* and *D*.

This article is divided in six elementary sections, including this introduction. In Section 2 we presented the related work and theory. In Section 3 we presented the process model to *Software Engineering Decision Support*. Section 4 we presented the validation of model through a case study. Section 5 we presented the results of the research. Finally, Section 6 we presented the conclusions and suggestions for future works.

II. THEORY

A. Analytic Hierarchy Process

The *Analytic Hierarchy Process* (AHP) was first proposed by Thomas L. Saaty [15] and its main characteristic is the *pairwise comparison* which consists of a *hierarchy of criteria and alternatives*. It is often used to analyze problems of decision-making multi-criteria. By using AHP, the structure of the problem must be decomposed into a hierarchy.

A hierarchy is a specific system based on the assumption that the entities can be grouped into disjoint sets with a group of entities which affects the other ones [15]. Pairwise comparison is an important component of the AHP. Two criteria are compared using a nine-point scale, where one (*1*) means "equal" importance, three (*3*) is "low" importance, five (*5*) "indicates" clearly "superior", seven (*7*) is "very" important and nine (*9*) denotes "extremely" important. With pair numbers being used to indicate intermediate values, if necessary. If there are *n* criteria to consider, *n(n-1)/2* comparisons of pairs had to be done. Thereafter, the reciprocal *nxn* matrix is constructed and weights are then obtained [11, 12].

The consistency of pair comparison matrix needs to be verified by means of the indexes: *Consistency Index* (CI) and *Consistency Rate* (CR). They are defined in equation (1) and (2) with λ_{max} being the principal value (Eigen) and *Random Index* (RI) is as shown in Table I. For consistency, CI and CR *must* be less than 0.1 for the AHP analysis is considerate acceptable [11, 12].

$$C.I. = (\lambda_{max} - n) / (n - 1) \quad (1)$$

$$C.R. = C.I. / R.I. \quad (2)$$

[2] Implemented through *Expert Choice*, http://expertchoice.com/

TABLE I. RANDON INDEX

n	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
R.I.	0	0	0.58	0.90	1.12	1.24	1.32	1.41	1.45	1.49

B. Related Work

In addition to recommendations of the MR-MPS guide [14], we found in literature some works that address issues related to management decisions during the software development lifecycle. In [7] the authors integrating the *Technique for Order of Preference by Similarity to Ideal Solution* (TOPSIS) and AHP into *Goal-Oriented Requirements Engineering* (GORE) in a *Decision Support System* (DSS) to produce a metric of choice among the best alternatives. However, this paper addresses only the *initial phase* of the project. In [8] the authors use a group decision technique only to the requirements phase. In [9] the authors use data mining techniques to software engineering decisions. This adds an *overhead* to the process. In [10] the authors document the decisions made throughout the software development lifecycle but without the concern in *structuring* decisions. This work presents a historical database. Table II compares related work with this model process.

TABLE II. RELATED WORK COMPARED WITH THIS MODEL PROCESS

Criteria	Related Work				
	[7]	[8]	[9]	[10]	This
Structured Decision	●	●	●		●
Decision Traced			●		●
During the Lifecycle				●	●
Historical Database			●	●	●
Non-Intrusive Process	●	○			●

● Strongly cares to ○ partially care to and without symbol don´t attends. These criteria support areas and expected results (from GDE1 to GDE7) of Decision Management Process of MR-MPS [14].

C. Reference Model for the Brazilian Software Improvement Process[3]

Developed in 2003 by the SOFTEX[4] as part of the MPS.Br[5] program, the MR-MPS consists of a reference model with the definition of prerequisites for the improvement of the quality of the software process. Besides it, the program is composed by an Assessment Method (MA-MPS) and a Business Model (MN-MPS), each one of them described by guides and/or document models.

In accordance with Capability Maturity Model Integration for Development (CMMI-DEV) and following the described headlines in its main program, this model was divided into seven maturity levels. These levels define steps to improvement processes in the organization [14]. Moreover, this division aims to enable its implementation and assessment in micro, small and medium enterprises.

These maturity levels are composed by processes which define what the expected results are, and capabilities which express its institutionalization level and implementation in the organization. Thus, it is noteworthy that the development

[3] Modelo de Referência para Melhoria do Software Brasileiro (MR-MPS)

[4] Associação para Promoção da Excelência do Software Brasileiro

[5] Programa para Melhoria do Processo do Software Brasileiro (MPS.Br).

among these levels happens cumulatively and only when all demands were found.

III. NON-INTRUSIVE PROCESS TO SOFTWARE ENGINEERING DECISION SUPPORT (NIPSEDS)

To characterize the proposed process we divided it into (*i*) Activities, (*ii*) Roles, (*iii*) Tools & Techniques and (*iv*) Inputs and Outputs.

A. *Activities*

We grouped the activities of process in groups identified as (*i*) Structure Decision, (*ii*) Make Decision (*iii*) Store Decision and (*iv*) Publish Decision. This division aims to facilitate understanding and enable semantics view related to the actors. These groups are based on *C* level of MR-MPS [14]. Fig. 3 shows these activities.

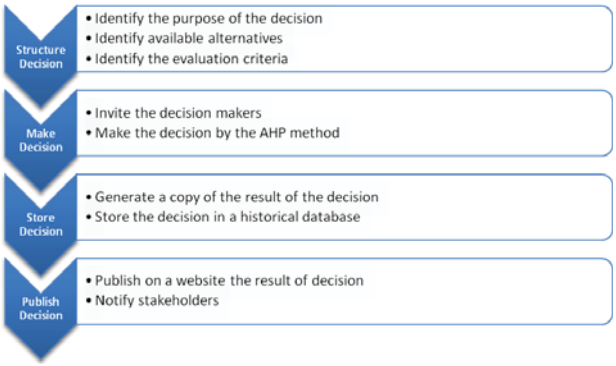


Figure 3. Process Groups of the Non-Intrusive Process to Software Engineering Decision Support (NIPSEDS)

This support process can be executed at any discipline or RUP iteration (Fig. 2). RUP is a process used by public university (Section 4) of this case study. The NIPSEDS can be applied to any process of software development. Fig. 4 holds a more detailed model.

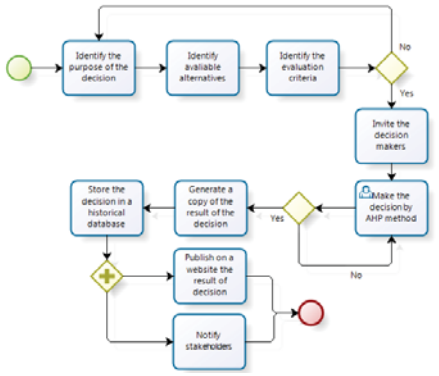


Figure 4. The Non-Intrusive Process to Software Engineering Decision Support (NIPSEDS)

B. *Roles*

The *roles* used in the process are two: (*i*) *Project Manager* (or *Scrum Master*) and (*ii*) *Decision-Makers* (which can be developers, database administrators, architects, testers, business analysts, and others). Fig. 5 shows these roles and their relationship with the activities.

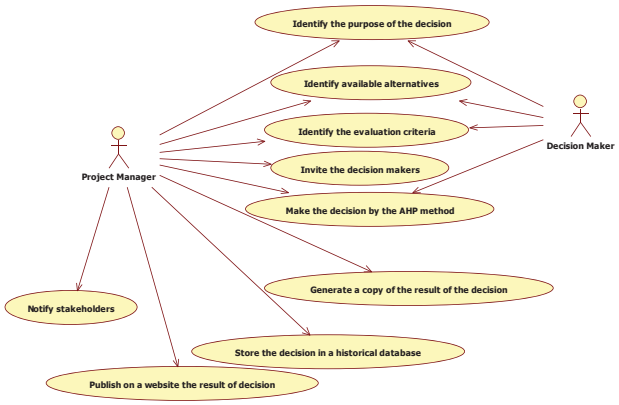


Figure 5. Roles of The Non-Intrusive Process to Software Engineering Decision Support (NIPSEDS)

Note that the *Project Manager* participates in all process activities. This is important to have an "Owner" of the process being responsible for ensuring the use of it and its constant improvement. The role of *Project Manager* was chosen to represent someone with administrative and managerial responsibilities for the project and not only with *technical responsibilities*.

C. *Tools & Techniques*

The AHP was the technique used to structure the decision. Further details and examples of how to use it can be seen in [11, 12]. For the tool we used the *Expert Choice*. It´s important to note that in this process is possible to use tools and techniques adapted to the software development process of the organization. The AHP technique comprises three activities of the group process *Structure Decision*:

- *Identify the purpose of the decision*. This activity seeks to identify the final goal of the decision, i.e., what we intend to achieve. As obvious as it may seem, this is not always trivial.
- *Identify available alternatives*. Identifying alternatives consists basically of an investigation process. The alternatives available are not always by the team known and / or have been used in the past by the organization. The important thing here is to research and rank the possible options that can be used for decision making.
- *Identify the evaluation criteria*. The criteria are the attributes that the alternatives listed must be compared. These criteria may be conflicting or mutually exclusionary. The AHP helps prioritize these criteria into a hierarchy [11, 12].

D. *Inputs and Outputs*

The process inputs are (*i*) the decision objective, (*ii*) a set of alternatives, (*iii*) a set of criteria, (*iv*) the stakeholders and a (*v*)

method to assist in structuring the decision (in the case AHP). The outputs are (*i*) the decision result and (*ii*) the decision documentation, thus creating an organizational memory [1].

These inputs and outputs are important to the creation of *Historical Database* (HDB). This artifact can be considered as an organizational asset [1], since it stored the decisions made throughout the lifecycle and to allow that future decisions are based on a set of criteria that are always feedback. Fig. 6 shows a HDB class diagram.

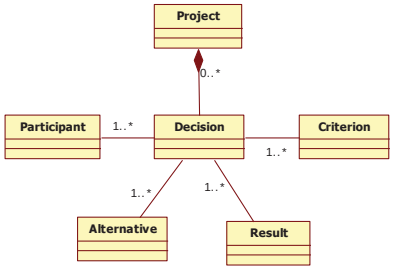


Figure 6. The Historical Database (HDB) class diagram

IV. VALIDATION

The research methodology used in this article was a case study. According to Yin [13], case studies offer an empirical research that investigates a contemporary phenomenon and offers researchers an object of applied study in its natural context. And, in addition, new facts and research issues about this environment can be identified [13].

In order to work on the case study, we selected a project of a software factory in a public university. Their teams were composed by undergraduate and master's students. Because of this, the organization suffers with the seasonality issues in periods of academic activity, lack of commitment, interest and a low rate of productivity in its members. Another problem of this organization is the lack of a process of preservation of intellectual capital generated during the projects.

During two projects with 6 iterations of 15 days each, we apply the NIPSEDS and 3 variables were collected (*i*) *Rework Index*, (*ii*) *Structured Decision* and (*iii*) *Performance Index*. To illustrate the NIPSEDS, one structured decision will be presented below separated by the process groups. This decision was performed in the second iteration of the first project and is intended to decide which persistence framework to use.

A. *Structure Decision*

The outputs of these process group activities are summarized in Fig. 7. The *alternatives* are (*i*) Entity Enterprise Java Beans[6], (*ii*) Hibernate[7], (*iii*) Java Persistence API[8] and (*iv*) TopLink[9]. These are some persistence framework to java software development. It is important to note that all *decision elements* (goal, criteria and alternatives) so collected by the team.

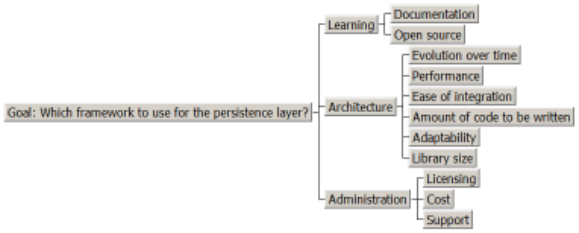


Figure 7. Criteria hierarchy ("*Struture Decision*" activities output)

B. *Make Decision*

With the established hierarchy made up some iteration where each participant reported their preference about the criteria and alternatives [11, 12]. The result of these preferences can be viewed in Fig. 8.

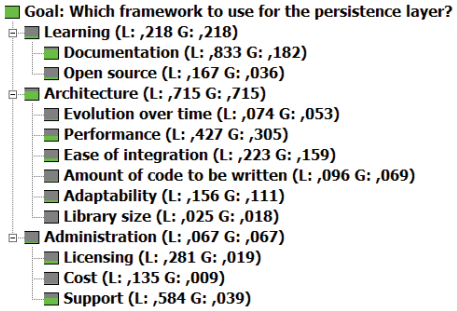


Figure 8. Hierarchy with the preferences result (more details in [11, 12])

After consensus about the choice, the outcome of the decision can now be display. Fig. 9 shows decision results.

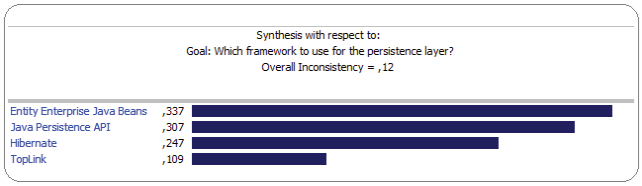


Figure 9. Decision results (represents a consensus about the choice)

The *Expert Choice* allows *different* analyzes about the decision taken. Two of them can be seen in Figures 10 and 11 respectively. Fig. 10 shows the sensitivity of alternative groups of criteria. Fig. 11 shows the result of adherence with relation to the criteria.

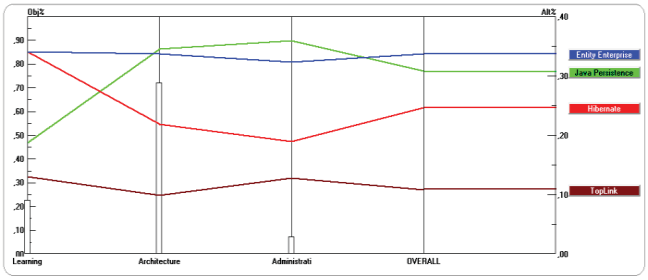


Figure 10. Alternatives sensibility of on criteria group

[6] http://docs.oracle.com/cd/E16764_01/web.1111/e13719/toc.htm
[7] http://www.hibernate.org/
[8] http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html
[9] http://www.oracle.com/technetwork/middleware/toplink/overview/index.html

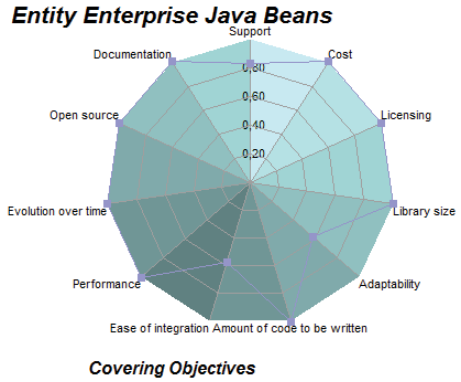


Figure 11. Adherence with relation to the criteria (note *coverage* of ≃72% regarding the the objective criteria)

C. Store Decision

After consensus about the choice, the outcome of the decision can now be stored. Table III shows an example of stored result.

TABLE III. EXEMPLE OF STORED RESULT

Attribute	*Value*
Date & Time	18/10/2012 – 10:50
Goal	*Which framework to use for the persistence layer?*
Alternatives	1. Entity Enterprise Java Beans 2. Hibernate 3. Java Persistence API 4. TopLink
Criteria	*1. Learning* 1.1. Documentation 1.2. Open source *2. Architecture* 2.1. Evolution over time 2.2. Performance 2.3. Ease of integration 2.4. Amount of code to be written 2.5. Adaptability 2.6. Library size *3. Administration* 3.1. Licensing 3.2. Cost 3.3. Support
Result	1. Entity Enterprise Java Beans
Decision Makers	Bill Mark Steve

These data were stored in the structure shown in Fig. 6 (Section 3 *D*). Artifacts such as images and Portable Document Format (PDF) can be annexed increase the quality of Historical Database (HDB). The Decision Maker's names were changed for confidentialy questions.

D. Publish Decision

The published result can be seen in Fig. 12. Effective communication creates a bridge between diverse stakeholders who may have different culture and organizational backgrounds, different levels of expertise, and different perspectives and interests, which impact or have an influence upon the project execution or outcome [1].

Project Lambda / Analysis and Design

Decision about the persistence framework

Attribute	*Value*
Date & Time	10/03/2013 – 10:50
Goal	Which framework to use for the persistence layer?
Alternatives	1. Entity Enterprise Java Beans 2. Hibernate 3. Java Persistence API 4. TopLink
Criteria	*1. Learning* 1.1. Documentation 1.2. Open source *2. Architecture* 2.1. Evolution over time 2.2. Performance 2.3. Ease of integration 2.4. Amount of code to be written 2.5. Adaptability 2.6. Library size *3. Administration* 3.1. Licensing 3.2. Cost 3.3. Support
Result	1. Entity Enterprise Java Beans
Decision Makers	Bill Mark Steve

Figure 12. Published decision in project´ website

V. RESULTS & ANALYSIS

In order to validate the process model, some performance indicators for information and data collection were defined and applied (Section 4). Through the analysis of these sources, it was possible to identify *advantages* and *limitations* of NIPSEDS. Afterwards, the results obtained with this research are described. The first indicator shows the variation in the rate of rework. This is because high levels of rework were presented as major problems during the development of a project.

Through decisions made throughout the project, we have tried to reduce the number of rework. Thus, solving the rework, this metric helps the project manager to identify the level of effectiveness of decisions. Fig. 13 shows this indicator.

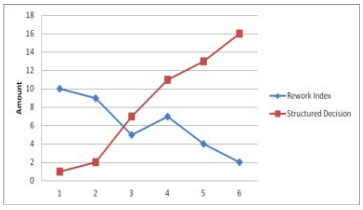


Figure 13. Rework Index vs. Structured Decision

As can be observed in Fig. 13, the structured decisions have strong relationship with the decreasing of rework. After the implementation of the framework, this fact is evidenced by the *decrease* in ≃78% (average) of this index in Projects, that research contributes to the quality in the development process. Besides this, there was an improvement on the perfomance index of members by structuring decisions during project, it is also important for improving quality in the development process. This happens because the effectiveness of the performance actually contributes to the effectiveness of the members. Thus, by sctructuring decisions, seeks to empower and qualify them so they can increase this indicator. Furthermore, through this measure, makes it possible to project manager to analyze the performance of its members and, if necessary, take steps to improve them. Fig. 14 has the graphics prepared for analysis of this index.

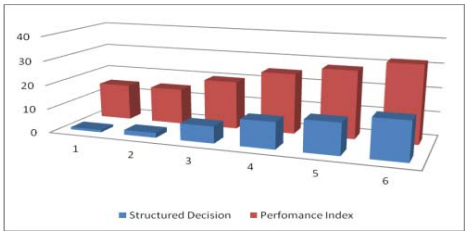


Figure 14. Performance Index vs. Structured Decision

Besides the rate of rework the structuring decision also maintains a strong relationship with the improvement in the performance index of members of the team. This fact strongly evidenced by analyzing the graphs shown in Fig. 14. Through them, it is noted that with the number of structured decision, implemented by the framework, an *increase* of ≃22% in performance of the members. And that contributes not only to meet the deadline and measurement of the team, but also to improve the quality of coding, and especially the ease of maintenance. Thus, in a general way, through the analysis of the performance indicators and collected information, the following advantages were identified:

- *Increased understanding of decisions*: With the structuring of decisions, the understanding of the problem to be solved increases. This is reflected in the later stages where decisions of the past can be retrieved and validated;
- *Improvement in the development process*: The ongoing process of analysis and monitoring ensured that the best options were selected for each objective;
- *Improvement in choice of criteria and/or alternatives*: These activities are improved by the selection and utilization of criteria, alternatives and objectives stored in the *Historical Database* (HDB);
- *Increase of the organizational memory*: The storage of experiences, estimates, knowledge and performance of team's members during the development of the projects, in the suggested HDB, has as objective to keep this information available in the beginning of every project in order to facilitate the future decisions.

Moreover, through continuous monitoring of performance and decision aspects in software projects, it can be stated that the mentioned advantages have contributed significantly for the decrease of its rework and for the increase in performance and improvement of its activities development. All these factors, besides contributing significantly for the application of the process model, also collaborate to the establishment of an asset within the organization.

VI. Conclusions and Future Works

Analyzing the results obtained during the case study development, we can evaluate the success in the implementation of the process model. It is highlighted, mostly, the increase in motivation of members and their performance, resulting in a significantly improvement in its development process and decrease rework.

Thus, focused on *increase* the *quality* of software development process, the process model presented was developed to attend, provide and add more value to process used by organization through planning and continuous structuring decisions. One possible limitation of this work is the need for a certain level of maturity in software development. *C* level according to MR-MPS [14]. In this case study the responsibility to lead the process was delegated to the most *experienced* organization member (*Project Manager* or *Scrum Master*). Finally, as presented in Table II, this process model differs from other existing process models in literature and can be applied in a *non-intrusive* way.

As future work we intend to analyze the relationship between times spent on decisions versus the time saved with rework. This has an *economic* objective related to software development.

References

[1] PMI, A Guide to the Project Management Body of Knowledge (PMBOK guide), Fifth Edition, Project Management Institute, Inc, 2013.

[2] P. Kruchten, The rational unified process: an introduction, ser. The Addison-Wesley object technology series. Addison-Wesley, 2004.

[3] Lavallée, M., & Robillard, P, "The impacts of software process improvement on developers: a systematic review". International Conference on Software, 113–122, 2012.

[4] Xuan S., "A Novel Kind of Decision of Weight of Multi-attribute Decision-Making Model Based on Bayesian Networks, Business and Information Management", 2008. ISBIM '08. International Seminar on , vol.2, no., pp.30,33, 19-19 Dec. 2008.

[5] Colwell, B., "Engineering decisions", Computer, vol.36, no.8, pp.9,11, Aug. 2003.

[6] Yang F., Zhang W., "Exploration and practice of constructing creative engineering laboratory on software development", Computer Science & Education (ICCSE), 2012 7th International Conference on , vol., no., pp.1597,1601, 14-17 July 2012.

[7] Vinay, S., Aithal, S. and Sudhakara, G., "Integrating TOPSIS and AHP into GORE Decision", International Journal of Computer Applications (0975 – 8887) Volume 56– No.17, October 2012.

[8] Felfernig A., Zehentner C., Ninaus G., Grabner H., Maalej W., Pagano D., Weninger L., and Reinfrank F., "Group decision support for requirements negotiation". In Proceedings of the 19th international conference on Advances in User Modeling (UMAP'11), Springer-Verlag, Berlin, Heidelberg, 105-116. 2011.

[9] Hassan, A., and Xie, T., "Software intelligence: the future of mining software engineering data". SDP workshop on Future of software engineering, 161–165. 2010.

[10] Lewis, T., Spillman, R., and Alsawwaf, M., "A software engineering approach to the documentation and development of an international decision support system". Journal of Computing Sciences. 2010.

[11] Gomede, E., Barros, R. M., "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. VIII Simpósio Brasileiro de Sistemas de Informação, p. 408-419. v. 1. 2012.

[12] Gomede, E., Proenca JR., M. L. and Barros, R. M., "Networks Baselines And Analytic Hierarchy Process: An Approach To Strategic Decisions." In: IADIS International Conference Applied Computing, 2012, Madri. IADIS International Conference Applied Computing, p. 34-41. 2012.

[13] Yin, R. K, Case Study Research: Design and Method, Third Edition, Applied Social Research Methods Series, Sage Publications, Inc, 2002.

[14] MR-MPS (Modelo de Referência para Melhoria de Processo do Software Brasileiro). Associação para Promoção da Excelência do Software Brasileiro, December, 2012.

[15] Saaty, T. L, The Analytic Hierarchy Process. New York: McGraw-Hill International. 1980.

Group Profiling for Understanding Educational Social Networking

[1]João Gomes, [1]Ricardo Prudêncio, [2]Luciano Meira
[1]Informatics Center – Federal University of Pernambuco - Recife, Brazil
[2]Department of Psychology – Federal University of Pernambuco - Recife, Brazil
[jeag, rbcp]@cin.ufpe.br
luciano@meira.com

[3]Alexandre Azevedo Filho, [1,4]André Nascimento, [1]Hilário Oliveira
[3]Computer Engineering Center - University of Pernambuco – Recife, Brazil
[4]Federal Institute of Education, Science and Technology of Pernambuco – Ipojuca, Brazil
agvaf@ecomp.upe.br
[acan, htao]@cin.ufpe.br

Abstract: **With the well-known success of social media, many different social networking services have been developed. One example of these new services are the educational social networks, which make use of social networking technologies for educational purposes. People sharing certain similarities or affiliates tend to form communities within social media. In educational social networks, several factors lead to agglomeration of users, e.g. studying in the same school or grade, curricular interests in common, etc. These diverse activities leave behind traces of their social life, providing clues to understand changing social structures. In order to explain the group formation resulted from educational social network, we applied a strategy of differentiation-based group profiling, using the Wilcoxon rank-sum test. The performed experiments showed that the method was effective in identifying tags to characterize the groups, pointing tags for 81.81% of groups. This research can assist network navigation, visualization and analysis, as well as monitoring and tracking the ebbs and tides of different groups in evolving networks.**

Keywords: educational social networks; communities; group profiling.

I. INTRODUTION

Educational data mining is an emerging field that involves the application of computational techniques to identify patterns in large educational data repositories [1] [2]. For example, in [1], the educational data mining is applied to assess the performance of students; in [2] it is investigated the impact of e-learning activities on the students' learning development. Sophisticated techniques of data mining are especially useful in the context of electronic educational systems, since an informal process of analysis is not feasible due to the complexity, variety and low-quality of data collected [3].

The challenges created by technologic changes are meaningful in society, modifying the ways people interact. Among these technologies, social networks had a great prominence and became a global phenomenon that is present in human social life. Social networks provide students the opportunity to interact with other students, teachers, administrators, without geographical limitations. Researchers support social networks for their capability to attract, motivate and engage students in meaningful communicative practice, content exchange, and collaboration [4]. A variety of educational social networks can actually be observed in recent years [1,2,3].

The deployment of social networks in educational systems brings new challenges concerned to the data mining process. In fact, data mining is tackling now the problem of mining richly structured, heterogeneous datasets (e.g. social networks). These kinds of datasets are best described as networks or graphs [5]. Studies on link mining have been made for social network analysis, among which stands out the specific problem of *group profiling* [6]. This problem aims to build descriptive profiled groups of people. A group (or community) is a set of users who interact with each other frequently [7]. The applications for group profiling include: understanding social structures, visualization and network navigation, monitoring the changes of a group of subjects, direct marketing and alarming cases [6].

It is evident the formation of groups in educational environments due to common interests or affinities [1]. For instance, some users may interact because they share the same school/classroom, are engaged in the same activities or are interested in the same study subject or course. Identifying the features that distinguish a group from the rest of the network is important to explain the dynamics of group formation and to support the decision making process in the education environment.

In this context, this paper proposes a methodology to perform group profiling in educational social networks, which can help to understand the process of community formation under online educational platforms. Our goal is to extract attributes of each individual user and verify whether the group members really have interests and/or common characteristics that differentiate them from the rest of the network.

In this approach, we initially applied the Multi-level Aggregation Method [8] for discovering groups in the social network data. Giving a set of attributes that describe the users, the group profiling is performed by identifying

[1] www.joystreet.com.br/products/oje
[2] www.nren.net.np/
[3] www.redu.com.br/

those attributes presenting a distribution within the group that is statistically different from the distribution observed in the rest of the network. More specifically, in our work, the Wilcoxon rank sum test [9] was deployed to identify the best discriminative attributes statistically. This test establishes a difference between two sample groups using magnitude-based ranks.

Experiments were performed using data collected from the educational social network OJE. The OJE is a web platform that works as a social network, where users are presented to challenges in the form of games and questions about several school subjects, called enigmas [10]. Based on the logs generated by the activities (games and enigmas), it is possible to collect attributes that enable the analysis of this social network. The information extracted from OJE can be useful for evaluating the learning process, as well as supporting the decision making by managers and administrators of educational systems.

The remaining of this paper is organized as follows: In Section II we present some related work, followed by Section III, where we describe the strategy of group profiling. In Section IV, we introduce the experimental settings. In Section V, we present and discuss the obtained results. Finally, in Section VI the concluding remarks and future work are outlined.

II. RELATED WORK

Group profiling describes the shared characteristics of a group of people. According to [6], the main objective of group profile is to understand the formation of explicit communities, using individual attributes. Besides, three sensible methods of group profiling are presented in a comparative study: aggregation, differentiation, and egocentric differentiation. This work uses individual attributes for group profile. In our study, we tested the effectiveness of the statistical method Wilcoxon rank sum, to generation of group profiling using a differentiation-based approach.

Other research extends topic models to extract groups based on both network and text information. Conventionally, a collection of documents are modeled as a set of latent topics, and each topic represents a distribution of words. In [11], the authors combines a topic model [12] and a mixed membership stochastic block model [13] by sharing the same latent mixture of communities for both, word topics and relation topics. In [14], connections between documents are considered in a different fashion, as they enforce the connected documents to share similar topics and use the network information as regularization to extract topics. The goal of these studies is to extract relevant themes of a collection of documents, while the study presented in this work aims to extract representative attributes that describe a particular group. The next section will describe the strategy used in this group profiling study.

III. PROFILING STRATEGY

In this section, we present our strategy for group profiling. Figure 1 presents the general process followed by our strategy for group profiling. Initially, the data set are preprocessed, for extracting features to user's representation. After that, the network structure is produced, composed by a set of nodes (representing the users) and their corresponding edges. A community detection method is applied on network for identifying the existing communities (or groups). Finally, the Wilcoxon rank sum test is applied to identify relevant features that discriminate each group.

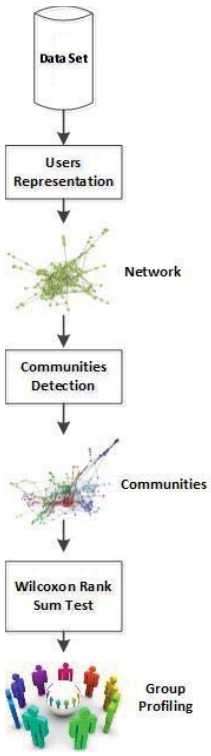


Figure 1. Group profiling strategy.

In the *Users Representation* stage, the relevant attributes for characterization of users and network formation are collected from the dataset. For instance, in an educational social network, the attributes might be: gender, age, school, game logs, enigma logs, among others. After this stage, we generate the network composed by nodes and edges. The nodes (users) bring all the information selected in the previous phase, and the edges represent the relationships between the nodes (e.g. if user A and B are friends, then an edge is created). As it will be seen, in our work, the network data is represented and manipulated by using the framework Gephi[4] (tool of interactive visualization and analysis of networked data).

Once we produced the network structure, we perform the phase of *Communities Detection*. In this stage, we adopted in our work the Multi-Level Aggregation Method algorithm [8] for identification of communities, since the groups are not explicit in the network. This is a multistep method based on a local optimization of modularity in the neighborhood of each node [15]. The optimization is performed in two steps. First, the method looks for "small" communities by optimizing modularity locally. Second, it aggregates nodes belonging to the same community and

[4] www.gephi.org/

builds a new network whose nodes are the communities. These steps are repeated iteratively until a maximum of modularity is attained and a hierarchy of communities is produced. According to [16], this method offers a fair compromise between the accuracy of estimating the modularity maximum and computational complexity, which is essentially linear in the number of links of the graph. It is a good alternative compared for instance to greedy methods for modularity optimization [17].

After the community detection is performed, we applied the *Wilcoxon Rank Sum Test* for group profiling. A natural and straightforward approach of group profiling is to find attributes that are most likely to occur within the group. But, instead of aggregating the features, we can select features that differentiate the group from the others in the network. The goal is to find out the top-k discriminative features that are representative of a group, but rarely appear in the other groups. According to [6], aggregating individual attributes is applicable only in a relatively noise-free environment. When profiles are constructed from noisy attributes, such as user blog posts, log of user activities or self-reported interests, differentiation-based methods will consistently outperform the aggregation-based approach.

In our work, we applied a differentiation method based on the Wilcoxon rank sum test [14] in our group profiling approach. This test is a nonparametric procedure to establish significant differences between two sample groups using magnitude-based ranks (in our case, a particular group against the rest of the network). Based on the test result, one can identify group profiling (attributes that would categorize the groups), that is, attributes that showed a statistically significant difference compared to the rest of the network (in our work we considered a p-value of 0.05 for statistical significance).

The Wilcoxon rank sum test uses a z-statistic to compute the approximate p-value of the test. If n_1 and n_2 are the sizes of two independent samples, where $n_1 < n_2$, we can calculate the z-statistic described in Equation (1):

$$Z = \frac{R - \mu_R}{\sigma_R} \quad (1)$$

where,

$$\mu_R = \frac{n_1 (n_1 + n_2 + 1)}{2} \quad (2)$$

$$\sigma_R = \sqrt{\frac{n_1 n_2 (n_1 + n_2 + 1)}{12}} \quad (3)$$

In equation (1), R is the sum of ranks of the elements in the smaller sample; n_1 = size of the smaller sample; n_2 = size of the larger; $n_1 \geq 10$ and $n_2 \geq 10$; μ_R and σ_R are respectively the mean and standard deviation. If both samples have the same size, either size can be used as n_1.

In the next section, we present the case study and experimental methodology adopted to evaluate the proposed approach.

IV. Case Study and Experiment Setup

In this section, we present the evaluation of our strategy to group profiling applied to an educational social network called OJE.

A. OJE

The OJE is a social network that connects students and teachers through games and enigmas. As main activities the OJE includes: (i) games with known mechanical youth to ensure their motivation, (ii) constructed enigmas in the format of the Brazilian National High School Exam's questions.

The OJE's platform provides an environment for conducting tournaments between teams and individual students (supervised by teachers) who engage in various disputes. The project also enhances the teaching processes through areas dedicated to teachers, such as a section of lessons tips to help them use the games in their disciplines, and a bank of questions (in development) that facilitates the composition of exercises from the enigmas.

B. Data Set

As mentioned earlier, to conduct a study of group profiling, it is necessary to have a rich suite of related data on individual attributes. Hence, we selected the OJE social network data in our case study.

The OJE network presents 5590 users, of which 5204 are active with 9340 relationships. Each user has an average number of 3.59 friends. In Figure 2, we provide a visualization of part of the generated network.

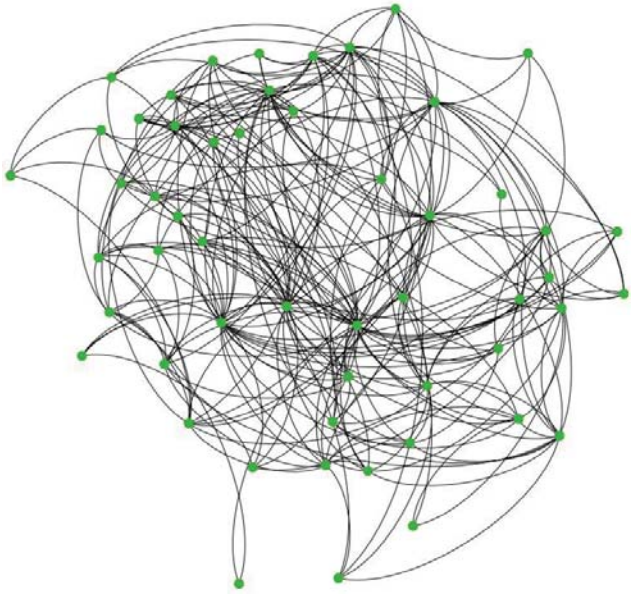


Figure 2. Fragment of the generated network.

C. Users' Representation

For users' representation, we performed several pre-processing procedures. Initially, a set of 40 individual attributes was extracted from the database. The most educationally descriptive features were selected, resulting in set of 13 attributes. Such discarded attributes were School, Grade and City, as these imply in obvious groups. The selected features are described below.

- *Age:* This attribute was used to verify the existence of groups by age ranges.
- *Access:* We applied a verification of the users activity level by establishing three attributes: Website, Games and Enigmas. These were extracted from server logs.
- *Participation in enigmas:* Aiming to analyze the participation on enigmas, three attributes were defined: the number of questions accessed, and number of correctly and incorrectly answered enigmas.
- *The Classification of games and enigmas by related educational area:* In OJE, each game and enigma has a classification that defines its educational area. There are six attributes used to group game and enigmas access number. Both games' and enigmas accesses were distinguished by *Nature*, *Literature* and *Humanities*.

D. Community Detection

The OJE has no definition in the context of groups, with no explicit communities. Thus, it demanded the application of external algorithms to identify communities groups. We started the pre-processing of the data by removing the singletons (single node) from the dataset. Since the objective is to build group profiling, they could not be in any community. Using the Multi-level Aggregation Method [8], we identified 29 groups.

The groups that had fewer than 10 users were removed, since they were considered too small and irrelevant for the study. We calculated the density for each group, as it is a common metric of how well connected a network is (in other words, how closely knit it is) [18]. Only 10 groups were selected, based on their density values. As the 10th and 11st showed the same density value, we added the last. The statistics of the preprocessed data set are summarized in Table I, in which is presented the number of users and links, the density, the network average degree, the network diameter and the number of groups.

TABLE I. STATISTICS ON NETWORK PRE-PROCESSED

	OJE
#Users	227
#Links	672
Link Density	0.026
Average Link	5.921
Diameter	8
Group Numbers	11

In Figure 3, we visualized the resulting network after the pre-processing step. Analyzing the figure we can distinguish groups formed in the network, and identify their labels. In Table II, we have all the statistics of each group individually, introducing size, density, and average degree of each group. In the table, we can identify groups that are more cohesive than others, for example, comparing the group 25 and 19.

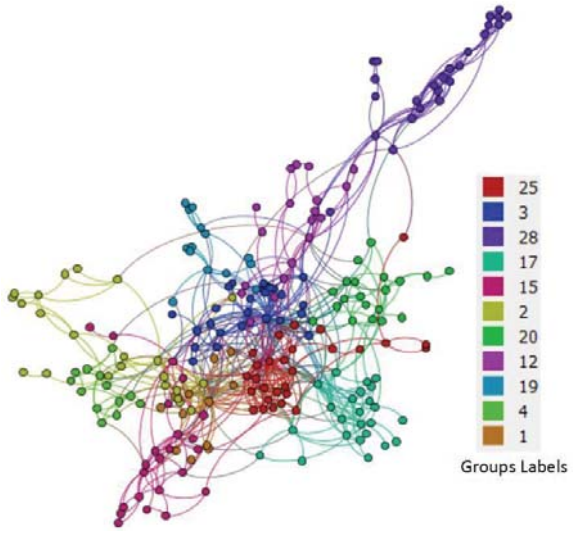


Figure 3. Network resulting from pre-processing.

TABLE II. STATISTICS ON GROUPS

OJE			
Group	**Size**	**Average Degree**	**Density**
1	12	2.5	22.7 %
2	23	3.217	14.6%
3	26	3.692	14.6%
4	13	3.538	29.5%
12	19	3.368	18.7%
15	23	4.0	18.2%
17	24	4.75	20.7%
19	14	2.286	17.6%
20	20	2.8	14.7%
25	28	5.857	21.7%
28	25	5.84	24.3%

E. Group Profiling

After the detection of communities, we applied the Wilcoxon Rank sum test as differentiation method to identify the attributes that characterize each community generated. The Wilcoxon rank sum test works by pairing the distribution of the attribute values of a particular group comparing to the values of the remaining groups.

By applying the test, we obtain the p-value for all communities' attributes. We considered as good descriptors of a community, those attributes in which a p-value lower than 0.05 was observed for the Wilcoxon test. The group profile is the list of features that characterize the community according to the statistical test.

V. RESULTS AND DISCUSSION

In this section, we present the results obtained from the application of the proposed group profiling strategy to characterize the groups of users, identified on OJE educational social network by the Multi-level Aggregation Method [8].

In Table III, we can visualize the relevant features for each group, according to the test results. As said, we selected only the features that presented statistically significant differences according to the Wilcoxon test. A feature is marked in blue when its average value in the

group is greater than the average feature value observed in the rest of the network. A red mark in turn indicates that the feature within the group has a lower average compared to the average value considering all network users.

TABLE III. RELEVANT FEATURES FOR EACH GROUP.

Groups	Labels / Tags
1, 17 and 20	No Features
2	website
3	number of answered enigmas, age
4	age, nature game, literature game, human game, games, website
15	number of answered enigmas, age
12	enigma, number of answered enigmas, nature enigma, literature enigma
19	age
25	enigma, number of answered enigmas, nature enigma, literature enigma, website, nature game, literature game, games
28	age, website

One can observe that the combination of the identified features for each group is unique, which demonstrates the potential of the proposed method to characterize the groups. In order get a better understanding of the results, here we discussed three concrete examples: the groups 12, 20 and 25.

As the experiments in the previous section showed, we found that the group 25 is the most prominent one in the network. Figure 4 (a) presents a bar plot containing the average value of each attribute describing group 25 compared to the average value for the rest of the network. At the end of the analysis, we concluded that this group consists of people who are very involved in OJE, from website usage to the participation in games and enigmas.

Regarding group 12, we can see opposite behavior, as it can be seen in Figure 4 (b). All features describing group 12 present an average value that is lower than the observed average for the rest of the network. This group of users is tied community, not only by the structure identified by the community detection method, but also by common behavior. The group has a good average (although not significant) access to the website, but very low level of access to enigmas, which is bad for the educational purposes of OJE. Further investigation could be applied on this set of users, to explain such behavior.

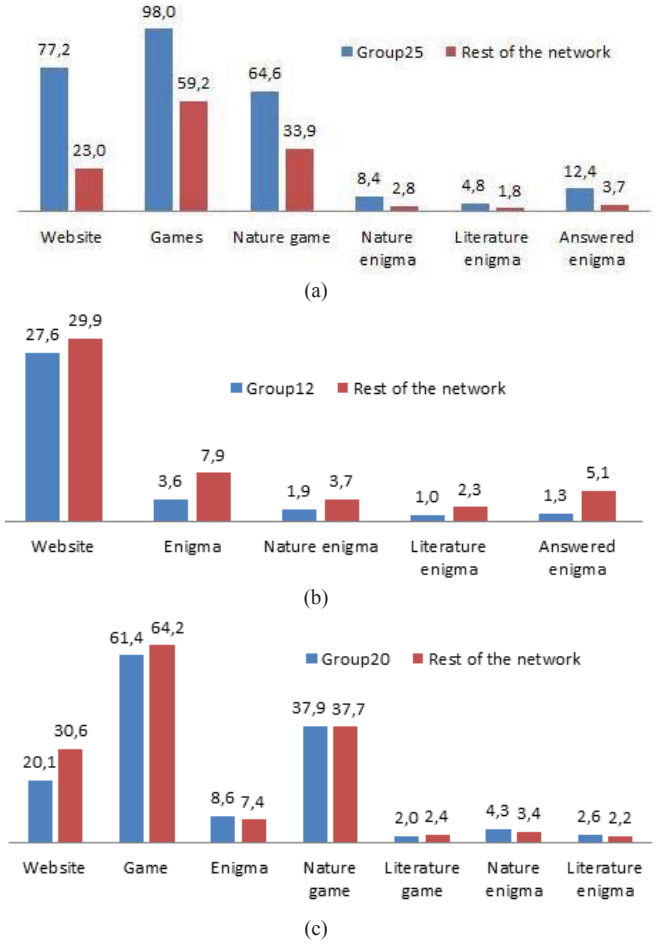


Figure 4. Averages of attribute values of one group (blue) and the rest of the network (red). (a) we see that the group 25 really stands out about the rest of the network (b) Despite of group 12 have a good OJE access average, have little access to enigmas (c) Observe that there are no significant differences attributes between Group 20 and rest of the netwwork.

The limitation of the proposed group profiling approach is in those groups where the differences between the attributes of the groups and the rest of the network are minimal (not significant). In such cases, the test did not reveal any feature to categorize the group, showing not be effective in such cases. In Figure 4 (c), we can see an example of the group 20.

VI. CONCLUSIONS AND FUTURE WORK

As we can see, there are several possibilities for the formation of communities in an educational social network. The group profiling discussed in this work is one technique that can be used to find out possible reasons that causes formation of a community or why individuals connect to or interact with each other.

In this work, we adopt a group-profiling approach to extract descriptive features for a given group. Different group profiling strategies can be adopted. A natural approach would be aggregating individual attributes and considering which attribute is shared most frequently inside a group. However, as pointed out by [6], this approach is not very feasible on noisy data and better results are obtained with methods based on differentiation. Based on this, we applied the Wilcoxon rank sum test method with the objective of identifying attributes that showed a statistically significant difference in one group compared to the rest of the network, characterizing it.

Despite not indicating any descriptive feature in case of groups that did not show any statistically significant differences, the method was effective in identifying tags to characterize the groups. In fact, descriptive features were identified for 81.81% of the groups. As seen in the analysis of the results, the labels identified by the test became good profiles for groups.

This work is an ongoing study of group profiling in educational social network. Many extensions of group profiling can be explored. In current work, we propose to understand emerging social structures based on group profiles. As a future work we expected to study solutions to our problem related to no generation of labels for some groups. This problem happened due the limitation of the application of Wilcoxon test for identifying profiling group. Besides, we intended to study the application of machine learning techniques for generation of group profiling in educational social networks.

ACKNOWLEDGMENTS

Thanks to Joy Street, the developer company of OJE, for the fundamental and unwavering support to the studies presented in this article, and CNPq for financial support.

REFERENCES

[1] B. Baradwaj and S. Pal, “Mining Educational Data to Analyze Students’ Performance”. International Journal of Advanced Computer Science and Applications, Vol 2, pp. 63-69, 2001.

[2] M. Falakmasir and J. Habibi, “Using Educational Data Mining Methods to Study the Impact of Virtual Classroom in E-Learning”, International Conference on Educational Data Mining, Pittsburgh, PA, USA, pp. 241-248, 2010.

[3] C. Romero; S. Ventura, “Educational data mining: A survey from 1995 to 2005”, In: Expert Systems with Applications: An International Journal. Tarrytown, NY, USA, pp.135-146, 2007.

[4] N. Mills, "Situated Learning through Social Networking Communities: The Development of Joint Enterprise, Mutual Engagement, and a Shared Repertoire", CALICO Journal, 28(2), pp. 345- 368. 2011.

[5] L. Getoor and C. Diehl. “Link mining: a survey”, ACM SIGKDD Explorations Newsletter, New York, NY, USA, vol. 7, pp. 3-12, 2005.

[6] L. Tang, X. Wang and H. Liu. “Group Profiling for Understanding Social Structures”, ACM Transactions on Intelligent Systems and Technology, New York, NY, USA, Vol. 3, pp. 15-40, 2011.

[7] S. Wasserman and K. Faust, “Social Network Analysis: Methods and Applications”, Cambridge University Press, 1994.

[8] V. Blondel, J. Guillaume, R. Lambiotte, and E. Lefebvre, “Fast unfolding of communities in large networks”, Journal of Statistical Mechanics: Theory and Experiment. Stat. Mech.: Theory, 2008.

[9] M. Hollander and D. Wolfe. “Nonparametric Statistical Methods”, Hoboken, NJ: John Wiley & Sons, Inc., 1999.

[10] L Meira, A. Neves and G. Ramalho. “LAN House na escola: uma olimpíada de jogos digitais e educação”. In: Anais do VIII SBGames, Rio de Janeiro, RJ, 2009.

[11] R. Nallapati, A. ahmed, E. Xing, and W. Cohen. “Joint latent topic models for text and citations”, In KDD ’08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, New York, NY, USA, pp. 542–550, 2008.

[12] D.. Blei, A. Ng and A. Jordan, “Latent dirichlet allocation”, Journal of Machine Learning Research 3, pp. 993–1022, 2003.

[13] E. Airodi, D. Lei , S. Ienberg and E. Xing. “Mixed membership stochastic blockmodels”, J. Mach. Learn. Res. 9, pp. 1981–2014, 2008.

[14] Q. Mei, D. Cai, D. Hang and C. Zhai. “Topic modeling with network regularization”. In WWW ’08: Proceeding of the 17th international conference on World Wide Web. ACM, New York, NY, USA, pp. 101–110, 2008

[15] M. Girvan and M. Newman, “Community structure in social and biological networks “, Proc. Natl. Acad. Sci. U.S.A.99, pp. 7821-7826, 2002.

[16] A. Lancichinetti and S. Fortunato, “Community detection algorithms: A comparative analysis”. Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools. Brussels, Belgium; pp. 27-28, 2009.

[17] A. Clauset, M. Newman, and C. Moore, “Hierarchical structure and the prediction of missing links in networks”, Phys. Rev. E70, pp. 98-101, 2008.

[18] L. Tang and H. Liu. “Community Detection and Mining in Social Media”, Morgan& Claypool Publisheres, 2010.

Understanding Common Perceptions from Online Social Media

Derek Doran and Swapna S. Gokhale
Dept. of Computer Science & Engineering
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

Modern society habitually uses online social media services to publicly share observations, thoughts, opinions, and beliefs at any time and from any location. These geo-tagged social media posts may provide aggregate insights into people's perceptions on a broad range of topics across a given geographical area beyond what is currently possible through services such as Yelp and Foursquare. This paper develops probabilistic language models to investigate whether collective, topic-based perceptions within a geographical area can be extracted from the content of geo-tagged Twitter posts. The capability of the methodology is illustrated using tweets from three areas of different sizes. An application of the approach to support power grid restoration following a storm is presented.

1 Introduction and Motivation

Online social media services are now deeply rooted in our modern culture and people routinely turn to these services to share their thoughts and opinions. These frequent updates can provide tremendous insights into how people perceive the world around them. A significant portion of these updates are shared via smartphones and mobile devices, and hence, have location information embedded in them. This geo-tagging offers a unique opportunity to understand how the content or *what* of the posts is influenced by the location or from *where* the posts are shared [4]. Such linking of "what" to "where" can be used to support many geographic information retrieval systems [1]. Commercial agencies can also use this association between perception and location to tailor their marketing strategies to geographic demands [10].

Presently, geo-tagged social media posts are linked to specific businesses using services such as Yelp[1] and Foursquare[2]. Through this linking, people can share their reviews and experiences and alert friends to their whereabouts. Opinions about specific businesses, however, may not offer insights into how people feel about the underlying abstract notions or topics. For example, reviews about specific fast food restaurants cannot indicate whether people in the area like to eat fast food, or that eating fast food is popular. Instead, posts that talk about fast food generally, with or without reference to specific restaurants, provide clues about area-wide perceptions on the topic of fast food.

This paper proposes a methodology that uses social media posts to identify localities where a specific topic-based perception runs strong. Partitioning a geographic area into non-overlapping sub-areas, the methodology trains probabilistic language models over posts from these sub-areas. This ensemble of models is then queried with a phrase defining a topic-based perception to identify sub-areas where that perception runs strong. Illustrations using Twitter feeds from three areas of vastly different sizes, population densities, and other characteristics show that despite the diversity, the methodology can identify sub-areas with strong perceptions for several common topics. The paper concludes with an industrial application of the methodology to support efficient power recovery following a major weather storm.

The paper is organized as follows: Section 2 presents our methodology. Section 3 describes Twitter data. Section 4 illustrates the methodology. Section 5 applies it to storm damage response. Section 6 compares related work. Conclusions and future directions are in Section 7.

2 Locating Perceptions

In this section, we motivate and present the methodology for finding perceptions.

2.1 Defining Perceptions

People's perceptions about various topics may be embodied in their spoken language and now in their social media posts. These topic-based perceptions may be influenced by the general characteristics of an area and also by specific local features. For example, although people across New York City may frequently talk and post about traffic congestion and delays, this issue is unlikely to be on their minds as they stroll through Central Park. Because it is impossible to exhaustively define all topics and their perceptions, we

[1] http://www.yelp.com
[2] http://www.foursquare.com

propose a flexible approach which models the language of the social media posts for each sub-area within a given area. These language models can then be queried with multi-word phrases which define a perception about a given topic to identify sub-areas which strongly represent that topic-based perception. For example, to identify perceptions of stressful (slow) traffic, we can use the query "hate traffic" ("traffic is slow"). Note that sub-areas with stressful traffic may or may not overlap those with slow traffic.

2.2 Specifying Language Models

A language model captures the features of the written language in a collection of documents or a training corpus. It defines a probability distribution over all n-grams, where an n-gram is an ordered sequence of n words $(w_1, ..., w_n)$. We define language models for non-overlapping sub-areas ℓ_i which comprise an area $\mathcal{L}$. The maximum likelihood estimate of an n-gram, computed over a corpus of social media posts within $\ell_i \in \mathcal{L}$, is given by [3]:

$$P_{\ell_i}(w_1, ...w_n) = \frac{c(w_1, ...w_n)}{c(w_1, ..., w_{n-1})}$$

where $c(.)$ is the number of times the sequence appears in the posts. The probability that the language within a sub-area generates a phrase $T = (w_1, ..., w_k)$ is computed as the product of the probabilities of the n-grams that comprise T:

$$P(T|\ell_i) = \prod_{j=n}^{k} P_\ell(w_{j-n+1}, ..., w_j)$$

Contextual information increases with n as higher order sequences of words are considered. However, because the frequency that larger sequences appear in social media posts is very low, prevalent language models over these posts restrict to the lowest order unigrams (1-grams), which model distinct words independent of their order [16, 5, 11]. Unigrams cannot model perceptions because these must be understood in the context of some topic or subject. For example, unigrams trained over "I love driving" and "I hate driving" will model the perceptions of "love" and "hate" but will not associate them with the topic of "driving". Bigrams, on the other hand, can model the perceptions "I love", "love driving", "I hate", and "hate driving" and associate them with a person ("I") and the act of driving. Thus, unigrams can only recognize "love" or "hate", while bigrams actually identify *what* is being loved or hated. However, because social media posts may contain numerous distinct bigrams presenting unique thoughts on various topics, the estimates of bigrams over these posts may be inaccurate. To improve this accuracy, we model the probability of bigram estimates using a linear interpolation of both the bigram and unigram estimates. This interpolation compensates for the low count of a bigram (e.g. "love driving") by incorporating the expected higher count of the unigram "driving". Thus, for a sub-area ℓ_i, the probability of observing the bigram (w_{j-1}, w_j) is given as:

$$P_{\ell_i}(w_{j-1}, w_j) = \lambda_1 c(w_{j-1}, w_j)/c(w_{j-1}) + \lambda_2 c(w_j)/|W(\ell_i)|$$

where $\lambda_1 + \lambda_2 = 1$, $|W(\ell_i)|$ is the number of distinct words in all posts in ℓ_i and $c(w_j)/|W(\ell_i)|$ is the estimate of the unigram that completes the bigram [3].

We use *smoothing* to compensate for the probability of future unseen bigrams, which allocates some of the probability of the training bigrams to those that are as yet unobserved [6]. The Modified Kneser-Ney (MKN) smoothing algorithm [9] is chosen because of its superior performance with interpolated language models [6]. 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 using a modified estimate of the unigram w_j, where only the number of *distinct bigrams* that w_j completes is considered:

$$P_c(w_j) = \frac{|\{w : c(w, w_j) > 0\}|}{\sum_v |\{w : c(w, v) > 0\}|}$$

and then weighing this proportion by the probability mass $\lambda(w_{j-1})$ taken from the known bigrams:

$$\lambda(w_{j-1}) = \frac{\hat{d}|\{w : c(w_{j-1}, w) > 0\}|}{c(w_{j-1})}$$

Thus, under MKN smoothing the probability of observing a bigram becomes:

$$P_\ell(w_{j-1}, w_j) = \frac{\max(c(w_{j-1}, w_j) - \hat{d}, 0)}{c(w_{j-1})} + \lambda(w_{j-1}) P_c(w_j)$$

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 ≥ 3 [14]. Using these approximations, we set $\hat{d}$ equal to d_1, d_2, or d_3 respectively:

$$d_1 = 1 - \frac{2n_2 n_1}{n_1(n_1 + 2n_2)}$$
$$d_2 = 2 - \frac{3n_3 n_1}{n_2(n_1 + 2n_2)}$$
$$d_3 = 3 - \frac{4n_4 n_1}{n_3(n_1 + 2n_2)}$$

(a) NYC: Local-level (b) DC: District-level (c) CT: Region-level

Figure 1: Tweet Distribution in the Areas

with n_i is the number of bigrams with frequency i.

The ensemble of language models, one for each sub-area, is then queried to compute the probability that a phrase T is generated from a sub-area ℓ_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-area ℓ_i and is given by $N(\ell_i)/N(\mathcal{L})$. $N(\ell_i)$ is the number of posts in ℓ_i and $N(\mathcal{L})$ is the total number of posts in the entire area $\mathcal{L}$. Finally, we we define define $P(T|\ell_i)$ as:

$$P(T|\ell_i) = \prod_{j=2}^{k} P_{\ell_i}(w_{j-1}, w_j)$$

3 Data Description

We harvested geo-tagged tweets from Twitter between January 29^{th} and February 28^{th} 2013 from three areas, namely, Downtown Manhattan in New York City (NYC), the greater Washington D.C. area and its surroundings (DC), and the entire state of Connecticut (CT). Although these areas can be partitioned according to town and city jurisdictions or even by zip code, for the sake of illustration, we divide them into 100 equal sub-areas along a 10×10 grid using latitudinal and longitudinal coordinates. The partitions of each area differ widely: (i) NYC sub-areas include a few blocks and provide a *local-level* perspective; (ii) DC sub-areas include substantial portions of cities, suburbs, and interstates providing a *district-level* perspective; and (iii) CT sub-areas contain multiple towns, entire cities and woods offering a *region-level* perspective.

For each area, we eliminated non-English tweets and those without geo-tags. Table 1 shows that the tweet density in NYC is an order higher than DC and two orders higher than CT. The lower densities in DC and CT, however, do not impede training of the language models, because in each area tweet distributions conform to population spread as shown in Figure 1 [7]. Thus, tweets in NYC are almost uniform, in DC they cluster around major cities and follow paths to major highways, and in CT they concentrate around the three major interstates, with sparse densities in the woods and farmland towns.

Area	Sub-area	Area Size	Tweets	Density
NYC	Local	82.3km^2	110,924	1,347/km^2
DC	District	3,452km^2	394,072	114/km^2
CT	County	22,140km^2	355,678	16/km^2

Table 1: Area-wise Summary of Tweets

Geo-tagged tweets were further pre-processed by converting all words to lowercase and by stripping punctuation, hashtags (terms starting with #), username replies (terms beginning with @), and Web links. Common words such as "at", "the", and "or" lack contextual information, and hence, were eliminated using a stopword list of 200 most frequently used words. The stopword list was limited to 200 which is approximately equal to 1% of the average number of distinct words across each area. We also include a "catch all" unigram "<misc>" to aggregate the probability of all words that occur only once. It also accounts for miscellaneous, shorthand, mis-spelled, and other user-specific notations. On an average 3.16% of the words in each area were mapped to "<misc>", suggesting that we can control this source of distortion without impacting the models' fidelity.

4 Illustrations

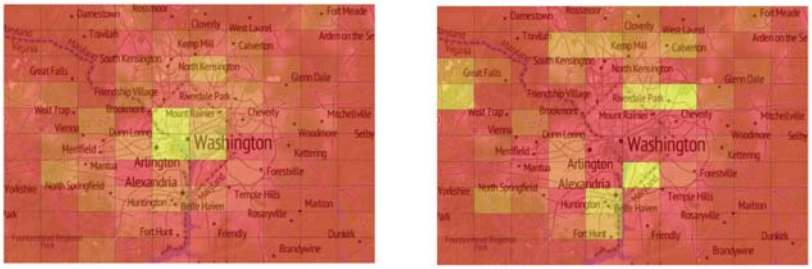
In this section, we illustrate how our approach can identify sub-areas with strong topic-based perceptions.

4.1 Perceptions in NYC

Downtown Manhattan is a popular tourist destination and includes Chinatown and Little Italy as well as Broadway and

(a) "Restaurant" (b) "Italian restaurant" (c) "Went to a great Italian restaurant"

Figure 2: Local-level Perceptions in NYC



(a) "traffic" (b) "traffic during commute"

Figure 3: District-level Perceptions in DC

Penn Station. Given its multi-cultural neighborhoods and popularity, this area is rife with many types of eateries, due to which we extract perceptions about restaurants. Figure 2a displays the results in the form of a heat map, produced by a generic query "restaurants". Brighter shades across many sub-areas indicate that people discuss restaurants broadly. Figure 2b shows the results of a refined query "Italian restaurants". The heat map now concentrates on fewer sub-areas, mostly in the southwest, which corresponds to Little Italy. It also includes northern sub-areas; home to many high-end Italian restaurants[3]. Finally, a specific query "went to a great Italian restaurant" produces Figure 2c which tells us that this perception is most strongly present in Little Italy, and at the entrances to the Holland Tunnel, Brooklyn and Williamsburg Bridges. That this perception is strong in sub-areas used to leave the city suggests that visitors may be more inclined to share their satisfaction about a great meal in Little Italy compared to the city's residents.

4.2 Perceptions in DC

This area encompasses Washington D.C. and the surrounding suburbs. Here, sub-areas include entire communities, parts of Washington D.C., portions of the I-95/495 interstate loop infamous for its heavy traffic, regional parks, and major roads that connect Washington D.C. to Maryland and Virginia. The city is dominated by office parks, federal agencies, and corporate headquarters bringing in a large number of commuters from outside towns and suburbs. We thus extract perceptions on "traffic" for this area. The heat map in Figure 3a, resulting from a generic query "traffic", shows that traffic is most strongly perceived inside and around the four sub-areas of downtown and decreases in prominence as we go farther away. The heat map in Figure 3b, resulting from a more nuanced query "traffic during commute", finds that people do not discuss traffic and commute within the city, but as expected in the sub-areas which contain portions of the I-95/495 interstate loop and those to the west neighboring Dulles Airport.

4.3 Perceptions in CT

This area covers the state of Connecticut featuring large sub-areas that include entire towns and cities. A hot-button issue that many people consider when deciding to relocate to a neighborhood is the public perception of crime. City and town governments must thus be aware of how crime

[3] http://www.zagat.com

(a) “crime” (b) “hospital”

Figure 4: Region-level Perceptions in CT

is perceived in their jurisdictions. The heat map in Figure 4a, resulting from a generic query “crime”, suggests that the people of CT do not think about crime except for sub-areas along the I-91 interstate containing the cities of New Haven, Bridgeport, Stamford, and Hartford, which are notorious for its dangerousness[4]. Because a state encompasses a large area, we also extract perceptions about topics that are less likely to be thought of at a local- and district-level. The heap map in Figure 4b, resulting from the query “hospital”, shows that the hottest sub-areas coincide with the Yale-New Haven Hospital and UConn Health Center. Also, adjacent sub-areas are more likely to think of hospitals, compared to other sub-areas in the state.

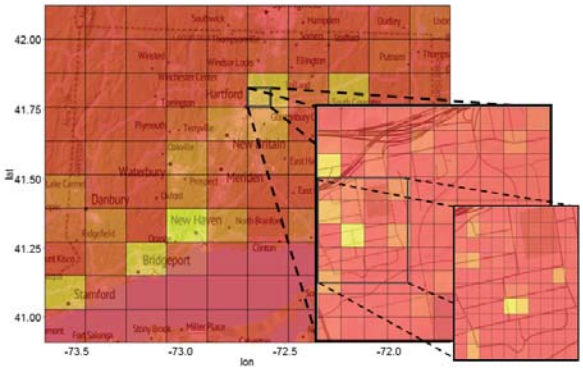


Figure 5: Perception of “power outage” in CT

5 Storm Power Grid Damage Response

The above examples illustrate the capability of our approach to identify varied topic-based perceptions. We now discuss how such identification can be leveraged for a disaster response scenario. Natural and other disasters can create potentially life-threatening conditions because of their destructive impact on an electric power grid. Responding to such outages efficiently and quickly can minimize this damage and reduce the costs of restoration and loss of productivity. Currently, utilities rely on experts to estimate locations of outages and the type and extent of damages in order to dispatch appropriate crews and materials. We describe how public perceptions on the damages caused to the power grid following a storm provides situational or “on-the-ground” data supporting the expert’s analysis. On January 31^{st} 2013, the CT area experienced hurricane force winds causing widespread power outages. The heat map in Figure 5, resulting from the query “power outage”, shows that locations of this perception correspond to the power outage map [5]. Zooming into an area northeast of Hartford, generates another heat map for the same perception, which now highlights a residential block. Zooming in even further identifies specific places on the streets, which may correspond to houses or electric poles. Based on this data, experts can tag these streets as prone to power loss. They can analyze the power grid in the area to determine what components failed, and using tweets within the highlighted sub-areas, hypothesize about the source of damage (e.g. downed trees crushing overhead lines). The appropriate crews and components can then be dispatched to quickly repair these failures.

6 Related Research

Language models have been built over social media posts for a variety of purposes. Iskandar *et al.* develop a query likelihood model with Dirichlet smoothing to retrieve content from social media and Wikipedia articles [2]. Li predicts the point-of-interest of a tweet with a unigram lan-

[4] http://www.fbi.gov/about-us/cjis/ucr/crime-in-the-u.s/2012

[5] http://www.ctpost.com/local/article/Storm-leaves-thousands-without-power-4238526.php

guage model [11]. The models are also combined with location information to evaluate the origin of tweets. Kinsella *et al.* estimate cities from which tweets originate by comparing KL-divergences among language models [8]. Chandra *et al.* also predict originating cities based on unigram models over tweet-reply chains [5]. Chang predicts positions based on the spatial usage of words in the tweets [15], Sadilek *et al.* incorporate the position of friends and content of tweets for prediction [13], Liu *et al* consider check-in histories with tweet content [12] and Wing *et al* use unigram models of tweet content across areas within geo-grids [16].

This work differs from contemporary efforts because rather than focusing on a specific set of tasks, we develop language models to generally identify perceptions of users across geographic areas. Also, our sophisticated language model uses smoothing to combine accurate estimation of unigrams with contextual information in bigrams compared to the prevalent models that consider only unigrams.

7 Conclusions and Future Work

This paper presented a methodology to identify where perceptions about a topic are strongly represented across a given geographic area. Central to the methodology are language models that can be queried using phrases that define any kind of perception for any topic. Without any *a priori* information and aid of external data sources, we demonstrate how the approach can identify where a specific topic-based perception is strongly represented in sub-areas with sizes ranging from just a few urban blocks to entire cities.

In the future, we plan to enhance the methodology with geographic and temporal variations in word usage. We will also explore the use of the methodology for many different applications including location prediction, storm and disaster management, and analytics for city planning and public services including mass transit.

References

[1] G. Andogah. *Geographically Constrained Information Retrieval*. PhD thesis, University of Groningen, 2010.

[2] D. Awang Iskandar, J. Pehcevski, J. Thom, and S. Tahaghoghi. Social media retrieval using image features and structured text. In *Proc. of Workshop on Initiative for the Evaluation of XML Retrieval*, pages 358–372, 2007.

[3] 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*, pages 179–190, 1983.

[4] R. B. Brandom. *Between Saying and Doing: Towards an Analytic Pragmatism: Towards an Analytic Pragmatism*. OUP Oxford, 2008.

[5] 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.

[6] 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.

[7] P. Division. Land Area, Population, and Density for Plances and (in selected states) County Subdivisions: 2000. *United States Census Bureau*, 2000.

[8] S. Kinsella, V. Murdock, and N. O'Hare. I'm eating a sandwich in Glasgow: modeling locations with tweets. In *Proceedings of Intl. Workshop on Search and Mining User-Generated Content*, pages 61–68. ACM, 2011.

[9] 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.

[10] J. A. Lesser and M. A. Hughes. The generalizability of psychographic market segments across geographic locations. *Journal of Marketing*, pages 18–27, 1986.

[11] W. Li, P. Serdyukov, A. P. de Vries, C. Eickhoff, and M. Larson. The Where in the Tweet. In *Proc. of Intl. Conference on Knowledge Management*, 2011.

[12] H. Liu, B. Luo, and D. Lee. Location Type Classification Using Tweet Content. In *Proc. of Intl Conference on Machine Learning and Applications*, 2012.

[13] A. Sadilek, H. Kautz, and J. Bigham. Finding Your Friends and Following Them to Where You Are. In *Proc. of Intl. Coference on Web Search and Data Mining*, 2012.

[14] M. Sundermeyer, R. Schlüter, and H. Ney. On the estimation of discount parameters for language model smoothing. *Interspeech*, 2011.

[15] H. wen Change, D. Lee, M. Eltaher, and J. Lee. @Phillies Tweeting from Philly? Predicting Twitter User Locations with Spatial Word Usage. In *Proc. of Intl. Conference on Advances in Social Networks Analysis and Mining*, pages 111–118. IEEE, 2012.

[16] B. Wing and J. Baldridge. Simple supervised document geolocation with geodesic grids. In *Proc. of the Annual Meeting of the Association for Computational Linguistics*, volume 1, pages 955–964, 2011.

Analyzing Social Behavior of Software Developers Across Different Communication Channels

Aftab Iqbal, Marcel Karnstedt and Michael Hausenblas
Digital Enterprise Research Institute (DERI)
National University of Ireland, Galway (NUIG)
firstname.lastname@deri.org

Abstract

Software developers use different project repositories (i.e., mailing list, bug tracking repositories, discussion forums etc.) to interact with each other or to solve software related problems. The growing interest in the usage of social media channels (i.e., Twitter, Facebook, LinkedIn) have also attracted the open source software community and software developers to adopt an identity in order to disseminate project-related information to a wider audience. Much research has been carried out to analyze the social behavior of software developers in different project repositories but so far no one has tried to study the social communication patterns of developers in other social media channels. We in this paper presents a new dimension to the social aspects of software developers and study if the social communication patterns of software developers is different on project repositories and social media channels (i.e., Twitter).

1 Introduction & Motivation

In software engineering, many tools with underlying repositories have been introduced to support the collaboration and coordination in distributed software development. Research has shown that these project repositories contain rich amount of information about software projects. By mining the information contained in these project repositories, practitioners can depend less on their experience and more on the historical data [14]. Examples of project repositories are [15]: source control repositories, bug tracking repositories, mailing list archives etc. Software developers[1] use these repositories to interact with each other or to solve software-related problems. Much research has been carried out to analyze the social network structure and behavior of software developers by extracting rich information from these project repositories [22, 13, 24].

[1] In this paper, we use the term "software developers" or "developers" to represent those who have commit rights on the source control repository of a project.

The growing interest in the usage of online social media channels (e.g., Facebook, Twitter, LinkedIn etc.) have also attracted the open source software community. Open source projects are often found to adopt an identity on these social media channels (e.g., Apache Solr/Lucene[2] on Twitter, MySQL[3] on Facebook) in order to disseminate project-related information (release announcements, major bug fixes etc.) or gather feedback/questions posted by the users. Software developers contributing to open source projects also exists on social media channels. Quite often, they discuss, debate or share experiences with others relevant to a software project using hashtags (e.g., `#apache`, `#maven`, `#hadoop` etc.). Hence, the discussions covering open source projects are not limited to dedicated forums or mailing lists, there also exists huge amount of information on the social media channels. However, on the social media channels, less technical details relevant to the project's architecture, code or bugs are discussed. Much of the information available is regarding the experiences[4] or announcements[5] particular to a software project but such valuable information can not be ignored.

It is worth mentioning that the information related to open source projects are distributed on the Web in heterogeneous data islands i.e., social media channels and project repositories. Hence, there is a need to bridge the connection between project repositories and social media channels as shown in Figure 1. By enabling this connection, we will have an integrated view on the software project which can be exploited to support certain use case scenarios:

- End-users response on a particular release of a software project.
- The popularity of a software project by applying sentiment analysis [17] on social messages (i.e., tweets,

[2] `https://twitter.com/SolrLucene`
[3] `https://www.facebook.com/mysql`
[4] `https://twitter.com/olamy/status/231031288734285824`
[5] `https://twitter.com/olamy/status/305334578103582720`

posts etc.).

- Keeping track of software developer's social activity related to a software project.
- Analysis of the social behavior of software developers in different communication channels (i.e., social media channels, project repositories).

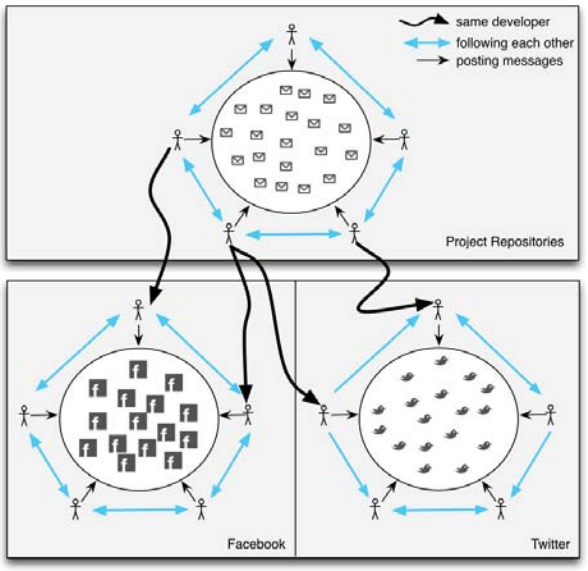


Figure 1: Linking Social Media Channels and Project Repositories.

The social behavior of users have been studied in depth in the past on different communication channels[6] separately. However, to the best of our knowledge, no research work has been done so far on the comparison and analysis of the social behavior of software developers in different communication channels. There is no research work available which analyzes the behavior of software developers communication with each other on the mailing list/bug repositories and their communication on social media channels (e.g., Twitter). This motivates us to study the social communication patterns of software developers in different communication channels.

Among different social media channels available to date, we chose Twitter as a social media channel for this study. Our initial investigation reveals that software developers contributing to open source projects also use and communicate with each other on the social media channels (Twitter in particular). For example, Figure 2 shows the developers social network structure (derived from the communication happened on the mailing list) of an *Apache* project. Among them, few developers are also found on Twitter. We derived the social network structure based on their tweets (e.g., mentioning other developers in tweets) which is shown in Figure 3. We removed the labels from nodes (cf. Figure 2 and Figure 3) in order to keep the privacy of developers.

Figure 2: Social Relation on Mailing List

Figure 3: Social Relation on Twitter

The social network structures of software developers (cf. Figure 2 and Figure 3) contributing to the same software project provides us the basis to investigate the social behavior of software developers in different communication channels. We will investigate if software developers use Twitter as another medium of communication in contrast to the traditional medium of communication (mailing list, bug tracking repositories, forums etc.). This will laid down the foundations to study the social behavior of software developers with each other in different communication channels. In the current scope of this paper, we will not take into account what software developers are discussing on Twitter but instead we will focus on the communication happened between developers on Twitter in a given period of time and compare it with their communication happened on project repositories for the same period of time.

The contribution of this work is manifolds: we have identified social media channels as a platform which is used by the open source community and software developers to disseminate project-related information to a wider audience. Further we highlighted the need to integrate project repositories and the social media channels (interlinking project-related tweets/posts/hashtags, developer ID(s), project ID(s) etc.) in order to get an integrated view on the software project. We have introduced a new dimension to analyze the social aspects of software developers by taking into account non-traditional communication channels (i.e., Twitter, stackoverflow, LinkedIn, Facebook etc.) which are also used by software developers. We have conducted an initial experiment to investigate the correlation between software developers communication with each other on Twitter and in project repositories by analyzing their communication data over time.

2 Methodology

In this section, we describe our methodology to extract information from different data sources and the usage of a common model and standard format to represent extracted

[6]We use the term "communication channels" to refer project related communication channels (e.g., mailing list, bug tracking repository, discussion forums etc.) and online social media channels (e.g., Facebook, Twitter etc.)

information in order to support better query and integration. Further, we describe our approach to compute communication network data which later is used to understand how software developers communicated with each other in different communication channels over the period of time.

2.1 Transforming Data Sources into RDF

With "Linked Data Driven Software Development" (LD2SD) [20], we have introduced a Linked Data-based methodology to relate and integrate data across software repositories explicitly and unambiguously. We propose to use Semantic Web technologies to represent data from different software repositories. As such, we propose to use RDF [21] (Resource Description Framework) as the core, target data model. Once modeled in RDF, the data can be easily integrated, indexed and queried using the SPARQL query[7] standard and associated tools. Finally, the integrated data can be published on the Web using Linked Data principles[8] allowing third parties to discover and subsequently crawl the knowledge, and also allowing to interlink with background or other related information available remotely on the Web. We refer the readers to [16] for details on how these standards would be used. Instead here we focus on transforming data from project repositories and Twitter to RDF. We used our custom written script to convert mailing lists and bug tracking repositories data to RDF. An excerpt of an exemplary RDF representation of an email is shown in Listing 1[9]. Due to space limitations, we do not show the RDF representation of a bug report but refer the readers to [19] for further details on the RDFication process.

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix sioct: <http://rdfs.org/sioc/types#> .
@prefix email: <http://simile.mit.edu/2005/06/ontologies/email#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix sioc: <http://rdfs.org/sioc/ns#> .
@prefix : <http://srvgal85.deri.ie/linkedfloss/mail2rdf/> .

:A9DB451E-4F0F-435F-8E13-9F4D86996BA4 a sioct:MailMessage ;
email:from <http://srvgal85.deri.ie/linkedfloss/aheritier> ;
dc:subject "Re: guice & memory usage was:" ;
email:body "There's been little to no feedback on beta-2 so ..." .
sioc:reply_of :4C5D409A.9060901 ;
dc:Date "2010-08-18T22:43:37+02:00";

<http://srvgal85.deri.ie/linkedfloss/aheritier> a foaf:Person ;
foaf:name "Arnaud Heritier" ;
foaf:mbox <mailto:aheritier@example.org> .
...
```

Listing 1: An Exemplary Email RDFication.

In order to compute the social communication of a software developer with other fellow software developers on Twitter, we first manually checked if software developers exists on Twitter and using the Twitter account frequently. We found few software developers who does exist on the Twitter platform but tweeted very little (≈10-20 tweets only). We skipped such software developers in the data crawling and transformation process due to less data available for them. Twitter offers an Application Programming Interface (API)[10] which makes it easy to crawl and collect data from Twitter. We crawled developers Twitter profiles and their tweets using Twitter API and later transformed it to RDF using our custom written scripts. An excerpt of an exemplary RDF representation of a developer's Twitter profile is shown in Listing 2.

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix ls: <http://lab.linkeddata.deri.ie/linkedfloss/ns/#> .
@prefix dcterms: <http://purl.org/dc/terms/> .

<http://srvgal85.deri.ie/linkedfloss/twitter/brettporter> a foaf:Person ;
foaf:accountName "brettporter" ;
foaf:name "Brett Porter";
foaf:homepage <http://twitter.com/brettporter> ;
ls:followers "994";
ls:following "707";
ls:status_count "6050";
dcterms:created "2007-03-26T00:05:50";
dcterms:description "CTO of MaestroDev, Director of ASF, and long time
      Maven, Archiva, open source guy. Author and sadly infrequent coder.
      Christian. Husband. Father. Australian." ;
ls:location "Sydney, Australia" ;
dcterms:language "en";
foaf:knows <http://srvgal85.deri.ie/linkedfloss/twitter/aheritier>;
...
```

Listing 2: An Exemplary Twitter Profile in RDF.

Developers often tweets about project-related information using hashtags (e.g., `#maven`, `#lucene`) or communicate with other fellow developers by explicitly mentioning his/her name in a tweet. An excerpt of an exemplary RDF representation of a tweet is shown in Listing 3. After transforming the data sources to RDF, we loaded the RDF data sets into our public SPARQL endpoint[11].

```
@prefix sioctypes: <http://rdfs.org/sioc/types#> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix sioc: <http://rdfs.org/sioc/ns#> .

<http://twitter.com/brettporter/statuses/20971803268>
     a       sioctypes:MicroblogPost ;
dcterms:created "2010-08-12T13:40:49" ;
dcterms:creator <http://srvgal85.deri.ie/linkedfloss/twitter/brettporter> ;
sioc:content "@aheritier the template hasn't been updated for the
      Confluence 3 upgrade, since we don't typically use the static
      rendering" ;
sioc:id "20995024391832780 9" ;
sioc:mentions <http://srvgal85.deri.ie/linkedfloss/twitter/aheritier>
.
```

Listing 3: An Exemplary Tweet RDFication.

In this paper, we are not focusing on interlinking developer's information and project related tweets from Twitter to the various software artifacts (i.e., bug, email, commit ID, source-code, developer ID etc.) contained in project repositories. Hence, we do not present any approach on creating `owl:sameAs` links between relevant entities across different data sources. Different approaches [23, 11, 18, 19] could potentially be utilized in order to achieve the interlinking between Twitter data sets and software artifacts but it is not in the current scope of this paper.

2.2 Social Relation Computation Approach

The social network structure based on the mailing list archives was constructed by using the reply structure of the

[7] `http://www.w3.org/TR/rdf-sparql-query/`

[8] `http://www.w3.org/DesignIssues/LinkedData.html`

[9] The URIs used in the listings are just for illustration purposes and are not dereferenceable.

[10] `https://dev.twitter.com/docs`

[11] `http://linkedfloss.srvgal85.deri.ie/sparql`

email threads for direct communication among developers. This approach defines a link as the interaction between the poster of actual email and the replier to the poster email. For example, Listing 1 indicates that the email is a reply of another email (cf. line#12). Hence, we can easily query the poster of email `4C5D409A.9060901` (cf. line#12) in order to create a social link between both software developers. In the case of bug tracking repository, link was defined based on the comment posted by a software developer on a particular bug and the immediate previous commenter on the same bug. The social network structure based on the Twitter data was constructed by exploiting the common practice of using well defined markup in a tweet: @ followed by a user identifier address the user. This approach defines a link as the interaction between the software developer who posted the tweet and the software developer mentioned in the tweet. For example, Listing 3 indicates that the tweet mentions a software developer which is also reflected through `sioc:mentions` property (cf. line#11). Hence, we can create a social link between both software developers (`aheritier` and `brettporter`). Social relations between developers on the Twitter was extracted by querying the Twitter data sets using `sioc:mentions` predicate property. Social relation between any 2 software developers were computed only if both software developers communicated directly to each other on the project repositories and the Twitter platform. Periods without any communication are common as software developers may still contribute to the same project even if they haven't communicated for many days. To tackle this issue, communication between software developers were captured on monthly basis where each month value represent the number of times both software developers communicated directly to each other. Collecting communication data on monthly basis provides good amount of data for every pair of software developers in order to analyze their social communication patterns over the period of time. The initial time-stamp for calculating the social relation between any 2 software developers were computed by comparing the earliest dates where communication happened between both software developers on the project repositories and Twitter. The later date was then used as the starting time-stamp to compute the social relation between both software developers. For example, let say earliest communication happened between 2 developers on the project repositories was `2008-05-25` and on the Twitter was `2010-03-15`. Thus, we will consider `2010-03-15` as the starting time-stamp and compute monthly social interaction between both software developers on the project repository and Twitter over the period of time.

3 Evaluation

Before we discuss the results of our evaluation, we describe the projects selected for evaluation. We gathered data from project repositories of 10 *Apache* projects (c.f Table 1). The reason of choosing *Apache* projects is that the repositories are on the Web and available to download (i.e. mailing list archives, bugs, commit logs etc.). We selected data from the beginning of each *Apache* project to date. The primary source of communication among developers in *Apache* projects is through mailing lists. Most *Apache* projects have at-least 3 different mailing lists: `user, dev` and `commits` but some projects have more than 3 mailing lists (e.g., `announcements, notifications` etc.). For our study, we downloaded only the `dev` mailing list archives of each *Apache* project under consideration. The reason is, software developers communicate often with each other on the `dev` mailing list rather than on any other mailing list. From the source-control repository data sets of each *Apache* project, we extracted a list of software developers who made commits to the project. Later, we manually checked if these software developers also exist on Twitter and using the Twitter account frequently. Table 1 shows for each *Apache* project the number of software developers who have made commits to the source-control repository and the developers found on Twitter.

Apache Projects	Developers (SVN)	Developers (Twitter)
Apache Camel [1]	36	24
Apache Directory [2]	51	11
Apache Felix [3]	47	17
Apache Hadoop [4]	97	35
Apache Logging [5]	37	7
Apache Lucene [6]	51	18
Apache Maven [7]	40	10
Apache Mina [8]	28	9
Apache MyFaces [9]	82	16
Apache OfBiz [10]	25	11

Table 1: Developers Contributed to Apache Projects and found on Twitter.

The results in Table 1 shows good evidence of the existence of software developers on Twitter. Although, not all software developers contributing to the *Apache* projects found on Twitter. Based on the methodology described in previous section, we found 107 distinct pair of software developers who communicated with each other on the project repositories and Twitter. In the specific case of *Apache OfBiz* project, we didn't find even a single pair of software developers who communicated with each other on project repositories and Twitter. For each pair of software developer, we computed how many times both software developers communicated directly with each other on the project repositories and Twitter, on monthly basis. For an example, we show the communication happened between a pair of software developers over the period of time in Figure 4. The figure shows that communication pattern among both software developers is different throughout the time period under consideration. For example, in `2009-08-12` both developers communicated directly with each other 7 times on the project repositories in contrast to 6 times commu-

nication on Twitter for the same month. Further, we see few months where both developers didn't communicate at all (e.g., `2011-09-12`) and in certain months they appear to communicate on only one communication channel (e.g., `2011-03-12`).

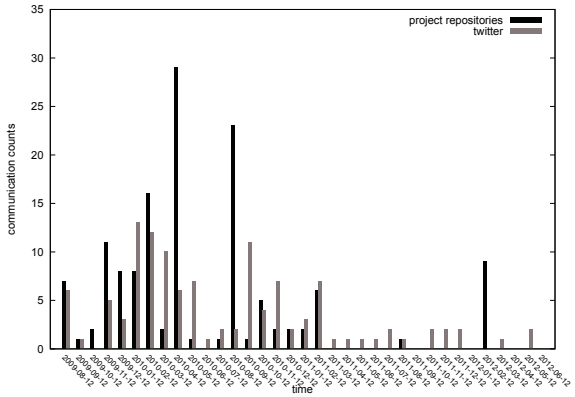


Figure 4: Social Communication Between 2 Software Developers on Different Communication Channels.

Given the social dynamics of software developers in different communication channels (cf. Figure 4), we evaluated if there is a correlation between both developers communication pattern on project repositories and Twitter by measuring Pearson's correlation. The Pearson correlation test based on the communication data between both developers yielded a correlation value, `r=0.447`. The `r` value indicates that the social communication between both developers on different communication channels is significant and show a positive correlation of developers communicating on different communication channels.

In order to find if the communication among software developers is directly proportional on different communication channels, we aggregated the monthly communications occurred on the project repositories and Twitter for every pair of software developers. We plotted the resulting graph in Figure 5. The graph shows that for majority of software developers, the communication on project repositories and Twitter is not directly proportional.

The highest correlation value we found is a developer pair with the value (193,105) where 193 indicates the communication counts on project repositories and 105 on Twitter. The Pearson correlation was calculated for that particular pair based on their monthly communication data which resulted in `r=0.522`. Similarly, the lowest correlation value found is a developer pair with value (107,1) which resulted in `r=-0.040`. We calculated the Pearson's correlation for all developer pairs and computed the mean and median value which is shown in Table 2.

Developer pairs	Mean	Median
107	0.191	0.12

Table 2: Mean and Median of Correlation Values

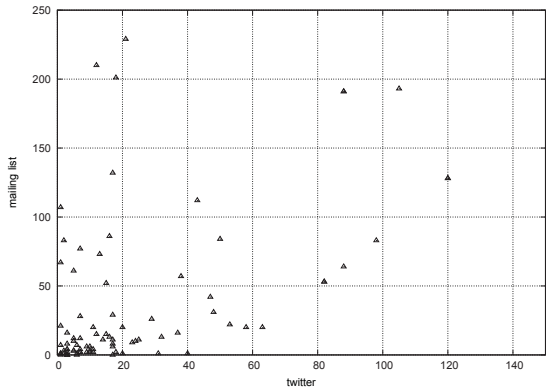


Figure 5: Scatterplot of Twitter vs. Project Repositories Communication Between Software Developers

Based on the results in Figure 5 and Table 2, we found that the correlation between developers communication on Twitter and project repositories is not strong. One potential reason could be the 140 characters limitation on Twitter which keeps software developers to communicate more through traditional communication channels (i.e., mailing list, IRC channels, forums etc.). Other reason might be the usage of Twitter to communicate non-work related activities. However, it is quite interesting to observe how developers communicate with other fellow developers in different communication channels. In our experiment, we found many developers who communicated less on Twitter but their communication on project repositories were strong.

As a next step for future work, we will also take into account the social communication graph of a developer with fellow developers as well as non-developers. Based on that, we will be able to understand if developers communication pattern on Twitter is low in general or if they have less communication only with fellow developers.

4 Conclusion & Future Work

In this paper, we have motivated and introduced a new dimension to the analysis of social dynamics of software developers by taking into account social media channels. The usage of social media channels is becoming popular among open source software community and software developers to disseminate project-related information to a wider audience. This motivated us to investigate if the communicational behavior of software developers is same across different communication channels or different from each other. Our initial results based on the data of 10 different *Apache* projects shows that there is very low correlation between software developer communications across different communication channels. Further, our results shows that the social communication between software developers on Twitter is comparatively low than the traditional communication

channels (i.e., mailing lists, bug tracking repositories etc.).

The work proposed in this paper laid down the idea of taking into account all possible social media channels which developers could possibly use to communicate with each other. Based on that, researchers will be able to measure and compare the hierarchy and centralization of software developers in different communication channels in contrast to previous studies where researchers had been using only mailing lists, bug tracking repositories or discussion forums [24, 12]. Furthermore, integrating social messages/-posts to project-related artifacts will open up new research challenges allowing to analyze the impact of end-user's response on the success/failure of an open source project.

In the near future, we will take into account stackoverflow communication network data and further analyze the behavior of software developers communication patterns across a variety of communication channels.

Acknowledgment

The work presented in this paper has been funded by Science Foundation Ireland under Grant No. SFI/08/CE/I1380 (Lion-2).

References

[1] `http://camel.apache.org/`.

[2] `http://directory.apache.org/`.

[3] `http://felix.apache.org/`.

[4] `http://hadoop.apache.org/`.

[5] `http://logging.apache.org/`.

[6] `http://lucene.apache.org/`.

[7] `http://maven.apache.org/`.

[8] `http://mina.apache.org/`.

[9] `http://myfaces.apache.org/`.

[10] `http://ofbiz.apache.org/`.

[11] M. Conklin. Project entity matching across floss repositories. In *Proceedings of 3rd International Conference on Open Source Systems*, 2007.

[12] K. Crowston, , K. Crowston, and J. Howison. Hierarchy and centralization in free and open source software team communications. *Knowledge Technology Policy*, 18:65–85, 2005.

[13] K. Crowston and J. Howison. The social structure of open source software development teams. In *First Monday*, 2003.

[14] A. E. Hassan. The Road Ahead for Mining Software Repositories. In *Future of Software Maintenance (FoSM) at Int. Conf. on Software Maintenance(ICSM)*, 2008.

[15] A. E. Hassan, A. Mockus, R. C. Holt, and P. M. Johnson. Guest editor's introduction: Special issue on mining software repositories. *IEEE Trans. Softw. Eng.*, 31(6):426–428, 2005.

[16] T. Heath and C. Bizer. Linked data: Evolving the web into a global data space (1st edition). *Synthesis Lectures on the Semantic Web: Theory and Technology*, 1(1):1–136, 2011.

[17] M. Hu and B. Liu. Mining and summarizing customer reviews. In *Proceedings of the tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, pages 168–177, New York, NY, USA, 2004. ACM.

[18] A. Iqbal and M. Hausenblas. Integrating developer-related information across open source repositories. In *IEEE 13th International Conference on Information Reuse and Integration (IRI), 2012*, 2012.

[19] A. Iqbal and M. Hausenblas. Interlinking developer identities within and across open source projects: The linked data approach. *ISRN Software Engineering*, 2013.

[20] A. Iqbal, O. Ureche, M. Hausenblas, and G. Tummarello. LD2SD: Linked Data Driven Software Development. In *International Conference on Software Engineering and Knowledge Engineering (SEKE 09)*, 2009.

[21] G. Klyne, J. J. Carroll, and B. McBride. Resource Description Framework (RDF): Concepts and Abstract Syntax). W3C Recommendation 10 February 2004, RDF Core Working Group, 2004.

[22] Y. Long and K. Siau. Social network structures in open source software development teams. *Journal of Database Management*, 18(2):25–40, 2007.

[23] G. Robles and J. M. Gonzalez-Barahona. Developer identification methods for integrated data from various sources. *SIGSOFT Softw. Eng. Notes*, 30(4):1–5, 2005.

[24] A. Wiggins, J. Howison, and K. Crowston. Social dynamics of floss team communication across channels. In *Fourth International Conference on Open Source Software (IFIP 2.13*, 2008.

Effective Crowdsourcing for Software Feature Ideation in Online Co-Creation Forums

Karthikeyan Rajasekharan, Aditya P Mathur, See-Kiong Ng
Information Systems Technology and Design
Singapore University of Technology and Design
karthikeyan@sutd.edu.sg, aditya_mathur@sutd.edu.sg, ngseekiong@sutd.edu.sg

***Abstract*—Many software companies are creating firm-centric online forums for customer engagement. These forums can be an effective crowdsourcing platform for software product feature ideation and co-creation with the end users. We studied the community interaction data from the ideation forums of two software providers. Link analysis revealed that a small core community was responsible for generating a large proportion of the implemented ideas. This indicated the need to identify key users in the online forum. Our analysis showed the applicability of centrality measures such as *betweenness* in ranking key users. We also found that commenting was likely to produce better community formation amongst the participants than voting.**

Keywords-co-creation; key users; ideation; link analysis; crowdsourcing; social network analysis; expertise ranking; software feature requirements

I. INTRODUCTION

Company-centric online user forums are an attractive platform for company and end-user interactions and offer the potential to co-opt customer knowledge as part of the innovation process. Several consumer goods companies such as Dell, Nike etc. manage online participation communities that help to strengthen their product portfolio through customer-suggested features. In particular, for the Software-as-a-Service (SaaS) arena, the new product development process carries higher risks of market adoption relative to the risks of technical failure. Such company-owned online user forums can be used to help mitigate the market adoption risk by transferring knowledge from the user to the company, thereby enabling better decision making pertaining to creating new customer-centric product features.

User-led innovation has been suggested to be a key part of the ideation process that can lead to breakthrough product features, [1] found that "*that on average user ideas score higher in novelty and customer benefit, but lower in feasibility. Even more interestingly, user ideas are placed more frequently than expected among the very best in terms of novelty and customer benefit.*" [12],[2] argued in favor of taking advantage of online communities for generating ideas and suggest that the system needs to be open and social for it to be successful. In this paper, we focus on the addition of new features to an existing software product through crowdsourcing in firm-centric online forums.

For a company to effectively extract value and manage knowledge creation in such an online community, there are two key questions that merit consideration

1. How can a firm identify the key users for ideation in the online ideation forum?
2. Which of the activities in the online ideation forums are more effective in fostering community formation?

II. DATA GATHERING

To perform the analysis, the online ideation forums of Salesforce.com (SFDC) and SAP were used. Salesforce.com is a leading SaaS provider and as part of its online community SFDC involves end users in its ideation process in a forum entitled Ideaexchange. Given the nature of its business (providing software services over the internet), the company has an active ecosystem of partners and customers who interact with each other and with SFDC in this online forum. SAP also has an active ecosystem and has been pursuing Open Innovation and Crowd-Sourcing as a means of generating new customer insight. SAP's ideation forum was called IdeaPlace.

A. Ideation Forum Structure

The screenshot in Figure 1 shows the structure of an ideation forum using SFDC as an example.

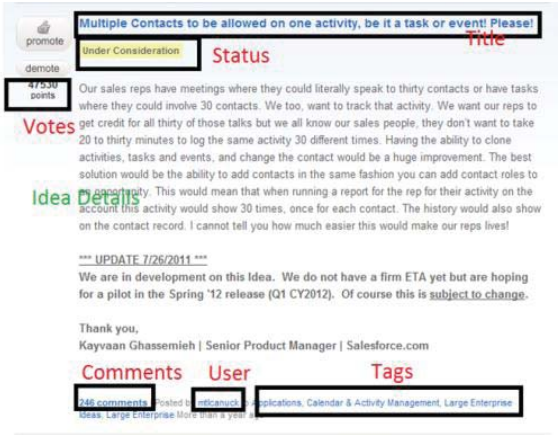


Figure 1. Salesforce Idea Exchange. *(forum structure)*

The key activities that users can perform in such a forum are the suggestion of ideas, voting on ideas (up and down), commenting on ideas and annotating the ideas with meta-data tags. Each idea belongs to a single user and users cannot vote on an idea more than once. They can however, comment on a single idea many times. Each user is uniquely identified by a user identifier. Ideas and comments are linked to the users who created them.

B. *Crawling the Forum*

The forums of Salesforce and SAP were crawled using the Selenium and Scrapy toolkits for publicly available ideation information and the data that was obtained were encapsulated into PostgreSQL databases for further analysis.

C. *Dataset Description*

The datasets that were obtained is described in detail in table I.

TABLE I. IDEATION FORUM DATA THAT WAS GATHERED

Forum	*#Ideas*	*#Participants*	*#Comments*	*#Votes*
SFDC	19,593	73,942	62,389	516,514
SAP	7,506	2,226	7,276	40,765

In the subsequent sections, the discussion will focus on the SFDC dataset; similar results were obtained on the SAP dataset and are summarized in the section on related work.

III. ACTIVITY GRAPH GENERATION

We construct activity graphs (one for voting and one for commenting) from the dataset as follows. Each Node in the activity graph represents a unique user account. The edges in this activity graph correspond to a particular communication activity between two users.

An assumption made in this analysis is that each user account refers to a unique individual. Each node is annotated with properties such as the number of ideas and votes that were contributed by that user.

Each edge in our graph is a reflection of communication between two users in relation to a particular idea. Edges are derived through the procedure illustrated by example below

1. User A makes an idea contribution to the community. User A is identified as the originator and the node's idea count is increased by 1.
2. User B then comments on the idea proposed by User A. Thus, this indicates a communication interaction from User B to User A on his or her idea. An edge is created from User B to User A to capture this interaction. The number of such interactions between the two users will determine the strength of that edge.
3. User C comments on the same idea. Now an edge is drawn between C and A.
4. User D introduces a new Idea. User D's Idea count is incremented but no edges are drawn.
5. User C and User A comment on User D's idea. Edges are drawn from User A and User C to User D.

This is illustrated in the Figure 2. This process is repeated with the voting activity data to obtain the voting activity graph.

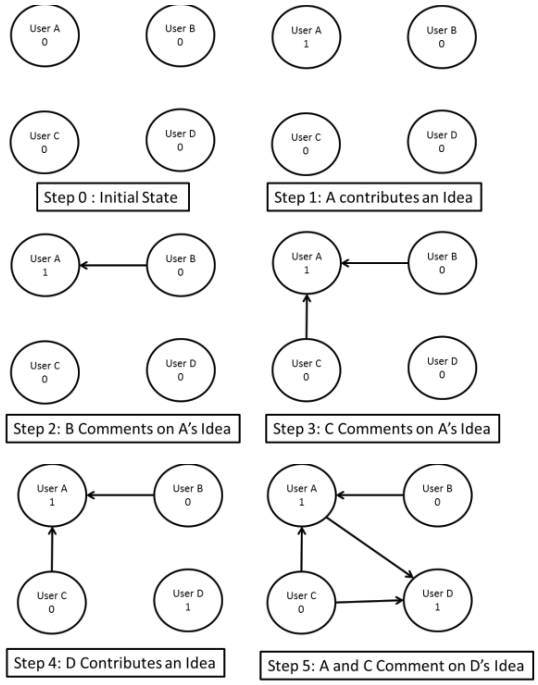
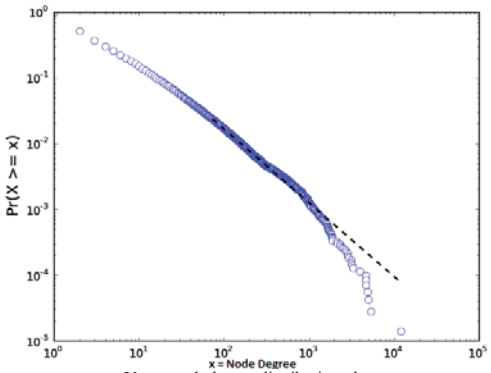
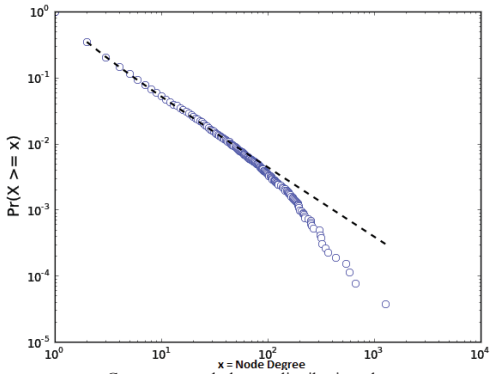


Figure 2. Activity Graph Construction



Vote graph degree distribution plot



Comment graph degree distribution plot

Figure 3. Degree Distribution of Activity Graphs

The activity graphs were visualized using [5] and were found to exhibit a core-periphery structure. There is a highly connected (relative to rest of the graph) core community whose members have diverse interests and connect with the less active periphery community of users. Studying the degree distributions plots of these activity graphs as shown in Figure 3, the activity networks' degree distribution likely follow power law distributions as per the formulation,

$$p(x) \approx x^{-\alpha}$$

Typically, for real world power law distributions, the value of α is between 2 and 3. The α values that were obtained were 2.12 and 2.05 for the vote graph and comment graph respectively. The power law fitting libraries used in [3] were used to make these calculations. This seems to suggest that these are scale free networks within empirical limits and show behavior similar to those observed in other empirical networks in [3].

IV. ISOLATING THE CORE COMMUNITY

Given the observation that the user community structure is that of a core-periphery type, we develop a heuristic algorithm based on average degree of a sub-graph to isolate the core ideation community. Intuitively, the sub-graph that forms the core of the activity graph will have an average degree that is maximal.

Data: A Community Activity Graph G of V nodes and E edges
Result: A sub-graph that contains the core community
set dcutoff = 0 , maxavgdegree = 0, corecutoff = 0 , let D_{max} be the maximum observed degree;
while *dcutoff* $< D_{max}$ **do**
  isolate subgraph G_{sub} with nodes whose degree > dcutoff ;
  compute Avg Degree of G_{sub} as Avg_{temp};
  if $Avg_{temp} > maxavgdegree$ **then**
    maxavgdegree = Avg_{temp};
    corecutoff = dcutoff;
  end
  dcutoff = dcutoff + 1;
end
create subgraph G_{core} consisting of nodes whose degree > corecutoff ;
return G_{core};

Algorithm 1: Core Community Isolation Heuristic

A. Core Community Isolation Results

The above algorithm was applied and the results are shown in Figure 4. The Y axis tracks the value of the Average Degree of the sub-graph and the X axis shows the degree cutoff. Based on the maximal average degree of the sub-graph, we find the degree cutoff points for the core were 150 and 61 for the Vote graph and the Comment graph respectively

Having obtained the degree cutoffs, the core community can be isolated. We used the actual ideation output to evaluate the core community detected. Table II showed that while the core community comprises of a relatively few users, they contribute a significant portion of the ideas that are implemented. This result when combined with the fact that SFDC implemented only 4.3% of the total ideas put forth by the users suggests that it is important to identify the key users in the community for effective ideation co-creation.

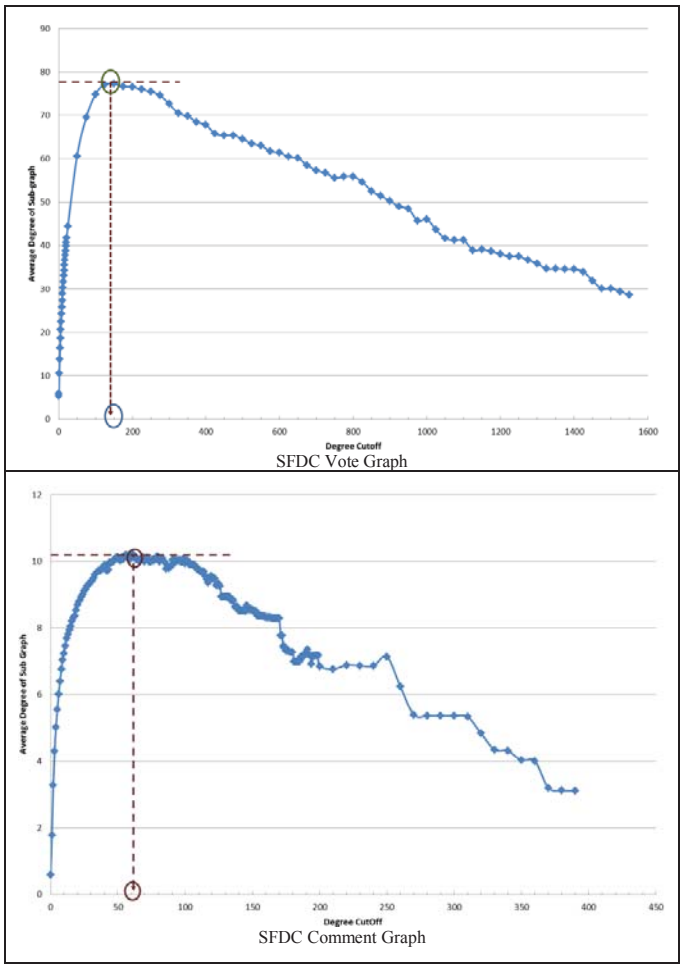


Figure 4. Core Community Isolation

TABLE II. SFDC IDEA EXCHANGE CORE COMMUNITY PERFORMANCE

#Graph	*% of total users in Core*	*Idea contribution fraction of core*	*Implemented Idea fraction of Core*
Vote Graph	0.35%	$\frac{3304}{19593} = 16.8\%$	$\frac{322}{843}$ =38%
Comment Graph	0.68%	$\frac{2885}{19593} = 14.7\%$	$\frac{258}{843} = 31\%$

V. KEY USER RANKING

We conduct link analysis to rank community users for their relative importance. This can be done by calculating the prestige of a node and also by looking at measures of centrality of a node. Structural prestige in network analysis has been the basis for analyzing many networks. [11] details the PageRank algorithm that was used to rank web pages according to structural prestige. [16] proposed the idea of betweenness centrality as a measure of a node's importance in the overall graph.

Based on the original page rank algorithm [11], we define the community activity rank as follows

$$C(i) = (1-d) + d \sum_{(j,i)\in E} \frac{C(j) * W_{j,i}}{\sum W_j}$$

Where C(i) is the community activity rank of node i, E is the set of all edges in the graph, d is the damping factor (set to 0.85), $W_{j,i}$ is the weight of outbound link from node j to i, $\sum W_j$ is the sum of all of the weights of outbound edges from node j. Thus, the final activity rank of a user is dependent on the activity ranks of the users who collaborate with the user in question. The key difference is that the original page rank algorithm didn't cater for edge weights and in our formulation we use a directed graph with weighted edges.

Betweenness Centrality is defined as follows

$$C_b(i) = \sum_{j<k} \frac{P_{jk}(i)}{P_{jk}}$$

Where P_{jk} is the number of shortest paths between j and k and $P_{jk}(i)$ is the number of shortest paths that have node i as part of the path. Thus, Betweenness is a measure of the number of times a node is part of the shortest path between any two other nodes in the graph. The intuition that guides this centrality measure is the idea that a node in the shortest path between two other nodes can influence the flow of information between those two nodes.

A. Ranking Results

The two approaches to ranking users were applied using [5] and the users were ranked. An abbreviated subset of the results (due to space constraints) - the top 10 users - for the comment graph are shown in tables III and IV

TABLE III. SFDC IDEAS COMMENT COMMUNITY RANK TOP 10

#	User Name	Community Activity Rank	Community Recognition
1	Alexander Sutherland	0.019588813	MVP Winter 11
2	Christoph K	0.008686445	None
3	werewolf	0.007827351	MVP Winter 11
4	Andres G	0.007087102	MVP Winter 11
5	jcohen	0.006898484	None
6	TomaszO	0.006523066	None
7	ToddJanzen	0.005924399	SFDC
8	eyewellse	0.005483297	None
9	ErikM	0.005006845	None
10	chris925	0.004876349	None

TABLE IV. SFDC IDEAS COMMENT BETWEENNESS CENTRALITY TOP 10

#	User Name	Betweenness Centrality	Community Recognition
1	Alexander Sutherland	0.019771398	MVP Winter 11
2	Rhonda Ross	0.015190872	MVP Winter 11,12,13
3	Scott J	0.013384961	SFDC
4	Andres G	0.00886717	MVP Winter 11
5	Matthew Lamb	0.008623408	MVP Spring 11
6	AMartin	0.007343319	MVP Spring 11
7	Mattias Nordin	0.005807566	MVP Winter 11,12
8	mattybme1	0.005726696	MVP Winter 11,12,13
9	Christoph K.	0.00516802	None
10	Jakester	0.004668099	None

B. Evaluation of the Ranking

To evaluate the ranking of nodes, a measurement of the firm's evaluation of the importance of a user is useful. Salesforce runs a community recognition program called the MVP program where it periodically chooses members from the community for their outstanding achievements and recognizes them with virtual badges as MVPs. The Salesforce.com website describes the program as "*This program recognizes exceptional individuals within the Salesforce community for their leadership, knowledge, and ongoing contributions. These individuals represent the spirit of the community and what it is all about!"*

In the result tables, the Community Recognition column shows if the individual has been the recipient of any such award. In cases, where the contributor is part of Salesforce, the employee is not eligible for recognition. Such members have also been highlighted. To evaluate the ranking approaches, the MVP recognition of a user can be used to as a qualitative measure. I.e. to what extent can network prestige or centrality be linked to the firm's recognition of individual users.

If the firm's recognition of community member's contribution is the key criteria then the Betweenness measure does much better than the Community Activity Rank measure. Most of the people in the top 10 as ranked by the betweenness measure are already members that the firm (SFDC) has also recognized publicly. This does imply that this could be measure that can potentially be used to identify users who have not been yet recognized. This measure could also be used in a dynamic fashion (as the community grows) to identify newer key users. It is interesting to note that the community rank based approach didn't perform as well as the betweenness centrality measure. While the transfer of prestige from one user to another through out-links has an intuitive appeal, in this instance, it didn't perform as well empirically. [15] performed a similar analysis on the java question and answer forum and report similar findings that in online expertise networks PageRank derivatives did not outperform simpler measures.

The results also pose interesting qualitative questions for analysis. For instance, the user Jakester (number 10 as per betweenness ranking) has suggested 26 ideas, of which 10 have been implemented by SFDC. He has also contributed 534 comments and 771 votes on ideas. It would be of interest to understand the reasons in the decision making process of the firm that led to him not being recognized. In a similar fashion, it would be interesting to understand the motivational impact of having been granted a MVP badge. While, the analysis

covered in this paper didn't evaluate this, it presents an interesting avenue for further research.

Thus, betweenness centrality is a potential tool to answer the first question posed at the start of this paper. In an actual implementation scenario, this metric could be calculated in an offline batch mode for analysis. [6] has proposed a fast way of calculating betweenness centrality that could be used to perform this calculation.

VI. COMPARING VOTING AND COMMENTING

The next key question then is which of the two online forum activities (voting and commenting) encourage a tighter and close knit community to be formed? This question is tied to what motivates users to engage and participate in innovation forums with the firm. If the activity fosters intrinsic motivational factors, then it is likely to be self-sustaining. [13] note that in innovation communities a key motivating factor for users is learning. In [9], Lakhani and Eric Von Hippel studied the Apache Open Source community and report that in their study "*98% of the effort expended by information providers in fact returns direct learning benefits to those providers*".

To evaluate the voting activity against the commenting activity, a measure of community quality is required. [10] uses the notion of conductance as a measure of community quality. According to [10], if A is the adjacency matrix of the graph G = (V, E), then

$$\emptyset(S) = \frac{\sum_{i\in S, j\notin S} A_{ij}}{min\{A(s), A(\overline{s})\}}$$

Where $A(S) = \sum_{i\in S}\sum_{j\in V} A_{ij}$

Conductance is a measure of the intra-community connections versus the inter-community connections. The lower the value of conductance, the better the quality of the community i.e. the community is densely connected internally and sparsely connected to the rest of the graph.

[10] also introduces the notion of a community profile plot. Network Community Profile (NCP) plot characterizes the best possible community over a range of size scales. In this plot, the size of the nodes in a community (community size) is plotted on the x axis and on the y axis the best possible community of the given size (based on conductance) is tracked. Both the axis are on a log scale. In real world networks, the value of conductance decreases initially and then starts to increase. In our analysis, the global minimum of the NCP plot can be a measure of the community formation tendencies of an activity graph. A comparison of the community size at which the global minimum occurred was used to draw conclusions on community formation characteristics of voting and commenting activities.

Using this approach, the activity graphs constructed out of comment and voting data were treated as un-directed graphs and used to create separate network community profile plots. The plots for the vote activity graph and the comment activity graph are shown in the figures 5 and 6 respectively. The SNAP [17] (Stanford Network Analysis Project) toolkit was used to create this plots.

Both the profile plots show the expected behavior of initially decreasing conductance followed by increasing conductance. This is to say that the quality of communities increases with node count for a while and then starts to degenerate. The vote activity graph reaches a community size of 10 nodes when conductance is at the global minimum, while for the comment activity graph; the community size where the global minimum is found is around ~33 nodes. In other words, in the comment activity graph, the highest quality community was found involving up to ~33 users whilst in the vote activity graph, the best community size is comprised of only 10 users.

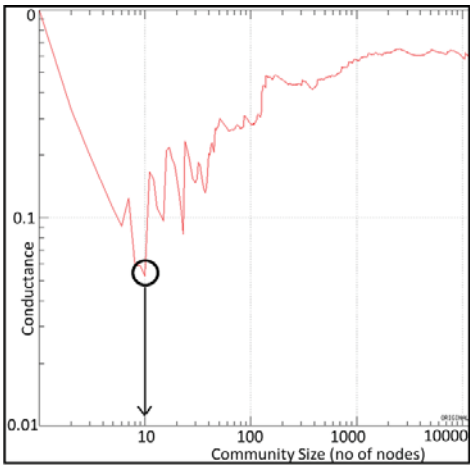


Figure 5. SFDC Vote Graph NCP Profile Plot

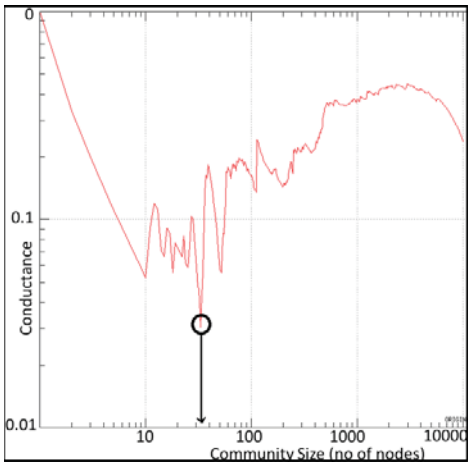


Figure 6. SFDC Comment Graph NCP Profile Plot

This comparison suggests that commenting activity has a higher community creation effect than voting activity. This is to be expected as psychologically, there is higher intrinsic motivations and rewards (through the knowledge gained) for engaging in discourse as opposed to merely voting on an idea. While, this analysis has been based on a single ideation community, it shows the distinction between voting and commenting activity in objective and measurable terms. Further work is required to analyze other ideation networks to understand if similar characteristics are observed there. This result is also in line with [9], [13] which have suggested that a key motivating factor is learning through participation. Such

understanding will be important for designing suitable activity features for the online user forums to be effective ideation co-creation platforms.

VII. Related Work

Similar analysis was performed on the SAP dataset and the following results were obtained. The activity graph also displayed power law distribution of node degree with an α of 2.57 and exhibited similar core-periphery structure. The size of the core community obtained by the heuristic algorithm was 5.4% of the overall community but accounted for 20% of the suggested ideas and 46% of the implemented ideas (only 4% of all suggested ideas were implemented). Qualitative analysis of the ranking also demonstrated that betweenness performed better than the PageRank derived community activity rank.

Many analyses of online networks have used the notions of node prestige to rank and evaluate participants. [7] used a PageRank based approach to identify key users in online communities. [14], [15] applied activity based ranking techniques to the study of expertise in online question and answer forums. In these forums, one user poses a question and other users contribute answers to the posed question. [15] obtained similar results where the PageRank derivatives of node importance did not outperform simpler measures. In their analysis, they found that "z_score" and "z_num" -simple metrics derived from a node's in and out degree- performed best in their dataset. [8] used the notion of out-links as a means of identifying rising stars in bibliography networks. The intuition here is that the nodes in this network (namely researchers) have prestige which they confer on others through their co-authorship and collaboration. [4] analyzed the online ideation community of DELL and concluded that past success likely has detrimental effects on the productivity of new ideas. While much work has been done on online communities, the study of ideation in online communities is still evolving and presents an opportunity for continued research.

VIII. Conclusions

In this paper, we have performed link analyses on the online ideation communities of two software providers for crowdsourcing new product features. We found that most of the implemented ideas were originated from a small core community in the forums. To identify the key users for product feature ideation, we found that Betweenness centrality is a better measure for user ranking than PageRank. We also found that the community cohesion tendencies of commenting activity were higher than that of voting activity. These findings will be useful for designing such company-centric user forums for effective co-creation of new product features.

A. Limitations

The analysis in this paper adopted a static approach to the network activity. In reality, collaborations in online communities weaken / strengthen over time. If two users communicated on a certain task once, it doesn't necessarily imply that the link remains active for their entire lifetime on the community. This could potentially be handled by varying the edge weight as a function of time. This is a potential area for further research.

The analysis of community formation required splitting the community into sub-communities. Other approaches such as those demonstrated by [18] could be used to measure community quality of overlapping communities. These will be evaluated in future work on the data set.

References

[1] MK Poetz and Martin Schreier. The value of crowdsourcing: can users really compete with professionals in generating new product ideas? Journal of Product Innovation, 29(2):245-256, 2012.

[2] Dahlander, Linus, Lars Frederiksen, and Francesco Rullani. "Online communities and open innovation." Industry and innovation 15.2 (2008): 115-123.

[3] Clauset, Aaron, Cosma Rohilla Shalizi, and Mark EJ Newman. "Power-law distributions in empirical data." SIAM review 51.4 (2009): 661-703.

[4] B. Bayus. Crowdsourcing and individual creativity over time: the detrimental effects of past success. Available at SSRN 1667101, 2010.

[5] Mathieu Bastian, Sebastien Heymann, and M Jacomy. Gephi: An open source software for exploring and manipulating networks. In International AAAI Conference on Weblogs and Social Media. Association for the Advancement of Artificial Intelligence, 361-362 ,2009.

[6] Ulrik Brandes. A faster algorithm for betweenness centrality. Journal of Mathematical Sociology, 25(1994):163-177, 2001.

[7] Julia Heidemann, Mathias Klier, and Florian Probst. Identifying key users in online social networks: A PageRank based approach. Information Systems Journal, 4801(December):12-15, 2010.

[8] XL Li, C Foo, K Tew, and SK Ng. Searching for rising stars in bibliography networks. In Database Systems for Advanced Applications,pages 288-292, 2009.

[9] KR Lakhani and Eric Von Hippel. How open source software works:free user-to-user assistance. Research policy, 32(July 2002):923-943, 2003.

[10] Leskovec, Jure, et al. "Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters." Internet Mathematics 6.1 (2009): 29-123.

[11] L Page, S Brin, R Motwani, and T Winograd. The PageRank citation ranking: bringing order to the web. pages 1-17, 1999.

[12] E. Prandelli, M. Swahney, and G. Verona. Collaborating with customers to innovate: conceiving and marketing products in the networking age. Edward Elgar Publishing, 2008.

[13] Anna Stahlbrost and Birgitta Bergvall-Kareborn. Exploring users motivation in innovation communities. International Journal of Entrepreneurship and Innovation Management, 14(4):298-314, 2011.

[14] KK Nam, MS Ackerman, and LA Adamic. Questions in, knowledge in?: a study of naver's question answering community. Human Factors,pages 779-788, 2009.

[15] Jun Zhang, MS Ackerman, and L Adamic. Expertise networks in online communities: structure and algorithms. Proceedings of the 16th international conference on World Wide Web, pages 221-230, 2007

[16] Freeman, Linton. "A set of measures of centrality based on betweenness". Sociometry, 40: (1977):35–41

[17] Stanford Network Analysis Project, http://snap.stanford.edu/index.html

[18] Palla, Gergely, Imre Derényi, Illés Farkas, and Tamás Vicsek. "Uncovering the overlapping community structure of complex networks in nature and society." Nature 435, no. 7043 (2005): 814-818.

Profiles for Convenient Front-end Privacy

Ronald Maier
Dept. of Information Systems
University of Innsbruck
Innsbruck, Austria
ronald.maier@uibk.ac.at

Johannes Sametinger
Dept. of Information Systems – Software Engineering
Johannes Kepler University
Linz, Austria
johannes.sametinger@jku.at

***Abstract*— Privacy can be described as the state of being unaccompanied or unobserved without unauthorized intrusion. We define front-end privacy as privacy when accessing data from a device, e.g., when working jointly on a computer. This is a matter of visibility with the problem that information can get directly disclosed. In this paper, we will define kinds of information that we want to consider for not being disclosed or for being hidden on the screen. Starting from typical knowledge situations we will categorize what we call front-end situations and define risk levels. We then introduce spheres and profiles as a means to effectively and conveniently ensure front-end privacy. Usability and implementation considerations wrap up our approach to tackle this neglected form of privacy.**

Keywords-privacy; front-end privacy; profiles;convenience;

I. INTRODUCTION

In recent years, the share of work that can be characterized as knowledge work (KW) has risen continuously [1], comprising key characteristics of a wide array of activities concerned with creating, translating or applying new knowledge [2,3,4]. KW has strong communication, coordination and cooperation needs, is highly mobile, flexible, distributed and requires a strong yet flexible, personalized and adaptable support by information and communication technologies (ICT) [5]. Consequently, knowledge workers as users of advanced ICT want to have data with them, have full control about access to them, share them when, where and with whom they decide and yet be confident that valuable information and knowledge is protected against unauthorized access and use. Increasing requirements concerning usefulness, ease of use and convenience have been fulfilled with a plethora of ICT solutions that deploy simple mechanisms such as the more data are on the server, the more convenient is the access from wherever or the more data is on a mobile device, the easier it can be accessed by its user. However, users' valuable data are distributed across applications and computers, so that managing access privileges and knowing what happens to the data has become a nuisance to users. While organizations are increasingly aware that strong management of information security on an organizational level is important in today's knowledge economy and while on a societal level there has been substantial debate about privacy or the lack thereof backed by showcases such as the introduction of Google Streetview in Germany [6], awareness of the individual knowledge worker's role in information security management has only recently been put on the radar of information security management initiatives, e.g., [7,8,9]. Early computers were used to solve specific problems, e.g., spreadsheets to perform business calculations. Over the years and decades, we have moved more and more data onto our computers and have long ago reached the point where a work life without computers is not possible any more for most of us. In the early computing days, all data was kept centrally on mainframes. The personal computer has brought computing power onto our desktop, and the amount of data being administered has increased. The Internet has interconnected all these computers and taken more data on the computers, e.g., personal communication via e-mail. There is also a trend to centralize data again. Companies use servers to store and secure information. But also individuals use servers for the purpose of having data available on several client machines, e.g., to synchronize data between servers, both organization-internal and external to an organization, e.g., social business networks like XING or LinkedIn, laptops, smart phones, and home/office computers. Also, people increasingly use subscriptions to information, e.g., to get current weather information or to get current news headlines.

In the course of these developments, many knowledge workers have increasingly given personal information away for convenience and functionality. From the perspective of actual behavior, privacy seems to be a lesser concern for individuals. At the same time, many are annoyed about the lack of convenient possibilities for keeping their information secluded from unwanted access. In addition to the considerations on privacy of data held in remote servers and sent between the servers and the person's device which typically cannot be easily designed by individuals, knowledge workers enjoy freedom to personalize and design their own KW space, the front-end towards the diverse interconnected information spread over heterogeneous systems. People want to design (Gestalt) their KW space (including a network of selected people) in order to increase their individual productivity. They then use these KW spaces in diverse situations with a variety of requirements concerning privacy. It does make a difference whether we access our KW space alone or in company, at home, at an office or in a public place, just to mention a couple of cases. So far, there is a lack of concepts guiding us in secluding information from others while accessing it. Although individual applications offer numerous functionalities in order to decide how much information is disclosed, these functionalities are inconvenient, because knowledge workers typically switch continuously between accessing a plethora of applications with data distributed over many systems in order to pursue their activities.

In this paper, we will address the question on how users can balance convenience and privacy in various situations concerning a (set of) KW space(s) they access for activities they are engaged in. The paper's main aims are to describe barriers that

many knowledge workers face with respect to ensuring privacy of their data while still being able to perform their work tasks, to review opportunities offered by current applications that in combination can address the requirements and to provide concepts helping to design convenient solutions for managing front-end privacy. The paper is structured as follows. Section 2 discusses conflicting priorities between convenience and privacy. Front-end privacy is introduced in Section 3, where we also consider knowledge situations as well as privacy situations. At the end of Section 3 we discuss how users typically handle risks to front-end privacy and present some basic solutions. In Section 4, we introduce spheres and profiles as a means to provide front-end privacy. Usability and realization considerations are given in Section 5. Section 6 concludes the paper.

II. CONVENIENCE AND PRIVACY

Convenience is the "fitness or suitability for performing an action or fulfilling a requirement or something (as an appliance, device, or service) conducive to comfort or ease" [10]. It can be anything that is intended to save resources like time or energy. Inconvenience leads to frustration. The meaning of "convenience" can change over time. Something that is convenient today may be regarded as commodity in the future, e.g., mobile access to the Internet, GPS positioning. Privacy can be described as the quality or state of being apart from company or observation with the freedom from unauthorized intrusion [10]. Cultures and individuals have different considerations of what is private that are generally related with the particular environment in which privacy has to be considered [11], but there are also commonalities. Privacy is related to anonymity, where people remain unnoticed or unidentified in the public or in the Internet. Privacy is also an aspect of security. Privacy and convenience are at odds with each other. According to a poll, almost 70 percent of consumers do not mind if their identities are authenticated when they make a purchase, as long as their personal information is not collected [12]. It is surprising that users feel comfortable with the authentication of their identity. They want to have a secure, trusted transaction experience. Sometimes, they value convenience higher than privacy. Another survey has revealed that the top three privacy concerns of Internet users in the US have not changed from 2002 to 2008 [13]. This is despite the fact, that these years have seen a significant increase in online purchases. But people have become more concerned about the disclosure of their purchasing patterns. They also express a stronger desire to be notified about protection measures of their personally identifiable information. People also have become more concerned about the tendency of web sites to store information about sites that had been visited previously [13]. A requirements taxonomy for the reduction of web site privacy vulnerabilities is given in [14].

There are many examples, where convenience beats privacy, see for example [15]. Amazon can tell its customers what kind of books or music they might be interested in. Also, they do not have to retype their credit card numbers every time they make a purchase. Customers are happy because it is convenient. We often give away private information, because we get something valuable in return, e.g., frequent flyer miles, reduced prices in supermarkets, access from any device to documents stored on the web, exposure to significant others in social networks. We distinguish several forms of privacy:

- Information privacy. Privacy of any information that is personally identifiable, e.g., medical information, financial information, location-based information, information about someone's troubles with the law, lifestyle information, political information.
- Internet privacy. Same as information privacy, but considering any activities on the Internet, e.g., information that is shared in social networks, financial information that is transmitted and disclosed during online banking sessions. In social networks, users often are not aware of specific privacy settings and their consequences. This results in disclosed information without knowledge or consent of users.
- Back-end privacy. Privacy of data and activities on remote peers and servers. This is a matter of storage and whether we trust our service providers.
- Front-end privacy. Privacy when accessing data from a device, e.g., when working jointly on a computer. This is a matter of visibility with the problem that information gets directly disclosed. We usually want to see everything when alone, but want to hide specific things when not alone.
- Connection-based privacy. Privacy of data that is transferred between a local device and a remote machine. This is a matter of leaving data traces with the known problem of eavesdropping, e.g., unencrypted transfer of data like e-mail messages.
- Administrational privacy. Privacy of data protected from being disclosed to administrators. This is about access rights and needed trust to our administrators.

Privacy impact assessments are a public reaction against privacy-invasive actions of governments and corporations [16]. People increasingly want to know about organizations' activities and to have more control over their accesses.

III. FRONT-END PRIVACY

Front-end privacy is about information that we want to be displayed on our screens in specific situations.

A. *Information on the Screen*

Before we proceed, we have to define kinds of information that we want to consider for being hidden on the screen.

- File system. Branches of the file system can contain information that we do not want to be disclosed, i.e., we do not even want someone else to see that there is some information. Seeing the name of the file or folder can already be too much information. And, we do not want to have to say no when somebody else, e.g., our boss, asks us to open a specific folder or file.
- Applications. We sometimes do not want to disclose the fact to someone else that we have installed a specific application. For example, we do not want to unfold that we have a weakness for playing poker. Information about applications includes the start menu, control panel entries, desktop shortcuts, etc. In some situations, it is okay to disclose that we have an application installed (and often will not be seen anyway), but we

do not want to disclose the fact that we are currently running this application or that we have recently or frequently used it. With some applications, the opposite might be true so that we do not want to get caught not having (sufficiently) used the application.

- Application-specific information. Some applications manage extensive information and we want to disclose this information only partially. This is especially true for office applications like browsers, e-mail, contacts, and calendar. Information we do not want to unfold to everyone include specific calendar entries, specific contact information, parts of the browser history, etc.
- Alerts and Notifications. Alerts and notifications are given by the operating system or by applications. We treat them separately, because there are situations where we do not want to be disturbed by them at all.
- Connections. Which connections can be seen and used? At what level of detail can they be accessed?

B. Knowledge Situations

In organizations, it is common that administrators have full access to computers of employees. This makes administration of the ICT infrastructure much easier. Additionally, administrators can conveniently solve employees' ICT problems by using remote desktop connections, i.e., by having full control over employees' machines. Thus, administrators have potential access to an enormous amount of private data. We can argue that a computer paid for by a company is not supposed to have any private data on it. But even if this is the case, there are many possibilities to invade someone's privacy, e.g., data about what someone is currently working on, e-mail messages, and logon/logoff times. All too often we are unable to decide ourselves about how much privacy we are willing to barter for perceived value in general and convenience in particular. This is the case when using the infrastructure of our employer. When we use web sites and services like those provided by Amazon, Apple, Facebook or Google, then we often have two choices. Either we refrain from using the site or service. Or we give up privacy in accordance with the site or service provider. In addition to administrators and service providers, people around us to whom we can, should or must disclose our screen can intrude our privacy. When, for example, we sit in our office working on our computer, and someone is approaching us in order to make an appointment, we open our calendar and reveal information about delicate appointments. While disclosure of private data to administrators and service providers typically is on the radar of information security professionals and often dealt with using, e.g., awareness raising measures, policies and software-based counter-measures, the decision about how much information our work environment should disclose to us and others around us is not a simple one. Example privacy situations with potentially increasing attention from outsiders towards what happens on our screen are:

- Isolated. We work privately and isolated from other people, e.g., in a personal office or at home.
- Office work. We are surrounded by colleagues.
- Meeting. We are in a meeting with participants sitting together having our mobile devices in front of us.
- Public. We work in a public location, with outsiders being able to freely roam next to our mobile devices.
- Approached. We get approached by uninvited persons.
- Presentation. We present slides in front of people.
- Joint work. We jointly work on our computer.

These examples show how diverse situations can be among which people can switch. For example, in a team that closely works together, people would be expected to instantly switch between isolated work, approached and joint work as well as office work, joint work, ad-hoc meetings and presentations, just to mention a few switches that might occur frequently every day. The members of the team are not necessarily limited to organization-internal people. Advanced collaboration technologies foster virtual teams so that members can switch between these situations without being geographically collocated, e.g., using co-authoring tools, multi-party videoconference, screen and application sharing systems such as Adobe Connect, GoogleDocs, Microsoft Sharepoint or Skype. Our list of example privacy situations is non-exhaustive and can be extended almost to an infinitely long list of specific situations describing (slightly) different requirements towards privacy. However, with respect to the consequences for profiling and for restricting access of others to valuable personal information, the depicted dimensions seem to sufficiently describe a set of circumstances of privacy that seems conveniently manageable.

For example, we get informed about the fact that a new update from software XY is available as an inconvenient interrupt during a presentation. Or we get a notification about the arrival of an e-mail message, maybe even including information about its sender and the subject. Or we use the browser and reveal information about our browsing history. If we sit in front of our computer and jointly write an e-mail message, all too often this reveals much information about our e-mail traffic, like senders, receivers, subject lines, used e-mail accounts, folders, etc. As teachers at the university we sometimes need to look up an individual entry in a list like student grades in a spreadsheet. How do we get this information with the student next to us without revealing information that is not intended for this person? In contrast to the situation where we do something with someone else in front of our computer (known intruders), there are situations where someone approaches us without invitation (unknown intruders). If we work in a public place, people sitting next to us will see information that we are not willing to reveal, e.g., in an airplane, at the airport, at a conference. How do we foil uninvited people from staring at our screen? There are solutions that limit visibility of screens horizontally, however, this also encumbers joint work sharing the screen and people sitting behind us will still be able to observe our screen.

C. Front-end Situations

Applications typically operate based on the assumption that once a user is authenticated and authorized, she is the only one in front of the computer or all other persons that get a glance of the screen are authorized to see all information that the application reveals. In other words, the application interacts with the user without any consideration of the circumstances of the situation at hand. Take the example of online banking which not by coincidence has been termed "home banking". If a user

is in a public environment and simply wants to transfer an amount of money, this is often not possible without revealing further financial information such as the current balance on all accounts or the most recent transactions. We distinguish the front-end privacy situations *exposed*, *surrounded*, *together*, and *secluded*, see Fig. 1.

- Exposed. The display is exposed to the looks of others. For example, we make a presentation, or someone approaches us in order to set up an appointment.
- Surrounded. We are surrounded by people. The display is not directly exposed to the looks of others, but people can get a glimpse of what is on the screen. We may be approached rather quickly, e.g., we work in the lounge of an airport or in some other public place.
- Together. We are together with people we know and with whom we work together, e.g., in project meetings.
- Secluded. The screen cannot be seen by other people, e.g., we work alone in our office or at home.

The distinction among these situations is not always clear, but a finer granularity will counteract convenience.

D. Front-end Risks

The term risk is discussed heterogeneously and focuses either on its causes or its impacts. Risk is related to any business operation and includes possible losses that result from the realization of uncertain undesired events [17]. Risk can also be defined as a condition in which a deviation from a desired outcome can occur [18]. Deviations can refer to targets, plans or results of a decision. A potential positive deviation is considered as opportunity and a negative as threat or risk in a narrow sense. Moreover, risks can be characterized by a probability that an undesired event occurs and an extent of loss that goes along with the occurrence of this event [17]. Risk management typically comprises identification, assessment, governance and evaluation as basic steps [19]. Risk management has also been discussed with respect to knowledge assets, e.g., [20,21], and is part of formal organizational initiatives such as information security management initiatives, e.g., [7,8]. Risk management includes measures about how to deal with risks, specifically avoidance, acceptance, transfer, reduction, e.g., [22], based on [23]. Examples in this paper's domain are:

- Reduction**.** Privacy risks can be reduced, e.g., by using private acronyms and "cryptic" expressions in a calendar. Thus, if other people see the calendar entries, they at least do not immediately make sense to them.
- Acceptance. Many simply accept risks. If someone sends a job application via email from her current employer, then there is definitely a risk. Sometimes this is simply ignorance or lack of technological knowledge, i.e., the facts that email messages are sent via plain text and that administrators can easily read them.
- Avoidance. Risks can be avoided, for example, by sending text (SMS) from a private telephone instead of using email or by not storing selected sensitive information on a computer at work.
- Transfer. Risks can also be transferred to another party, for example, by insuring against problems resulting from information being disclosed to competitors. However, this is associated with additional costs.

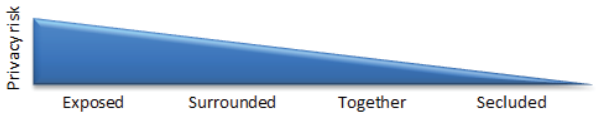


Figure 1. Risk levels of front-end privacy situations

Most individual users will not go to the trouble of formally assessing their risks and deploying corresponding personal risk management measures. Instead, users resort to convenient measures in order to save time and energy and interfere as little as possible with the pursuit of their goals. Concerning connection-based privacy, users on the one hand typically trust secure connections like https or VPN and on the other hand do not have any alternatives in regular transactions they are engaged in. Concerning administrational privacy, many users are aware that the policies in place are not sufficient and that they have to take into account the possibility that administrators access their data beyond administrational duties. While users are increasingly aware of back-end privacy issues, it seems like most people are not yet aware of potential breaches of front-end privacy which are consequently not dealt with appropriately.

E. Partial Front-end Solutions

There are solutions available that solve parts of the problem. For example, MindGems offers *Boss Key,* an application that hides and restores windows when pressing a hotkey. It covers tracks of running programs, even from the task bar and the system tray. These applications are kept hidden in the background and can later be restored from the point of interruption [24]. MS Outlook provides the possibilities to create several folders for mails, contacts, calendars and task lists. These folders can be given different permissions and they can be visualized in overlay mode. But all data still resides on one single server. It is not possible to combine information from two different servers, say one for a highly confidential research project and another one for corporate use.

IV. Spheres and Profiles

Data, applications and network connections can be used for several purposes. Some of them are closer to one purpose than to another. For example, a project manager uses project management software, presentation software and a collaboration client to manage data about a project and to communicate with project members while she uses an enterprise resource planning software accessing data on personnel, material and other resources for administrative purposes. In this case, the workspace can straightforwardly be divided into two parts. However, the two parts can overlap due to using the collaboration client also for exchanging data with administrative or controlling units.

A. Spheres

The phenomenon of overlapping workspaces can be described with the metaphor of information spheres. The term sphere means "any of the concentric and eccentric revolving spherical transparent shells in which according to ancient astronomy stars, sun, planets, and moon are set", a "natural, normal, or proper place", "an area or range over or within which

someone or something acts, exists, or has influence or significance" [10]. An information sphere is a collection of objects, specifically information (contents, documents etc.), applications and pieces of information used in these applications. Objects in spheres "gravitate" around a user-defined purpose. Examples are a project or an ongoing individual or collective activity, e.g., joint experience or collaboration with persons. Spheres have conceptual rather than physical boundaries and are transparent with respect to where, i.e. on which devices or machines, objects are located. Using the metaphorical analogy of the term denoting parts of the celestial system, gravitational forces metaphorically mean that objects are attracted to other objects. Objects potentially belong to several spheres as gravitational forces from objects in several spheres can be at work. The definition of spheres is inherently a user task. But automatic definition can be supported by classifications based on machine learning algorithms that are similar to the ones that are used for task detection. The following dimensions can help to guide a user in defining a sphere:

- When. Time is the primary dimension for presenting objects. It is the pace making dimension of KW.
- Why. Purpose is a leading dimension because it defines the center of a sphere and the gravitational forces that are at work. It is the sense making dimension of KW.
- Who. The dimension about persons and institutions defines who else is involved in a sphere. For example, it influences access rights to others' collections of objects or allocation of communication acts to spheres. This is the networking dimension of KW.
- How. Format and type define how knowledge is materialized and is the representational dimension of KW.
- What. Topic and domain define what a sphere is about thematically and thus is the content dimension of KW.
- Where. Location primarily acts as a proxy for contextual factors that come into play when accessing a sphere in a specific geographical location. It is the situational dimension of KW.

Meta-information about documents contains values for these dimensions, e.g., time of document creation or modification, names of authors, file types. The risks of front-end privacy breaches can be reduced by defining and activating spheres and consequently by restricting data, applications and connections that are disclosed in a situation to those that are part of the currently activated sphere. In order to simplify this process we suggest profiles, which are described in the subsequent section.

B. *Profiles*

We suggest the use of profiles that we see as filters and set them in order to selectively reduce the amount of information being disclosed on the screen. For that purpose, we define ranges for the dimensions presented in the previous subsection. Values can be including or excluding. For example, let's say we define the range 2013 in the dimension "when". This includes everything within the year 2013. The range "not 2013" has the opposite meaning. We can filter persons or organizations in the "who" dimension. The how dimension lets us filter specific file formats or information items like calendar items, etc. Figure 2 shows a profile sample where we include everything in the year 2013 that is about front-end privacy and has the persons R. Maier and J. Sametinger assigned. Additionally, we restrict the profile to text documents, images, slide presentations, and email messages. It will be useful to let users combine file extension like .gif, .jpg, .png and others to an image category, or any office document to an office category.

Dimension	Filter
When	2013
Why	Front-end privacy
Who	R. Maier, J. Sametinger
How	.doc, .ppt, .jpg, email
What	-
Where	-

Figure 2. Sample profile

Some values in our dimensions can easily be filled automatically, e.g., the when dimensions. Every file in a file system has dates associated with it, e.g., date of creation or last modification. Persons and organizations can also be retrieved quite easily. The why and what dimensions are more difficult to fill. Users are not willing to manually fill in meta-information for their documents. Simple automatisms and heuristics have to be used for that purpose. For example, folder or file names can be used for the why dimension. In the example of Fig. 2, this will include everything that is stored in folders named "front-end privacy". The "what" dimension defines topics. For user convenience, we suggest to let users define values for this dimension arbitrarily. For the "why" dimension we suggest heuristics to be defined in order to assign useful values. The dimension "where" can be seen in many different ways. It can mean the location where information is stored. This can be a geographic location or the name of a server or a cloud. It can also be the location that is assigned to a person or organization. For example, Linz and Innsbruck can be used in that dimension and can, thus, include all persons that live or work in these towns. In this case, contact information can be used to find out who is at which location. However, contact information usually illustrates the current situation. If someone moves from one location to another, then automatic assignments become more difficult and error-prone. For information filtering, we have to define the effects of filters for different types of information.

- File system. Meta-information about files and the names of enclosing folders for the "why" dimension.
- Applications. Applications can be restricted to the ones that are necessary to work on documents that pass a specific filter. Separate application profiles can also be used in order to hide applications in specific situations.
- Application-specific information. Applications often store information in big files. If stored separately, meta-information about the separate files could be used for filtering. Thus, filtering can be done outside the application. If single files are used, then the application will have to do the filtering.
- Alerts and Notifications. Applications running in the background can send alerts and notifications at any time. If applications are filtered, then their alerts and notifications should be postponed until the application becomes active, i.e., unfiltered, again.

V. USABILITY AND REALIZATION

In this section we will discuss the usability of filters and how profiles can be implemented.

A. Usability

Even though front-end privacy is important to users in various situations, they are usually not willing to expend any extra effort for that purpose. Particularly, they cannot be expected to fill in meta-information for documents. As mentioned earlier, we have to define simple heuristics in order to provide values for specific dimensions. We imagine a widget with 4 front-end privacy situations, the 3 most commonly used profiles, and the other ones available via pop-up. These should be pre-configured along the lines described above and the widget should offer the possibility to create new and configure existing profiles. We also imagine an intrusion hotkey. Profiles should be switchable via hotkeys and they should not (always) be visible on the screen, because others get suspicious if they realize that we switch to not-alone when they are approaching. If we add a new contact or create a new file, it can automatically be assigned to our current spheres or spheres are determined by the category we assign it to in our mail system.

B. Realization

It is not possible to develop an application that can implement the functionality on top of current operating systems and applications. Spheres and profiles are hard to implement without modifications in an operating system. The file system has to filter files according to active profiles. Application-specific information is specific to applications only because these applications, i.e., their developers, have decided to do so. For example, contact and date entries can be stored as file each in a specific location on disk. If filtering is done on the file level, then applications do not have to care about filtering. Alerts and notifications are usually done by calling operating system functions that open a dialog or display a message. If the operating system had the information about active profiles and knows that the call of a function, i.e., displaying information, is in order to provide an alert or a notification, then this information can be postponed until an application becomes unfiltered again. Apple's notification center provides some of this functionality.

VI. CONCLUSION

We discussed conflicting priorities of privacy and convenience. After the description of various forms of privacy we introduced front-end privacy in more detail, considering knowledge situations, front-end situations, and front-end risks. There are rudimentary approaches for front-end privacy, but they are far from being helpful in typical settings and practices of knowledge workers. We then described spheres and profiles that can provide front-end privacy conveniently and effectively. We outlined possible solutions but we have so far stood back from implementing such a solution, because we need some extensions to the operating system with additional interfaces for applications. And, not all but some of today's most frequently used applications, with the MS office suite as a prominent example, will have to store information in a different way. We believe, however, that our reflections on front-end privacy are helpful to embark on a learning journey of how to improve their front-end privacy. Our suggested changes are not really extensive, yet would greatly enhance chances that users not only are aware of front-end privacy, but can readily deal with the issues they identify. If we attribute enough importance to this form of privacy, software vendors will follow suit.

REFERENCES

[1] E.N. Wolff, The growth of information workers. CACM, 48, 37-42, 2005.

[2] P.F. Drucker, Landmarks of Tomorrow, New York, Harper, 1959.

[3] E.K. Kelloway, J. Barling, Knowledge Work as organizational behavior. International Journal of Management Reviews, 2, 287-304, 2000.

[4] U. Schultze, On Knowledge Work. In: HOLSAPPLE, C. W. (ed.) Handbook on Knowledge Management. Berlin Springer, 2003.

[5] R. Maier, T. Hädrich, R. Peinl, Enterprise Knowledge Infrastructures, Berlin, Springer, 2009.

[6] Spiegel, Google Launches Street View Germany, Spiegel Online International, 11/18, 2010. http://www.spiegel.de/international/business/0,1518,729793,00.html

[7] M.E. Johnson, E. Goetz, Embedding Information Security into the Organization. IEEE Security & Privacy, 16-24, 2007.

[8] Q. Ma, A.C. Johnston, J.M. Pearson, Information security management objectives and practices: a parsimonious framework. Information Management & Computer Security, 16, 251-270, 2008.

[9] M. Siponen, R. Willison, Information security management standards: Problems and solutions. Information & Mgmt., 46, 267–270, 2009.

[10] Merriam-Webster, 11th Collegiate Dictionary, 2004.

[11] E.L. Bertino, D. Lin, W. Jiang, A Survey of Quantification of Privacy Preserving Data Mining Algorithms. In: AGGARWAL, C. C. Y., PHILIP S. (ed.) Privacy-Preserving Data Mining: Models and Algorithms. Ney York: Springer, 2008.

[12] D. Takahashi, Web users will trade privacy for security and convenience, VentureBeat, September 2009. http://digital.venturebeat.com/2009/09/15/web-users-will-trade-off-privacy-for-security-and-convenience/

[13] A.I. Antón, J.B. Earp, J.D. Young, How Internet Users' Privacy Concerns Have Evolved since 2002, IEEE Security & Privacy, Vol. 8, No. 1, pp. 21-27, Jan/Feb 2010. http://doi.ieeecomputersociety.org/10.1109/MSP.2010.38

[14] A.I. Antón, J.B. Earp, A Requirements Taxonomy for Reducing Web Site Privacy Vulnerabilities, Requirements Eng. J., Vol. 9, No. 3, pp. 169–185, 2004.

[15] T. Weber, Will convenience beat privacy? BBC News, January 2006. http://news.bbc.co.uk/2/hi/business/4649292.stm

[16] R. Clarke, Privacy impact assessment: Its origins and development, Computer Law & Security Review, Vol. 25, Issue 2, 123-135, 2009. doi:10.1016/j.clsr.2009.02.002

[17] S. Kaplan, B.J. Garrick, On the quantitative definition of risk. Risk Analysis, 1, 11-27, 1981.

[18] R.R. Gallati. Risk mgmt. and capital adequacy, NY, McGraw-Hill, 2003.

[19] C.A. Archbold, Managing the bottom line: risk management in policing. Policing: An International Journal of Police Strategies & Management, 28, 30-48, 2005.

[20] J. Jordan, J. Lowe, Protecting Strategic Knowledge: Insights from Collaborative Agreements in the Aerospace Sector. Technology Analysis and Strategic Management, 16, 241-259, 2004.

[21] K.C. Desouza, G.K. Vanapalli, Securing Knowledge in Organizations. In: DESOUZA, K. C. (ed.) New frontiers of knowledge management. Basingstoke: Palgrave Macmillan, 2005.

[22] D. Baccarini, G. Salm, P.E.D. Love, Identification and Management of Risks in Information Technology Projects. 14th Australasian Conference on Information Systems. Perth, Australia, 2003.

[23] H. Zhi, Risk management for overseas construction projects. International Journal of Project Management, 13, 231-237, 1995.

[24] MindGems, BossKey, http://www.mindgems.com/products/Boss-Key/boss-key.htm

A Real-time Personalized Gesture Interaction System Using Wii Remote and Kinect for Tiled-Display Environment

Yihua Lou, Wenjun Wu
State Key Laboratory of Software Development Environment
Beihang University
Beijing, China
{louyh, wwj}@nlsde.buaa.edu.cn

***Abstract*—Gesture interaction is more convenient than traditional input methods in tiled-display environments. As the advance of somatosensory technologies, more and more somatosensory devices such as Wii Remote and Kinect that can be used in gesture interaction are commercially available. We present a real-time personalized gesture interaction system targeting for two-handed gestures for tiled-display environment in this paper. By applying the dynamic time warping (DTW) algorithm, the system does not require a special training stage and is easy to use. Experiment shows that the combination of acceleration data from Wii Remote and skeleton point data from Kinect leads to higher recognition accuracy with lower recognition error rate, and the system can also recognize an ongoing gesture before it ends while the recognition accuracy is kept high enough. Our usage of this system in a real tiled-display environment shows that the system is practical.**

Keywords - Gesture recognition, Tiled-display environment, Dynamic time warping, Kinect, Wii Remote

I. INTRODUCTION

Nowadays tiled-display environments are commonly used in demonstrations and scientific discussions. Such tiled-display facilities usually have extremely large physical size and resolution, and are often driven by tiled-display software such as SAGE [1]. Traditional PC oriented input methods such as mice are not well suited for interaction scenarios in these environments. Instead, it demands alternative interaction models such as gesture based interaction that can offer smoother and more intuitive user experiences.

In contrast to the device-based interaction, the design of gesture-based interaction elicits new technical challenges. Especially, gestures are usually dependent upon individuals, which suggests that people may use different gestures to express the same intention and action, and it is difficult to define a standard gesture vocabulary except a few simple or single-handed gestures. However, in a practical tiled-display environment, it is not adequate to only design single-handed gestures for human-display interaction, as many actions such as zooming and rotation of visual objects in the display should be performed in a two-handed manner. Moreover, even the trajectory of a same gesture from the same individual may vary from time to time.

Traditionally, it is not very convenient to set up a gesture interaction system with wired devices such as data gloves. However, as the advance of somatosensory technologies, now there are many commercially available somatosensory devices such as Wii Remote and Kinect, which significantly simplify the effort to establish a gesture interaction system. Wii Remote is a wireless accessory of Nintendo Wii, which integrates a 3-axis accelerometer which can measure the acceleration from -3g to +3g at 100Hz with a noise below 3.5mg [2]. With this accelerometer, Wii Remote can offer researchers to utilize the acceleration data as an input source for gesture recognition. Kinect, the accessory for Microsoft XBox 360, provides even more powerful sensing features. With the built-in skeleton tracking engine, one Kinect can simultaneously track the movement of six human figures at 30fps, two of which can be retrieved with detailed skeleton position information [3]. The motion of skeleton joints is also a good data source for gesture recognition.

In this paper, we present a new personalized interaction system for tiled-display environments. This system will take advantage of both data sources, the acceleration captured by Wii Remote and the skeleton position captured by Kinect, to improve the accuracy of gesture recognition. Moreover, we use DTW, a non-statistical recognition algorithm, to avoid the long-time training process required by other machine learning algorithms such as HMM. The system is capable for recognizing both two-handed and single-handed gestures, and it employs an ongoing gesture recognition process, which is not mentioned in other relevant researches.

The rest of the paper is organized as follows: In section 2, we will review some related works and literatures. Then the design aspects and the overview of the system are presented in section 3. In section 4, the design of the core recognition algorithm is described and section 5 gives the experiments and analyzes the results. The conclusion is in section 6.

II. RELATED WORKS

Gesture recognition is not a new technology and has been widely investigated. Though the definitions of gesture are not consistent in the research community, the majority of these works define a gesture as the locus of hand movement. Computer vision based technologies [4] have been the most popular research area of gesture recognition, as building a gesture recognition system based on cameras is easy and cheap, and there exist many excellent pattern recognition tools working with videos and images. However, the greatest disadvantage of computer vision based technologies is its expensive cost of computation.

In recent years, Wii Remote or Kinect based gesture recognition is becoming a new research focus, and most of such researches focus mainly on single-handed gestures. An acceleration-based personalized gesture recognition system called "uWave" was proposed in [5]. By collecting over 4000 simple single-handed gesture samples from eight users, the authors of this work proved that their proposed DTW-based algorithm can achieve over 98% accuracy. In [6], another Wii Remote based gesture recognition approach was proposed and implemented in a library called "wiigee" [7]. The gestures evaluated in this work are more complex and the overall accuracy of the proposed HMM-based algorithm is proved to be above 85%. In [8], a Kinect based gesture interaction interface with high accuracy in recognizing eight simple gestures was proposed by Lai et al. Another Kinect based gesture interaction system was proposed in [9].

As the core part of gesture recognition, the choice of recognition algorithm is also important. The most popular algorithm for gesture recognition is HMM [10], which is good for user-independent gesture recognition with a set of pre-defined gestures. However, like other statistical model based algorithms, it requires a large training dataset to build the model before recognition, otherwise its effectiveness will be greatly downgraded. Therefore HMM is not suitable for personalized gesture recognition, as in such scenarios each user may create his own gesture freely, which makes the separate training process unavailable. In the contrast, DTW, a dynamic programming based algorithm that has been widely used in speech recognition, does not rely on training process, and thus is more suitable for personalized gesture recognition. DTW introduces a distance calculation mechanism to calculate the distance between two time series – an input time series and a pre-stored time series called "template". This procedure is called "template matching" and the distance between the two time series is called the "matching cost". A smaller matching cost indicates that the two time series are more similar, and the template that has the smallest matching cost with the input time series is called the "matched template".

III. SYSTEM DESIGN

A. Design Aspects

1) Easy-to-use

As the large tiled-display environment is sometimes used for technical demonstrations or ad-hoc discussions where users may not be well-trained before they begin to access such systems, therefore the gesture interaction system should be easy-to-use and does not require a long training stage.

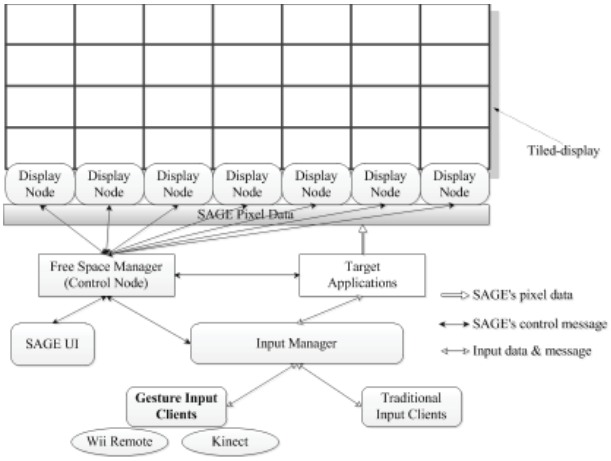


Figure 1. Architecture of the system

2) Personalized

As described in section 1, gestures are fully personalized. Therefore, the system should allow users to define their own gestures instead of restricting the users in a limited range of gestures. Moreover, the system should be able to identify its current user and recognize his gesture by matching the data with his gesture templates.

3) Two-handed

As described above, most current research proposals on gesture interaction focus mainly on single-handed gesture recognition. Although single-handed gestures are sufficient in many cases, especially small and medium size displays, some gestures such as zooming and rotation designed for interacting with large visual objects rendered in a tiled display often involve coordinating movement of two hands.

4) Ongoing gesture

Many gesture recognition researches only aim to improving the recognition accuracy for the motion trajectory pattern. However, in a practical environment, if an ongoing gesture can be correctly classified, the features in the rest gesture motion can be extracted as supplementary attributes of the gesture to calculate a continuous update on the rendered visual object in the focus. For example, given a rotation gesture defined to spin a window on the tiled-display, the rotation angles of a classified ongoing rotation gesture can be measured continuously for a more smooth progressive rotation rendering than a sudden window rotation.

B. System Overview

The system shown in Figure 1 is an extension of the multi-user interaction system in [11] with a gesture input client added. The gesture input client receives motion data from Kinect and Wii Remote and recognizes gestures from the inputs. It can interact with both the windows in the virtual desktop provided by SAGE and the visual contents rendered by applications. The processing flow of the gesture input client is shown in Figure 2. It contains a pre-processing module to filter and quantize the original input data, a core module for gesture classification using

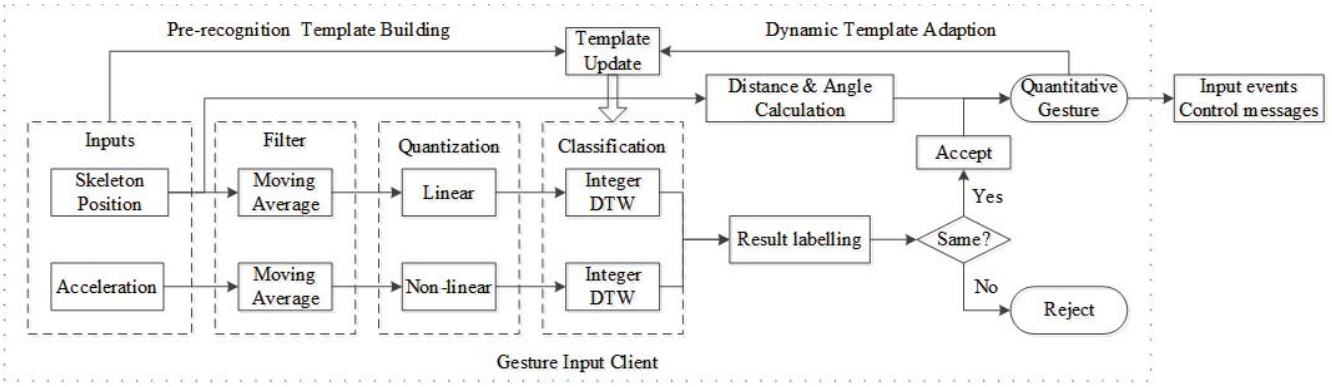


Figure 2. Processing flow of the Gesutre Input Client

the DTW algorithm, and a post-processing module that handles DTW template adaption.

IV. Algorithm Design

A. Feature Selection

The first important consideration for a gesture interaction system is to select appropriate features for recognition. In this paper, we use a combination of the acceleration data provided by Wii Remote and the human skeleton position data provided by Kinect, as that shown in Figure 3.

The feature selection of acceleration data is simple, as the Wii Remote and its accessory Nunchuk both can provide a 3-axis acceleration data independently, the two 3-axis acceleration data can be packed into a 6-dimensional feature vector. Assume the acceleration data of the left hand and the right hand are (a_x^l, a_y^l, a_z^l) and (a_x^r, a_y^r, a_z^r) respectively. Then the feature vector can be defined as follows:

$$\vec{f}_{acc} = (a_x^l, a_y^l, a_z^l, a_x^r, a_y^r, a_z^r). \quad (1)$$

However, the feature selection of skeleton position data is more complex. As the skeleton position data reported by Kinect are represented by 3-D coordinates relative to the center of the Kinect sensor, they cannot be used directly as we need the body-relative features. Therefore, the distances between the coordinates of the two hand joints and the spine joint are used, which can be packed into a 6-dimensional feature vector. Assume that (x^l, y^l, z^l), (x^r, y^r, z^r), and (x^s, y^s, z^s) are the coordinates of the left-hand joint, the right-hand joint and the spine joint respectively, and d_x^l, d_y^l, d_z^l, d_x^r, d_y^r, d_z^r are defined as follows:

$$d_x^l = x^l - x^s, d_y^l = y^l - y^s, d_z^l = z^l - z^s,$$
$$d_x^r = x^r - x^s, d_y^r = y^r - y^s, d_z^r = z^r - z^s. \quad (2)$$

TABLE I. Quantization of acceleration data

Original acceleration value	Quantized value
a > +2g	31
+g < a ≤ +2g	21 to 30 (linearly)
-g ≤ a ≤ +g	-20 to 20 (linearly)
-2g ≤ a < -g	-30 to -21 (linearly)
a < -2g	-31

Then the feature vector can be computed as follows:

$$\vec{f}_{pos} = (d_x^l, d_y^l, d_z^l, d_x^r, d_y^r, d_z^r), \quad (3)$$

B. Data Filter and Quantization

As the data from Kinect and Wii Remote contain noise that may degrade the recognition accuracy, a moving-average filter is employed to filter the original noisy data. The filter window size is set to five samples for acceleration data and three samples for skeleton position data, and the filter window moves at a step of one data sample.

Then, the filtered floating-point data should be quantized into discrete integers to improve the algorithm efficiency by eliminating the time-consuming floating-point computation in DTW. As we found that in normal gesture samples, there are far more acceleration values within the range of [-g, +g] than those without the range, and a non-linear approach is introduced to quantizing the acceleration data. It converts the acceleration values between -3g and +3g into 63 levels between -31 and +31, as shown in Table I. When quantizing skeleton position data, a linear approach is used, as we found that the distance values are distributed relatively equally in the range of [-1m, +1m]. It converts the distance values between -1m and +1m into 61 linear levels between -30 and +30, with an additional level -31 for distance values smaller than -1m and +31 for that larger than +1m.

C. Dynamic Time Warping

Given the different sampling rate of the acceleration data and the skeleton position data, and the high complexity of processing

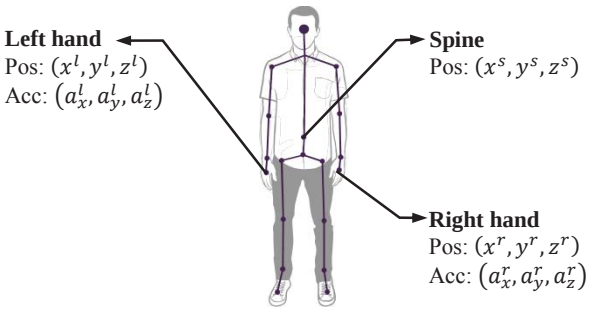


Figure 3. Features used in the proposed system

high dimensional vectors, they should not be just simply combined together. Therefore both types of data are processed by DTW for gesture classification independently, and then the preliminary results are combined based on a simple classification rule.

To simplify process procedure, the Euclidean distance is used as the DTW distance function in both DTWs. For the acceleration data, assume that there are two feature vectors $\vec{\boldsymbol{f}}_{acc}$ and $\vec{\boldsymbol{f}}'_{acc}$, then the DTW distance is defined as follows:

$$d_{acc} = \left\| \vec{\boldsymbol{f}}_{acc} - \vec{\boldsymbol{f}}'_{acc} \right\|^2. \quad (4)$$

Note that as the input data are all quantized into integer values, the distance used in DTW will also be the square of the actual Euclidean distance to avoid non-integer results. Similarly, the DTW distance between two skeleton position feature vectors $\vec{\boldsymbol{f}}_{pos}$ and $\vec{\boldsymbol{f}}'_{pos}$ can be defined as follows:

$$d_{pos} = \left\| \vec{\boldsymbol{f}}_{pos} - \vec{\boldsymbol{f}}'_{pos} \right\|^2. \quad (5)$$

As the distance between two single vectors is defined, the distance between two time series can also be defined. Assume that $\boldsymbol{S}[1 \dots M]$ and $\boldsymbol{T}[1 \dots N]$ denote two time series of feature vectors, then a $M \times N$ matrix $\boldsymbol{M}$ can be produced with each element $\boldsymbol{M}_{i,j}$ being the distance between $\boldsymbol{S}_i$ and $\boldsymbol{T}_j$. Given a monotonic path from $\boldsymbol{M}_{1,1}$ to $\boldsymbol{M}_{M,N}$, the cost of the path is the sum of all the distance values along on the path. The distance between the two time series is defined as the minimal cost from all possible costs. By using the dynamic programming technique, both the space and time complexity of one DTW distance computation process are $O(M \times N)$. After the distances between the input time series and each pre-stored template are calculated, the template that has the minimal distance with the input time series is selected as the matched template, and the input time series can be labeled with the same gesture label as the matched template has.

After recognition processes are done independently on the acceleration data and the skeleton data for a user's single action, there will be two gestures labels generated by DTW. Only if the two labels are same, this action can be classified as a gesture, otherwise it will be rejected.

D. Template Adaption

Due to the use of the template matching in DTW based gesture recognition, the accuracy of the pre-stored templates will directly affect the recognition accuracy. However, it is impossible to create valid and static templates, since the acceleration data and the skeleton position data may vary each time when a user performs a gesture. The templates should be adaptive to ensure maximal recognition accuracy and a dynamic template adaption is introduced.

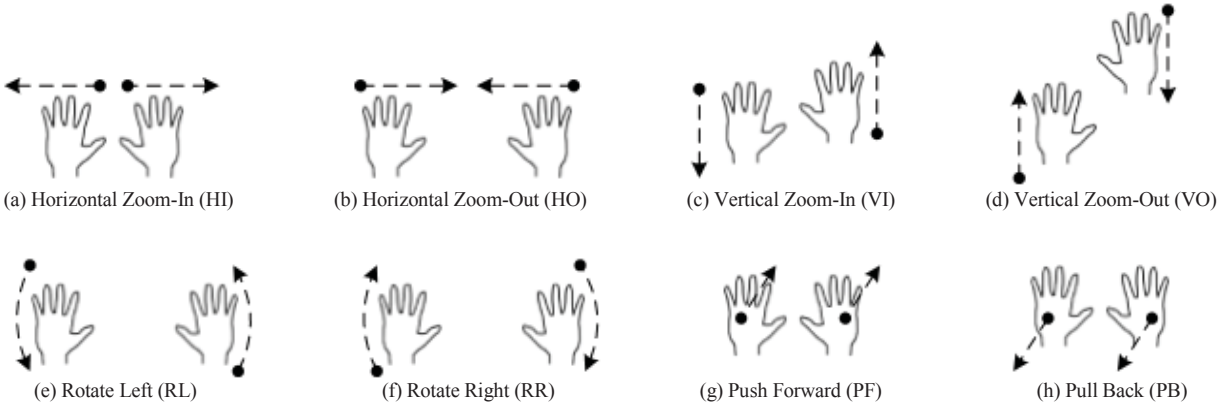
During the normal recognition process, after a gesture is successfully recognized, its time series data are temporary stored. If there are two continuous rejections or three accumulative rejections for a given gesture, and the number of temporary stored time series data is at least three for this gesture, then the dynamic template adaption process of this gesture is invoked. The process is done by the following steps: First, calculate the DTW distances between each two temporary stored time series including the old template's time series. Then, calculate each time series' accumulated DTW distance to all the other time series, and the one with the minimal accumulated DTW distance is chosen as the new template for the gesture. Finally, remove all the temporary stored time series, and reset the numbers of continuous and accumulative rejections to zero.

V. Experiments and Analysis

A. Experiment Environment and Dataset

To evaluate the performance of the proposed system, we implemented gesture recognition prototype using Visual C++ and Kinect SDK 1.6 for Windows. The prototype uses the press and release of Wii Remote's 'A' button to indicate the start and end of a gesture. The experiment environment includes a ThinkPad T420s laptop computer with a Core-i5 2520M CPU, a Wii Remote controller with the Nunchuk extension and a Kinect for Windows sensor.

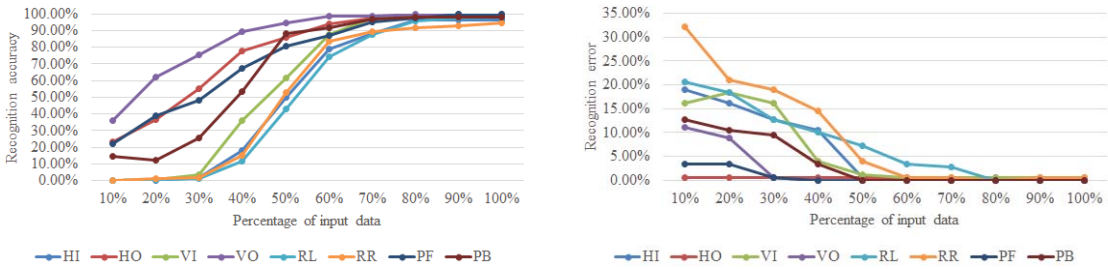
After setting up the environment, we collected a dataset of eight two-handed gestures shown in Figure 4 from six individuals (two females and four males) through five days. Each participant is asked to stand in front of the Kinect, holding the Wii Remote in his right hand and the Nunchuk in his left hand. And the collected dataset consists of 2400 skeleton position samples and 2400 acceleration samples.



(a) Horizontal Zoom-In (HI) (b) Horizontal Zoom-Out (HO) (c) Vertical Zoom-In (VI) (d) Vertical Zoom-Out (VO)

(e) Rotate Left (RL) (f) Rotate Right (RR) (g) Push Forward (PF) (h) Pull Back (PB)

Figure 4. Gestures used in experiments



(a) Recognition accuracy (b) Recognition error

Figure 5. Gestures used in experiments

B. *Gesture Recognition Accuracy*

First, we evaluate the recognition accuracy of data sampled from different days, with the first sample of each gesture from the first day as the initial template. The recognition accuracy in percentage is shown in Table II, in which the row and column represent the actual gesture and the recognized gesture respectively. The table illustrates that, with the use of dynamic template adaption, the recognition accuracy will increase by about 3%, while the error rate and the rejection rate decrease by 0.7% and 2.3% respectively.

If the template and the input data were both sampled on the same date, the recognition accuracy can be significantly increased, as shown in Table III. As the recognition accuracy without dynamic template adaption has already been higher than 98%, the use of dynamic template adaption does not affect the final recognition accuracy at all.

The above results reveal that the recognition accuracy of the DTW classifier created on data gathered from the same date is much higher than that created on data from different dates, which is apparently a proof that the gesture trajectory from the same individual may vary from time to time. And the lower recognition accuracy of the two rotation gestures may be caused by the higher variance of their input data, as we found the average DTW distances between two samples of the two gestures are much higher than those of others.

Then we compare the error rate of using different input sources without dynamic template adaption in Table IV. The results illustrate that the error rates of the gesture classifiers using single input source are much higher than the proposed classifier using the combination of the two types of input sources, regardless of whether the data is captured either on the same day or different days.

TABLE II. RECOGNITION ACCURACY FOR DATA FROM DIFFERENT DATES

	Without template adaption			**With template adaption**		
	Correct	***Error***	***Rejected***	***Correct***	***Error***	***Rejected***
HI	94.4%	0.0%	5.6%	96.7%	0.0%	3.3%
HO	95.6%	0.5%	3.9%	95.6%	0.5%	3.9%
VI	98.9%	0.0%	1.1%	98.9%	0.0%	1.1%
VO	98.9%	0.0%	1.1%	98.9%	0.0%	1.1%
RL	76.1%	7.8%	16.1%	78.3%	4.5%	17.2%
RR	78.9%	6.1%	15.0%	92.8%	3.9%	3.3%
PF	97.8%	0.0%	2.2%	98.9%	0.0%	1.1%
PB	91.7%	0.0%	8.3%	96.1%	0.0%	3.9%
Avg	91.5%	1.8%	6.7%	94.5%	1.1%	4.4%

The above results support our design assumption that the combination of multiple gesture data sources such as the skeleton position data and the acceleration data effectively eliminates many recognition errors that are normally unavoidable in other single-type input source based systems. As a tradeoff, this method increases the rejection probability in the gesture classification. We believe that it is better to reject uncertain actions than to recognize them as wrong gestures, as wrong recognitions may cause some unexpected behaviors in practical environments while rejection will not.

C. *Ongoing Gesture Recognition Accuracy*

As mentioned above, if an on-going gesture can be correctly recognized before finished, it may improve user experience. Therefore, we will then evaluate is how early a gesture can be correctly identified with a low recognition error rate. We use the data from the same day without dynamic template adaption and the result is shown in Figure 5. From the figure we can see that if the thresholds of recognition accuracy and recognition error rate are set to 85% and 5% respectively, then it is enough to use only the first 70% of the data items. This means that in a practical environment, a gesture can be recognized before the user ends his action and the following actions the user takes can be used as a guide.

TABLE III. RECOGNITION ACCURACY FOR DATA FROM THE SAME DATE

	Without template adaption			**With template adaption**		
	Correct	***Error***	***Rejected***	***Correct***	***Error***	***Rejected***
HI	96.7%	0.0%	3.3%	96.7%	0.0%	3.3%
HO	99.4%	0.6%	0.0%	99.4%	0.6%	0.0%
VI	99.4%	0.6%	0.0%	99.4%	0.6%	0.0%
VO	99.4%	0.0%	0.6%	99.4%	0.0%	0.6%
RL	97.8%	0.0%	2.2%	97.8%	0.0%	2.2%
RR	94.4%	0.6%	5.0%	94.4%	0.6%	5.0%
PF	99.4%	0.0%	0.6%	99.4%	0.0%	0.6%
PB	98.3%	0.0%	1.7%	98.3%	0.0%	1.7%
Avg	98.1%	0.2%	1.7%	98.1%	0.2%	1.7%

TABLE IV. ERROR RATE OF USING DIFFERENT INPUT SOURCES

	Acceleration		**Skeleton position**		**Combination**	
	Same day	***Different days***	***Same day***	***Different days***	***Same day***	***Different days***
HI	0.6%	1.7%	3.3%	5.6%	0.0%	0.0%
HO	0.6%	0.6%	0.6%	4.4%	0.6%	0.5%
VI	0.6%	0.0%	0.6%	1.1%	0.6%	0.0%
VO	0.6%	0.6%	0.0%	0.6%	0.0%	0.0%
RL	1.7%	15.6%	0.6%	19.4%	0.0%	7.8%
RR	4.4%	21.1%	2.8%	8.3%	0.6%	6.1%
PF	0.0%	2.2%	0.6%	0.0%	0.0%	0.0%
PB	0.0%	2.8%	1.7%	5.6%	0.0%	0.0%
Avg	1.0%	5.6%	1.2%	5.6%	0.2%	1.8%

Figure 6. Usage of the system in a tiled-display environment

D. Practical evaluation

We have tested the system in our SAGE-based tiled-display environment. As displayed in Figure 6, a user holding a Wii Remote with his hands standing in front of a Kinect camera was interacting with multiple window objects rendered on the tiled-display. Given the large size of the tile display, a scaling ratio between the physical gesture amplitude and the rendering effect can be set by user before using the system. For example, the user can set a scaling ratio for the zoom-in gesture to 100 pixels per centimeter, which means when he performs a zoom-in gesture on a window object, the target window is zoomed in by 100 pixels if the distance between his two hands increases by 1cm. And as the result of our ongoing gesture recognition feature, as long as the user does not perform the zoom-in gesture too fast, he may see that the zoom-in process begins before his gesture finishes and the window zooms in continuously as his gesture goes.

We also evaluated the performance of the system, and result shows that in normal cases the system can recognize a gesture within 2ms.

VI. Conslution and Future Work

In this paper, we proposed a personalized gesture interaction system for the tiled-display environment, which recognizes gestures based on both acceleration data from Wii Remote and skeleton data from Kinect. A DTW based real-time gesture recognition algorithm is designed and implemented, which can fast recognize user-dependent gestures in a high accuracy by using the carefully selected features. And in comparison with other gesture recognition systems that only focus on recognizing gesture itself, the proposed system can use the additional information of an ongoing gesture such as rotation angle and movement range which is useful in practical scenarios. As both Wii Remote and Kinect are commercially available consumer devices, this system can be easily deployed in any SAGE-based tiled-display environments.

We are now working on adding the user-identification function to the system. By enabling this feature, the system can automatically recognize a user and apply his gesture template instead of manually switching.

Acknowledgment

This work was supported by the State Key Laboratory of Software Development Environment Funding No. SKLSDE-2013ZX-03.

References

[1] B. Jeong, L. Renambot, R. Jagodic, R. Singh, J. Aguilera, A. Johnson and J. Leigh, "High-performance dynamic graphics streaming for scalable adaptive graphics environment," in Proc. Supercomputing, 2006.

[2] Analog Device, "Small, low power, 3-Axis ±3g MEMS® accelerometer: ADXL330 datasheet," 2006.

[3] Microsoft, "Skeletal Tracking," http://msdn.microsoft.com/en-us/library/hh973074.aspx.

[4] S. Calinon and A. Billard, "Recognition and Reproduction of Gestures using a Probabilistic Framework combining PCA, ICA and HMM," in Proc. ICML, 2005.

[5] J. Liu, L. Zhong, J. Wickramasuriya and V. Vasudevan, "uWave: Accelerometer-based personalized gesture recognition and its applications," Pervasive and Mobile Computing, vol. 5, pp. 657-675, 2009.

[6] T. Schlöme, B. Poppinga, N. Henze and S. Boll, "Gesture Recognition with a Wii Controller," in Proc. TEI, 2008.

[7] B. Poppinga and T. Schlömer, "wiigee - A java-based gesture recognition library for the Wii remote," http://www.wiigee.org.

[8] K. Lai, J. Konrad and P. Ishwar, "A gesture-driven computer interface using Kinect," in Proc. SSIAI, 2012.

[9] M. Caon, Y. Yue, J. Tscherrig, E. Mugellini and O. Khaled, "Context-Aware 3D Gesture Interaction Based on Multiple Kinects," in Proc. AMBIENT, 2011.

[10] S. Mitra and T. Acharya, "Gesture Recognition: A Survey," IEEE Transactions on Systems, Man, and Cybernetics – Part C: Applications and Reviews, vol. 37, pp. 311-324, May 2007.

[11] Y. Lou, W. Wu and H. Zhang, "Magic Input: A Multi-user Interaction System for SAGE Based Large Tiled-display Environment," in Proc. ICMEW, 2012.

Releasing the OMCS-Br Knowledgebase to Facilitate Insertion of Culture in Applications: Brazilian Experience

Andre de O. Bueno
Dep of Computing - Federal Universityof São Carlos
São Carlos, Brazil
andre.obueno@dc.ufscar.br

Junia C. Anacleto
Dep of Computing - Federal University of São Carlos
São Carlos, Brazil
junia@dc.ufscar.br

***Abstract*—In order to contribute with developers on developing culturally contextualized software, we describe the reengineering done in the Open Mind Common Sense in Brazil's (OMCS-Br) architecture that aims to make the task of using the project's cultural knowledgebase easier. At the same time,we provide a new way to collect the users' cultural knowledge from those culturally contextualized applications to feedback the OMCS-Br knowledgebase. Two new modules are being developed and integrated into the OMCS-Br's API: (i) a cultural filter, that receives the search query from the application with that user's profile and gives back the resulting cultural knowledge, modeled as a ConceptNent and (ii) a knowledge collector, responsible for feeding the OMCS-Br knowledgebase back with the data generated by the contextualized applications. A third result is a new web interface that was created to allow any usertodo searches at the OMCS-Br knowledgebase easy and intuitively.**

Keywords-cultural contextualization; commonsense;OMCS-Br

I. INTRODUCTION

Every day computers are becoming more and more present in our lives. By saying "computers", we want to encompass cellphones, smartphone and tablets. Everywhere we look, we see people using some of these devices to do a large range of activities, going from work to hobbies.In order to make software to these devices that better suit to the users, HCI researchers have been studying different approaches in the software development process. Anumber of approaches had been proposed, and many of them emphasizing only the technological aspect of this relationship.

Other aspects of social life like culture and behavior also are being observed and used in the ICT development process, i.e., humans are also being studied [1]. Some of these studies take in consideration the people's culture, because cultural differences are recognized to play a very important role in the correspondence between the computer interfaces with user expectations of different cultural backgrounds [2]. Systems quality and culture significantly affect trust in the ICT artifact and therefore in their adoption [3]. The Advanced Interaction Laboratory (LIA) at Federal University of São Carlos (UFSCar) in Brazil has created the OMCS-Br (Open Mind Common Sense in Brazil) project to collect common sense, a certain type of cultural knowledge from ordinary people on the web. Such project aims at building a knowledgebase of the Brazilian culture in a continuously and collaborative way through the project's website since 2005 [2]. Once there is this database, software developers can access it and use its data to develop contextualized applications, i.e., it becomes possible to use the user's own culture as requirement to build applications, making them more user friendly and easy to interact with, as stated by researchers since then in some of the published work as in references [4], [5] and [6]. However, some problems have been detected along the years, especially when related to the accessing process of the data stored in the OMCS-Br knowledgebase. Nowadays, the access is restricted to researchers responsible for the project, i.e., LIA's researchers only. Furthermore, it's hard to create new ways of collecting culture from users to enlarge the size of such knowledgebase.

We propose here the development of two new modules to be added to the OMCS-Br project, which are: a cultural filter (responsible for recovering contextualized data from the knowledgebase) and a knowledge collector (responsible for collecting into the OMCS-Br knowledgebase the data generated by applications using the filter). With these upgrades, we try to solve a recurring problem faced by the developers when developing an application using the knowledgebase of this project, which is filtering the used data in the knowledgebase according to the user profile. At the same time, the data from the user will be collected and inserted at the OMCS-Br database automatically, expanding the database size.Besides these two new modules, a web interface tool will allow non-developers to do searches at the knowledgebase in an easy and intuitive way. Designers for instance, could find information in the OMCS-Br knowledgebase about users that will be using their computer systems and, with that, it would be possible for them to create some personas as user's characterization resource.

II. RELATED WORK

Once Marvin Minsk said that "today, our robots are like toys, they just do simple things they are programmed to do, but it is evident that they are about to cross that barrier. People have fool reasons to explain why computers still do not think. The answer is that we still do not program them correctly... They just do not have enough common sense" [8]. Some research focusing on collecting common sense has appeared. Here, common sense is understood as the not specialized knowledge, shared by people in a social group in a determined period, which describes the culture considering the experiences, knowledge related to aspects spatial, physical and social in a community, not susceptible to judgments, representing aspects of a particular cultural group [2].

Cyc (http://www.cyc.com/) is one of the first projects trying to collect and to store common sense, using a specific language

and it is not open to the public. ThoughtTreasure(http://en.wikipedia.org/wiki/ThoughtTreasure) is another platform for natural language processing and common sense reasoning. Currently the project server is offline, what makes its use impossible.MediaLab-MIT started a project called Open Mind Common Sense (OMCS), which takes into account that anyone can contribute providing their common sense, turning the construction of a knowledgebase into a web collaborative work involving people in this challenge[9]. The data is stored using some structures based on the human knowledge representation model proposed by Marvin Minsky[9]. In order to use the knowledgebase, the researchers developed an API (Application Programming Interface). Nonetheless its use is restricted to the researchers' laboratory, i.e., they are the only ones who can access and use the data provided by the project and the API.

The main concern of these three projects is the creation of a knowledgebase to be used in machine learning. In this context, the LIA in Brazil started the OMCS-Br project, which is derived from OMCS-MIT, but has a different focus. In this case, its goal is enabling the creation of culturally contextualized applications, that is, using the user's common sense in creating applications that will later be better suited to the user's need and comprehension. To reach such goal, it is necessary to store the profile of the contributors, except name and other data that could eventually identify that person. The project will be better explained in the next section.

III. THE OMCS-BR ARCHITECTURE

OMCS-Br is a project that started as a collaboartionwith OMCS-MIT, and has been developed by LIA/UFSCar since August 2005. The project works on five work fronts: (1) common sense collection, (2) knowledge representation, (3) knowledge manipulation, (4) access and (5) use. Figure 1 illustrates the project architecture. First of all, common sense knowledge is collected through a website developed by the LIA's researches (http://sensocmoum.ufscar.br). The knowledge is then represented as a semantic network called ConceptNet, according to Minsky's model of knowledge. To build the semantic networks, the natural language statements are exported through an Export Module and sent to the Semantic Network Generator. The generator is composed of three modules: the Extraction, Normalization and Relaxation modules. The result of these three modules is a group of binary predicates. For dealing with Portuguese natural language processing, a natural language parser for Portuguese is used. The predicates are filtered according to parameters of the user's profile, such as gender, age and level of education, generating different ConceptNet according to these data. In order to make inferences over the ConceptNet, several inference mechanisms, which are grouped in an API (Application Programming Interface), were developed. The access to the API functions is made through a Management Server, which makes available access to instances of APIs associated with different ConceptNet. Through this access, applications use API inference mechanisms to perform common sense reasoning [2]. Besides collect statements from users, and model them in a ConcepNet, we also store the anonymized profile of the contributor. From that, we are able to make some inferences over the data like what people from 18 to 35 year old think about movies, or what people from Rio de Janeiro think about how to preserve the environment. To have a certain set of knowledge from a defined profile, it is necessary that the data in the knowledgebase is filtered at the moment of the search, which is usually a time consuming task to be processed. However, the access to the ConceptNet content is only possible through the API and, in this case, it can only be accessed by researchers who are part of the project. It is necessary to provide easy and practical ways to software developers or even to the public in general to use such knowledgebase.

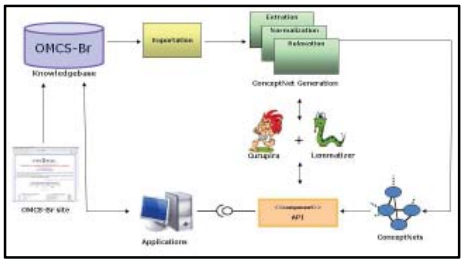


Figure 1 - OMCS-Br Project Architecture

IV. THE CURRENT USE OF THE OMCS-BR KNOWLEDGEBASE

Currently,to use the OMCS-Br knowledgebase the developer must be part of the LIA research group or contact the group. Also, it is necessary for the developer to create a method in the application to filter the data according to the target context, not a trivial task.The currently adopted strategy to filter a certain set of statements from the knowledge baseneeds to be standardized. Each developer implements its own filter in the application, whatleadsto each developer doing the same task every time a new application is developed. Observing this situation it becomes clear the necessity and urgency to create away to facilitate and standardize this access to the database and, also, to release this access for any developer who wishes to use the cultural data stored in the knowledgebase.

V. APPLICATIONS USING THE OMCS-BR

Culturallycontextualized OMCS-BR based ICT applications developed at LIA vary greatly from one to another, ranging from educational games to therapeutic applications. The target audience in these cases varies widely as well, because each of these applications has a different purpose, place and target user. Some of these applications that make use of the OMCS-Br knowledgebase are the quiz game "What is it?", the educational game "Contexteller", and the therapeutic tool called "FamilySense".

The quiz game called "What is it?" [4] focuses on helping teachers to approach the transversal themes defined by the official curriculum for elementary and high school: sexual education, ethics, healthcare, environment, cultural plurality, market and consumers. Figure 2 shows the player's interface. The game was the first application developed by LIA that allows teachers to contextualize the content of their classes according to the students' local culture. Teachers can set up the web quiz game based on the common sense knowledge from the OMCS-Br database corresponding to their students profile. Such knowledge is filtered according to the desired profile

related to age, gender, region of the country, to fit the teacher's pedagogical goals, i.e., the statements to compose the cards are contextualized to the target group. This was the first process implementing the filter based on the users' profile to provide cultural sensitiveness and context relevance.

Figure 2 - "What is it?" quiz game.

For the "What is it? Game,the OMCS-Br architecture was modified so that the filter could be developed (Figure 3). Such solution for contextualizing the application to a certain cultural group considered that the necessary filter requests the OMCS-Br application the ConceptNet from the statements of those contributors who attend the profile established by the filter module. The system saves the 10 structures previously generated so that, in case the requested ConceptNet from the filter was requested before, the processing time for the application is drastically decreased. This is a very simple policy implemented as a prototype that urgently needs to be refined.

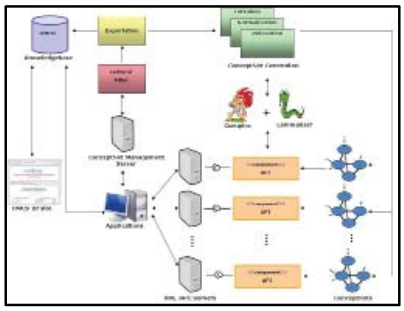


Figure 3 - The OMCS-Br Architecture and the Filter

Contexteller [5] is a RPG-like narrative game which allows cooperation between players on storytelling, aiming at self expression and socialization. The game has as participants a master, who has access to cultural knowledge to compose the profile of the character according to the players' cultural knowledge as well as access to the knowledgebase in real time to inquire about certain statements or characters mentioned in the narration by other players. The web-based therapist tool FamilySense is the third application to be used for families that may not be able to attend a therapy session together [6], it is a card game thatuses cultural information in order to help therapist to compose questions and alternatives to be questioned and answered by a parent and a child under therapy to promote their mutual knowledge leading to identify feelings, facilitate the expression and promote emotional closeness among the participants. Each instance of the game can be contextualized for that family's cultural values consideringtheir socioeconomic and cultural reality.

The successful experience on culturally contextualized software for better suit to the users' need lead us to realize that such resource of knowledge built from a collaborative effort on the web should be available to all developer not only the data themselves but also a tool to support their work on providing cultural contextualized software we believe is the next wave of software development. That is the motivation for developing the cultural filter presented here. Although at this moment we are developing for Brazilian Portuguese cultural contextualization, there are other projects collecting cultural knowledge in other languages that can benefit from the results of this research.

VI. CULTURAL FILTER, COLLECTOR AND WEB TOOL

To make a culturally contextualized application using the OMCS-Br knowledgebase, it is necessary that the referred data from the user's profile that will be using the application in question is informed while doing the searches at the project database to obtain the specific data to that user and, then, this data will be used during the application creation. By following this rule, the returned data will be culturally contextualized according to that profile (Figure 4). Once implemented, this filter will be integrated into the project API (responsible for making possible the access to the knowledgebase). Through this change, the developer no longer needs to worry about filtering data, he needs only to be sure that his application informs the user's profile data to the filter. Then, the filter which is connected to the project API will perform the filtering, search and it will return the culturally contextualized data to the application.

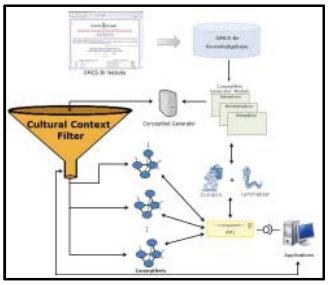


Figure 4 - The Cultural Filter

Thinking about developing new ways of collecting common sense from people, "participants must be able to contribute with their common sense in many ways, they should be able to provide their knowledge through a friendly interface that looks invisible" [9]. Since the OMCS project creation, a series of new projects had emerged trying different approaches to gather knowledge from the general public [4,5,6], using templates (pre-assembled structures) in natural language based on Minky's model of mind in the OMCS project [9]. We propose the creation of a module called Knowledge Collector. This module will be responsible for collecting the data generated in culturally contextualized applications that use the data from OMCS-Br knowledgebase. Through this new approach, the data generated by applications will be used to feedback the OMCS-Br project knowledgebase. With these changes, we expect to increase the data collection for the OMCS-Br project, which is important to the project.

Aiming at making the collected cultural knowledge available, wedeveloped a web interface tool to enable non-developers to search the OMCS-Br knowledgebase in an easy and intuitive way. Through this tool, it is possible for any people and even designers to search in the project database. Accessing the web tool, they will type the concept they want to know about and do the search. If they want to, they can do advanced searches by giving more information on age, gender, schooling, and the search will happen only among the data collected from the users who meet these requirements. In this interface, the user will type a concept and then push a button to do a search at the OMCS-Br knowledgebase. The results will be shown in two different ways in the screen to the user: 1 - A list of concepts related to the one searched for; 2 - a graph with the searched concept as being the root of it. If the user wants to perform a more specific search, he can choose among several options presented by the tool to verify the projection of the concept, its consequences, its purpose, where it is found, etc., as shown in Figure 5.It's possible to see at figure 5 the list of resulting concepts at the left, and the graph at the middle. In the right, we can see buttons that show different forms of graphs. In the graph, the lines that connect the head node (the entered concept) to its children nodes represent the type of Minsky's relation between these concepts. These relations are expressed on each one of these lines by its name, which helps the user in understanding that relations among them. By implementing this web interface tool, we want to make available to the general public the knowledge that somehow belongs to them and to which they usually do not have an easy access. Once users have the possibility to access this data and use it as they want to, they will see how important their contributions are to the project and, maybe, they will feel more engaged in helping to build the project knowledgebase by providing their common sense at the OMCS-Br website.

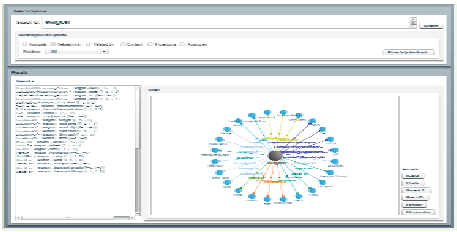


Figure 5- The Web Interface Tool

VII. CONCLUSION

As results of this work, we can see three main contributions: 1. Helping both researchers and developers in general on using the OMCS-Br knowledgebase by the development of a cultural filter. With this change, it is possible to use the OMCS-Br knowledgebasefor culturally contextualizedapplications. 2. A new way to collect common sense from users (the knowledge collector) is inserted into the applications,responsible for feedbacking the data generated into the OMCS-Br knowledgebase. Using these applications with the knowledge collector, it is generated data collected and inserted back at the OMCS-Br database. By doing that, we will have a new way of collecting common sense from people, expanding the size of the project knowledgebase faster than it is nowadays. 3. Finally, people now are able to do searches at the OMCS-Br knowledgebase through a web interface tool, which is online. In this tool, the user will type a concept and the user's profile so that doing a search it is providedthe culturally contextualized information about it, i.e., things that people said about it when they were providing their common sense at the OMCS-Br project website. With this approach, they will have the chance of seeingwhat people think about things, and why their contributions are so important to the project. Besides, they can use the results whenever they need to, without restraints and, more than that, in a very easy way. We are investing in this subject because our studies on supporting children and teenagers education as well as supporting the therapeutic process for family to promote emotional closeness lead us to state that when people see their values, beliefs and preferences expressed in the software, the design, the rules and the content, they consider the application more interesting and easier for them to learn and, more than that, to be engaged in using these software, what is essential to reach the defined goals.Our next steps are to finish the filter and collector implementations and use them more directly in some application to see their behaviors. To the web interface tool, we want to develop some tutorial to help people in using it in an effective way. We want to guide users in knowing how kind of things they can do with the results they get; How to learn something about their future users by doing searches at the OMCS-Br knowledgebase? At the end of the work, we intend to do a study case to see how the tool will work in a real situation by inviting some users to do certain task and, during that, observe how they use the tool, the results they will get and how they will use them.

ACKNOWLEDGMENT

We thank FAPESP, CAPES, Boeing for financial support.

REFERENCES

[1] Sarmento, A, "Issues of Human Computer Interaction", Information Management,pp. 22-23, 2005.

[2] Anacleto, J. C.; Lieberman, H.; Tsutsumi, M.; Neris, V.P.A.; Carvalho, A.F.P.; Espinosa, J., "Can common sense uncover cultural differences in computer applications?", Proc. WCC 2006, pp. 1-10, 2006.

[3] Vance, A.; Elie-dit-cosaque, C.; Straub, D. W. "Examining trust in information technology artifacts: The effects of system quality and culture.", J. of Management Information Systems, pp. 73–100, 2008.

[4] Anacleto, J. C.; Pereira, E. N.; Ferreira, A. M.; Carvalho, A. F. P. De; Fabro, J. A., "Culture Sensitive Educational Games Considering Commonsense Knowledge", Proc. ICEIS 2008.

[5] Silva, M.A.R.; Anacleto, J.C. ; Buzatto, D., "A game to support childrens' expression and socialization considering their cultural background.",IEEE SMC, p1230-1235, 2009.

[6] Anacleto, J. C.; Fels, S.; Villena, J. M. R., "Design of a web-based therapist tool to promote emotional closeness.", In: Proceedings CHI2010, ACM Press, pp. 3565-3570, 2010.

[7] Masiero, A.; Filgueiras, L.; Aquino, P. "Multidirectional Knowledge Extraction Process for Creating Behavioral Personas.", IHC, SBC, 2011.

[8] Minsky, M.; Singh, P.; Sloman, A. "The St. Thomas Common Sense Symposium: Designing Architectures for Human-Level Intelligence", AI Magazine, vol. 25, no. 2, pp. 113–12, 2004.

[9] Singh, P.; Lin, T.; Mueller, E. T.; Lim, G.; "Open Mind Common Sense: Knowledge acquisition from the general public", Proc. 1st Int. Conf. on Ont., Databases, and App of Sem. for Large Scale IS, California, 2002.

A Visual Approach to Validate the Selection Review of Primary Studies in Systematic Reviews: A Replication Study

Katia Romero Felizardo, Ellen Francine Barbosa and José Carlos Maldonado
Comp. Systems Department – University of São Paulo – São Carlos, SP - Brazil
katiarf, francine, jcmaldon@icmc.usp.br

Abstract

One of the activities associated with the systematic literature review (SLR) process is the selection of primary studies. When the researcher faces large volumes of primary studies to be analysed, the process used to select studies can be arduous, specially when the selection review activity is performed and all studies under analysis are read more than once. An experiment was conducted as a pilot test to compare the performance and accuracy of graduate students in conducting the selection review activity manually and using visual text mining (VTM) techniques. This paper describes a replication study that used the same experimental design and materials of the original experiment. The results have confirmed the outcomes of the original experiment, i.e., VTM is promising and can improve the performance of the selection review of primary studies. There is a positive relationship between the use of VTM techniques and the time spent to conduct the selection review activity.

1 Introduction

Systematic Literature Review (SLR) is a "means of identifying, evaluating and interpreting available research relevant to a particular research question, or topic area, or phenomenon of interest" [6].

Controlled experiments, case studies and surveys are examples of primary studies which compound the information source of SLRs. These empirical studies are grouped and summarized by SLRs, composing the secondary studies [5]. Kitchenham [5] proposed a process for SRs in Software Engineering (SE) that involves three phases: (i) planning the review, (ii) conducting the review, and (iii) reporting the review. During the planning phase, the need for a review is identified and the review protocol is developed. The protocol includes items, such as sources selection, search methods and keywords, inclusion, exclusion and quality criteria for primary studies. The activities of the second phase include the identification of relevant research, selection of primary studies based on the inclusion and exclusion criteria, *selection review*, assessment of study quality and data extraction. Finally, the third phase comprehends data synthesis and dissemination or reporting of the SLR's results to interested parties including researchers and practitioners.

According to the literature, a potentially problematic aspect of the SLR process is the primary study selection [12], which is both challenging and time-consuming. The selection of primary studies is usually a three-stage process: (i) initially the selection is based on a review of titles, keywords, and abstracts. The studies are selected against the inclusion/exclusion criteria defined in the protocol and studies that can help to answer the specified research questions are included and irrelevant papers are rejected; (ii) full copies of the papers classified as included in the first stage are obtained and selected against the same set of inclusion/exclusion criteria used previously; (iii) the studies should be reviewed (*selection review* activity) to ensure that relevant studies have not been eliminated.

The *selection review* activity aims to prevent the exclusion of relevant studies and can be conducted in two different ways [6]: (i) performed by two or more reviewers – uncertainties about the inclusion or exclusion should be investigated by sensitivity analysis, which involves repeating the selection activity in the studies divergently classified by reviewers; and (ii) performed by an individual – the researcher should consider discussing their decisions with other researchers or, alternatively, the researcher can re-evaluate a random sample of primary studies to determine the consistency of the decisions.

Consequently, the *selection review* activity implies additional effort to re-read the studies, mainly if more than one reviewer is considered. A highly successful approach to support tasks involving the interpretation

of a large amount of textual data suitable to be applied to the *selection review* activity is known as Visual Text Mining (VTM) [7]. VTM is an extension of Text Mining (TM), a well-established practice commonly used to extract patterns and non-trivial knowledge from unstructured documents or textual documents written in a natural language [11]. VTM is the association of mining algorithms and information visualization techniques that support visualization and interactive data exploration [1].

Felizardo et al. [2] have proposed an approach based on VTM techniques to assist the *selection review* activity in SLR. The techniques proposed by the authors offer, for example, clues about the studies to be doubly reviewed for inclusion or exclusion when an SLR is performed by only one reviewer, replacing the random choice strategy. The authors conducted an experiment to compare the performance and accuracy of PhD students in reviewing the selection of primary studies manually and using the VTM-based approach. One of the potential threats to the internal validity of the original experiment was related to the sample used (four subjects). Therefore, the goal of this paper is to replicate the initial experiment conducted by Felizardo et al. [2], involving a larger sample size of subjects.

The remainder of this paper is organised as follows: Section 2 provides background information on the original experiment. In the sequence, Section 3 brings a detailed view of the replication; subsection 3.1 summarises the results of the replication in isolation and compares them to those of the first execution. Conclusions and future work are discussed in Section 4.

2 Description of the Original Experiment

Replication is an essential component of experimentation. The term replication has come into use to refer to a systematic repetition of an original experiment to double-check its results [3]. This definition implies that a replication must be explicitly related to a previous experiment. The original experiment to assess the utility of VTM techniques in the *selection review* activity was conducted by Felizardo et al. [2] in 2012. The remainder of this section summarizes the original experiment, which involved two research questions: (1) RQ1: Do VTM techniques (content and citation maps) improve the performance (time spent) of the *selection review* activity in the SLR process?; (2) RQ2: Do VTM techniques improve the accuracy (agreement between systematic reviews as to which primary studies they should include) of the *selection review* activity in the SLR process?

The subjects were four PhD students with prior experience in conducting SLRs.

2.1 Materials

- **VTM Techniques**

The VTM techniques used were document-maps (content and citation maps). The process used to create the maps can be found in [2].

A **content map** (see Figure 1) is a 2D visual representation of primary studies that enables users to investigate content and similarity relationships among these studies. Each primary study is mapped to a graphical element represented by a circle. Similar documents, in terms of content (i.e. titles, abstracts and keywords) are placed close to one another and dissimilar documents are positioned far apart.

One of the techniques to review the selection activity is to create a content map containing the studies collected and analyzed in an SLR and highlight them using different colors[1] as a strategy to identify the two possible classes of studies – included or excluded (red points are studies excluded from the review and blue points represent the included ones). A clustering algorithm can be applied to the content map, creating groups of highly related (similar) documents. The resulting clusters are analyzed in terms of included and excluded documents in order to find inconsistencies. In this analysis, the possible situations a cluster can be configured and the possible consequences for the review process are: (a): Pure Clusters – all documents belonging to a cluster have the same classification (all included or excluded). Normally, such cases do not need to be reviewed; (b): Mixed Clusters – there are documents with different classifications in the same cluster. These cases are hints to the reviewer that there are similar documents with different classifications. The studies grouped there should be reviewed following the traditional method; (c): Isolated Points – there are documents that are not similar to others. These cases are also hints to the reviewer, and the isolated study, if classified as included, must be reviewed.

Examples of pure clusters are identified in Figure 1 as *p*. Mixed clusters are identified as *m*. The evaluation of these clusters can be refined with the help of other content-based strategies, detailed in [2].

Another technique to review the selection activity is to use the **citation map** (see Figure 2), which shows the primary studies (central points – circles),

[1] In general, visualization techniques employ color to add extra information to a visual representation. Therefore we suggest the reading of a color printing version of this paper for fully understanding the pictures.

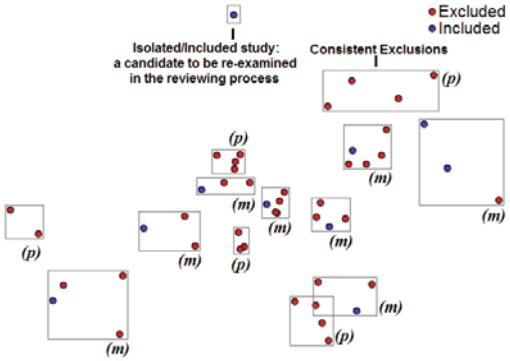


Figure 1. Example of a content map.

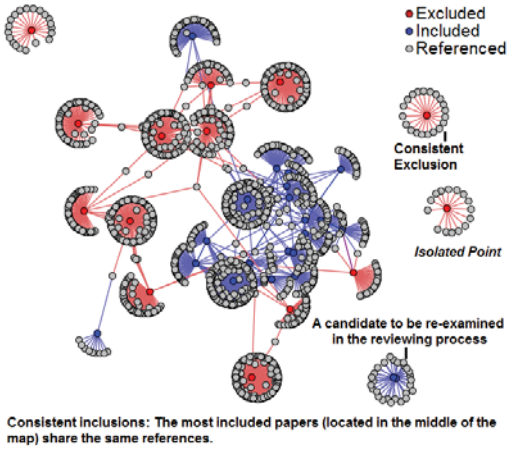


Figure 2. Example of a citation map.

their cited references (grey circles connected by edges) and how documents are related to each other through direct citations or cross-citations. It is possible to identify, for instance, studies that are not connected to any other, that is, studies that do not share citations. These studies, which are isolated in terms of references, deserve attention from experts (reviewers) if they are included in the review. Another situation, which requires attention, arises when a highly connected study, sharing citations with included studies, is not selected for inclusion. In this case, important studies may be missing since co-citation is also a valid criterion. In summary, papers that share references with a relevant paper could be more appropriate for inclusion in the SLR. On the other hand, primary studies that are not connected to any other studies (i.e., they do not share citations or references and are referred to as isolated primary studies) are more likely to be irrelevant documents in terms of a research question and may therefore be more readily excluded from the SLR.

Other VTM strategies of exploration, such as **coordination**, are detailed in [2].

Felizardo et al. [2] implemented a supporting tool, named *Revis – Systematic Literature Review Supported by Visual Analytics*, to enable users to explore a collection of documents (primary studies) using the VTM techniques (i.e., content and citation maps). It takes Revis only a few seconds to create and present content and citation maps with a few hundred documents. Figures 1 and 2 were created using the Revis.

- **Datasets**

The experiment design was organized in two sessions, training and execution. For training purposes, a small set of data (set 1, containing 20 primary studies) and a specific set of inclusion and exclusion criteria were used. To ensure that first impressions from the training would not interfere with the experiment, a different set of data (set 2, containing 41 primary studies) was used for the execution session. The set 2, including articles/papers of periodics and conferences, was originated from an SLR conducted and double-checked by experts in SLRs on the domain of software testing. The purpose of this SLR was to identify testing criteria and testing tools used in the area of concurrent program testing.

2.2 Definition of Users' Task and Metrics

The users' task was to review the studies and either confirm the previous classification – conducted by an expert – or change them, that is, to ensure that the studies marked as included were in accordance with the inclusion criteria and those marked as excluded were in accordance with the exclusion criteria.

Subjects were required to record the time they spent to perform the task, therefore their performance was calculated using the metric: $\frac{chosen\_and\_relevant\_articles}{review\_time}$. The articles marked as included by two or more subjects who participated in the experiment were considered relevant and taken as the oracle. The accuracy was calculated as the number of included studies that belonged to the oracle.

2.3 Experiment Conduction

Subjects were split randomly into two groups, one to conduct the *selection review* activity manually (group 1) and another to use the VTM techniques (group 2). Only the participants involved in the VTM-based task (group 2) were trained on how to use the VTM techniques and the Revis tool. In the execution session,

group 1 was given the list of the papers to be reviewed, based on their reading of the abstracts and the previous classification of the papers (included or excluded). Subjects from group 2 received the visualizations (content and citation maps) containing the same papers used by group 1 (included papers were colored in blue and excluded papers in red). Both groups were given the inclusion and exclusion criteria and a form to summarize their decisions.

2.4 Original Results

The main results are described as follows:

1. the performance of the subjects that used the VTM is higher than that of the subjects using the manual method; and

2. there is no difference in accuracy that used VTM or reading the papers.

3 Replication

This section describes the replication of the original experiment (detailed in Section 2). The subjects involved in this replication were 15 graduate students (10 PhD and 5 master's students) of an SE course at the USP (University of São Paulo), Brazil. They were divided into two groups: (i) one containing 8 subjects (group 1) and another containing 7 subjects (group 2). Each group contained 5 PhD students with prior experience in conducting SLRs.

The same VTM techniques (content and citation maps), datasets 1 (20 studies) and 2 (41 studies), and the set of inclusion and exclusion criteria from the original experiment were used in the replication. The same users' task (*selection review* manually and using VTM) and metrics from the original experiment were used. The design of the original experiment was duplicated for the replication without changes (2 groups, 2 sessions). No time limit was imposed for the experiment and the participants were not allowed to communicate with each other.

3.1 Replication Results

Table 1 shows a summary of the results. The time (see third column) spent by the subjects of group 1 to perform the *selection review* activity on the basis of reading the abstracts varied between 57 and 87 minutes and the time spent by the subjects of group 2 to perform the same activity using the VTM techniques varied between 32 and 56 minutes.

Table 1. Summary of Results.

Group	ID	Time	Performance	Accuracy
Group 1 (Reading)	1	62 min	0.24 articles/min	15
	2	60 min	0.20 articles/min	12
	3	58 min	0.25 articles/min	15
	4	65 min	0.23 articles/min	15
	5	62 min	0.24 articles/min	15
	6	75 min	0.17 articles/min	13
	7	57 min	0.22 articles/min	13
	8	87 min	0.13 articles/min	12
Group 2 (VTM)	9	50 min	0.30 articles/min	15
	10	34 min	0.38 articles/min	13
	11	60 min	0.18 articles/min	11
	12	35 min	0.28 articles/min	10
	13	62 min	0.22 articles/min	14
	14	32 min	0.53 articles/min	17
	15	56 min	0.26 articles/min	15

To answer the first research question (RQ1), the subjects' performances were measured (see fourth column) and the results show that subject 3 reviewed 0.25 articles/min using manual review (group 1), subject 8, also from group 1, reviewed 0.13 articles/min, and subjects 11 and 14 reviewed 0.18 articles/min and 0.53 articles/min respectively, applying VTM. The performance of the subjects that used VTM appeared to be better than that of the subjects that used the manual method. Therefore, the results suggest that the use of VTM can help to improve the performance of the selection activity in the SLR in comparison to a manual reading method.

Eighteen articles were marked as "included" by at least two subjects and considered the oracle. Table 1 (see fifth column) shows the comparison between the VTM and the manual reading approaches in terms of accuracy (RQ2). Regarding the accuracy of researchers, researchers 1,3,4 and 5 – group 1 correctly classified 15 articles of a total of 18 studies included as oracle using manual review, that is, researchers 1,3,4 and 5 correctly classified 83.3% of the articles. Researchers 9 and 15 – group 2 correctly classified 15 articles (83.3%) using VTM techniques. Researcher 14 showed a 94.4% precision (17 articles correctly classified).

Boxplots were used to show the distribution of the performance and accuracy of the subjects in reviewing the primary studies. They are based on non-parametric statistics and can help explain the behaviour of the summary statistics. The bar in the box shows the median (central tendency for the distribution) and the length of the box indicates the spread of the distribution. Figure 3(a) shows that there is no equal variance within the data (the variance of group 2 – using VTM is higher than those of group 1 – reading abstracts) and that the medians for both groups are different. The

highest performance (0.53 articles reviewed/min – an outlier, i.e., a point distant from the rest of the data) was obtained by one subject of group 2. The second highest performance (0.38 articles reviewed/min) was also achieved by one subject of group 2. The lowest performance (0.13 articles reviewed/ min) was obtained by one subject of group 1. Regarding accuracy (see Figure 3(b)), the boxplots show that there is similar variance within the data and that the medians for both groups are the same (14 studies correctly included as oracle).

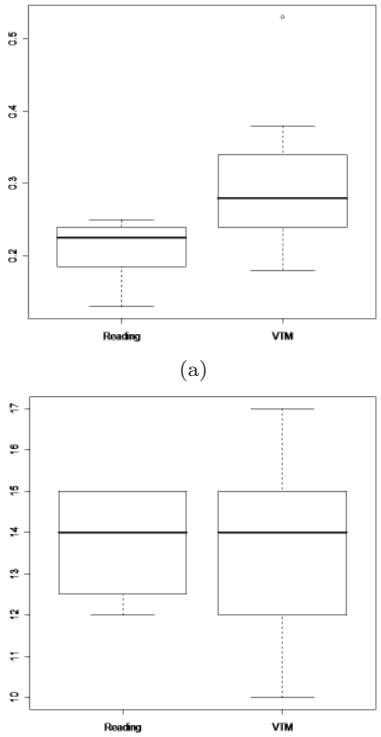


Figure 3. Boxplots showing the distribution of (a) performance and (b) accuracy.

To formally evaluate the results the Man-Whitney test, also called Mann-Whitney-Wilcoxon test, a non-parametric statistical hypothesis test, was used. Regarding performance, our results have shown that (see Table 2 – Performance) there is a statistically significant difference (P-Value = 0.0146 $<$ 0.05) between the time averages with the use of VTM and the traditional method (reading abstracts). Therefore, we can state that the use of VTM can improve the performance of the primary studies review task.

A plausible explanation to the significant difference in the performance using VTM is that VTM techniques usually allow a faster data exploration helping address the challenges that arise in the exploration of large datasets [4]. Moreover, VTM techniques facilitate the extraction of high-quality information from a large amount of primary studies usually through content/similarity and citation relationships. Our aim is not to eliminate the traditional method, i.e., reading the abstracts or full texts, to review the primary studies. Rather, our hypothesis is that exploratory visualization techniques may augment the traditional review approach. The employed visual representations can be used to support the decisions made by reviewers regarding inclusions and exclusions.

Regarding accuracy the results have shown that (see Table 2 – Accuracy) there is no statistically significant difference (P-Value = 0.0487 $>$ 0.05) between the accuracy averages with the use of VTM and the traditional method (reading abstracts). Therefore, we can affirm that the use of VTM exerts no effect on the accuracy of the primary studies review task. The reasons for the no significant difference in the accuracy using VTM are not clear but it may have something to do with different factors. Firstly, the subjects who participated of the replication were partly master students. The level of experience of the subjects in conducting SLRs could affect their capable to review the studies. Only after a few years of experience in a certain research field, researchers are capable to review studies. Secondly, how easy it is to review papers for selection in an SLR depends on the domain and the papers to be examined. In this replication, the subjects were not specialist in concurrent software testing. Thirdly, sometimes it is really hard to decide whether to include a paper or not, independent of whether using VTM or not. Finally, SLRs on the same topic may reach different conclusions [9].

Table 2. Results for Man-Whitney test.

Variable	*P-Value*	Statistically Significant?
Performance	0.0146	**Yes** (*P-Value* $<$ 0.05)
Accuracy	0.0487	No (*P-Value* $>$ 0.05)

We compared the replication results with the outcomes of the original experiment. In both experiments, the results showed that the incorporation of the VTM into the SLR study *selection review* reduced the time spent on this activity and did not increase the accuracy in comparison to a manual reading method.

4 Conclusions and Future Work

The main contribution of this research is the replication of a controlled experiment to compare PhD and

master students' performance and accuracy in reviewing primary studies manually and using VTM techniques. The results show that the answer to RQ1 is "*Yes*" – suggesting that the performance of the subjects that used VTM is higher than that of the subjects that used the manual method. VTM techniques usually allow a faster data exploration, therefore the main advantage of using VTM is the acceleration of the rate at which review of a large volume of primary studies can be undertaken. In other words, the use of VTM techniques speed up the *selection review* activity.

One of the potential threats to the internal validity of our study is related to the fact that typically, many SLRs involve a greater number of studies to be considered during the review stage (more than 100). However, we chose to use in our replication the same SLR used in the original experiment, which contained 41 primary studies. We made this choice on the assumption that adding too many studies to our replication could similarly influence the results, affecting the motivation of the subjects to carry out the assigned tasks.

The Revis and VTM techniques can be used in different domains of SLRs. According to the VTM experts, VTM tools work better with more articles [8], where a large number of candidate studies are considered – hundreds and even thousands. The key disadvantage of introducing VTM in the SLR process is the additional knowledge required, i.e., the subjects will need to become familiar with the visual tools. A current limitation of the Revis tool is that the documents (papers) under analysis need to be in bibtex format in order to be loaded into Revis. Thereby, if the studies are in any other format (e.g. PDF), it is necessary to convert them prior to the analysis. However, many electronic databases (e.g. ACM Digital Library; IEEE Xplore; Web of Science; Scopus and Springer Link) have the facility to export citations from datasets to the bibtex format, hopefully making this process as automated as possible.

The presented results are promising and reveal the benefits of using VTM techniques in supporting the conduction of SLRs. Further replications involving more subjects and a wider dataset will be conducted in order to identify significant effects in accuracy provided by the use of VTM techniques. Conclusive results regarding the use of VTM to support the SLR process should be achieve before widely adopting the use of VTM techniques.

The empirical SE comunity has been addressing several issues related to replication, including the role of lab packages to support replications [10]. The lab package of our experiment is available for replications upon request.

Acknowledgment

The authors would like to acknowledge the Brazilian agency FAPESP(Process n.2012/02524-4) for the financial support provided to this research.

References

[1] M. de Oliveira and H. Levkowitz. From visual data exploration to visual data mining: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 9(3):378–394, 2003.

[2] K. Felizardo, G. Andery, F. Paulovich, R. Minghim, and J. Maldonado. A visual analysis approach to validate the selection review of primary studies in systematic reviews. *Information and Software Technology*, 54(10):1079–1091, 2012.

[3] N. Juristo and S. Vegas. Using differences among replications of software engineering experiments to gain knowledge. In *3rd International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 356–366. IEEE Computer Society, 2009.

[4] D. Keim. Information visualization and visual data mining. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):1–8, 2002.

[5] B. Kitchenham. Procedures for performing systematic reviews. Joint Technical Report TR/SE-0401 (Keele) - 0400011T.1 (NICTA), Software Engineering Group - Department of Computer Science - Keele University and Empirical Software Engineering - National ICT Australia Ltd, 2004.

[6] B. Kitchenham and S. Charters. Guidelines for performing systematic literature reviews in software engineering. Technical Report EBSE 2007-001, Keele University and Durham University, UK, 2007.

[7] A. A. Lopes, R. Pinho, F. V. Paulovich, and R. Minghim. Visual text mining using association rules. *Computers and Graphics*, 31(3):316–326, 2007.

[8] V. Malheiros, E. Hohn, R. Pinho, M. Mendonca, and J. Maldonado. A visual text mining approach for systematic reviews. In *1st International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 245–254. ACM, 2007.

[9] F. Peinemann, N. Mcgauran, S. Sauerland, and S. Lange. Disagreement in primary study selection between systematic reviews on negative pressure wound therapy. *BMC Medical Research Methodology*, 8:16, 2008.

[10] F. Shull, J. Carver, S. Vegas, and J. N. The role of replications in empirical software engineering. *Empirical Software Engineering*, 13(1):211–218, 2008.

[11] P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Addison Wesley, 1 edition, 2005.

[12] H. Zhang and A. Muhammad. Systematic reviews in software engineering: An empirical investigation. *Information and Software Technology*, page In Press, 2012.

Andon for Dentists

Saulius Astromskis, Andrea Janes, Alberto Sillitti, Giancarlo Succi
Centre for Applied Software Engineering
Free University of Bozen/Bolzano, Bolzano, Italy
saulius.astromskis@stud-inf.unibz.it, {andrea.janes, alberto.sillitti, giancarlo.succi}@unibz.it

Abstract

The Lean promise, to help organizations to improve their efficiency can also be applied to healthcare services.

In this article we describe an application of the "Andon" concept for odontoiatric software. This concept is applied visualizing the treatment status in each room of a dentist using pre-attentive processing.

As a result, a dentist or assistant is able to grasp immediately what is going on within the team, which rooms are free, and if special rooms with particular equipment (such as the x-ray machine) is available.

The here described considerations can also be applied in other types of applications.

1. Introduction

One reason for Toyota's success—Toyota is the world's largest auto maker [1]—is the so called "Toyota Production System", a particular way to manufacture cars. The ideas behind this particular way became known as "Lean thinking" with the same-titled book by Womack and Jones in 1996 [2].

Lean thinking aims to maximize efficiency, i.e., to obtain the desired outcome minimizing the required resources. One method to achieve this is to provide everyone involved with a full picture of what's going on at any time. Lean Thinking facilitates the coordination among the various stakeholders eliminating the need to "transport" information from one to another [3].

The concept to ensure that everybody knows what is going on is "Andon" [3, 4].

In Japanese, the word "Andon" means "paper lantern". In manufacturing it describes a signboard that—hung above the production line—indicates the current status of the line.

"If everything is normal, the green light is on. When a worker wants to adjust something on the line and calls for help, he turns on a yellow light. If a line stop is needed to rectify a problem, the red light is turned on. To thoroughly eliminate abnormalities, workers should not be afraid to stop the line [4]."

The indicator is depicted in figure 1: the car at the work station 1 is fine, the car at the work station 2 has a missing wheel. As soon as the worker notices a problem, he pulls the Andon cord to stop the assembly line and to inform other works to come to help him to fix the problem.

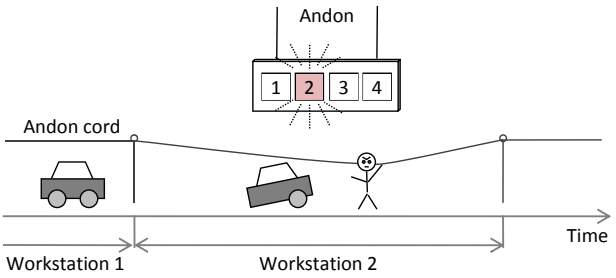


Figure 1. In the Toyota Production System, an Andon visualizes the status of some production step.

Knowledge management aims to support the deployment of knowledge across the organization [5]. Many organizations have systems in place that contain a lot of information about the ongoing business processes, the employed resources, and the produced output (see e.g., [6]). Unfortunately the already available data is not used to inform stakeholders about the current status or to guide decisions.

In this work, we present how we how we applied the Andon concept within a practice management application for dentists. This system was developed for an orthodontic practice in Italy, based on the requirements of the dentists and the experiences made after its introduction.

We aim to inspire others to follow our line of re-

search in trying to build knowledge dissemination into their software and in this way exploit the data already present within organizations.

2. Approach

Andons or information displays can be designed in a variety of ways. We distinguish two scenarios to provide information through a display: "pull" and "push". In the "pull" scenario the user wants to get a specific piece of information and uses the software to obtain it [7]. In such case aspects of technology acceptance become important, such as the perceived usefulness and the perceived ease of use [8]. When designing an information display, its important to consider [9]:

1. The dashboard should help the user to understand the context of the data, i.e., state why was this data collected, how should it be interpreted, how can we use this data in future projects [10], etc.

2. The dashboard should help the user to understand the meaning of the data, i.e., choose visualizations that require a minimum of effort to get the conveyed message, be coherent in the chosen visualization strategies, allow the user to choose the level of detail of the data, etc.

In the "push" scenario the display has to be designed so that important information is "pushed" to the user, i.e., captures his or her attention and informs the user.

Whether a display is more suited for the "push" or the "pull" scenario depends on how much effort a user has to invest to see the desired data. A display that "pushes" the information to the user has the advantage that it can inform the user in unexpected, unforeseen situations about problems, anomalies, etc. A display that was designed to support the "pull" scenario should offer more possibilities to explore the data, to filter and to search, to investigate the reasons that caused the data, etc [11].

To setup a display that is used in a "push" scenario, we found the following considerations important [9, 12, 11]:

1. The user should see the display without any effort. For example, in the car, the display is built in such a way that it is in the range of vision of the driver. An organizational display should be displayed on a monitor in the corridor or the office where many are passing by. The information will be pushed to the users without their active participation. An example for such a display is the Andon board, used in lean manufacturing, and placed visibly so that everybody sees if there is a problem on the assembly line (see fig. 1).

2. The user should not need to interact with visualizations to understand the data. The charts have to be designed in such a way that an interaction is only necessary when the user switches into the pull mode, i.e., the display got the attention of the user and he or she wants to investigate further.

3. Arrange the data to minimize the time needed to consult the display. Place the same information always on the same spot. Allow the user to develop habits, e.g., every morning, when passing by with the coffee in the hand, she can check the current size of the error-log that is displayed in the upper right corner of the display.

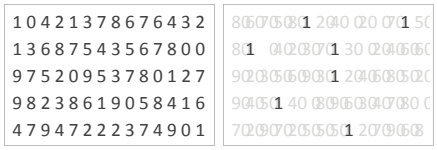
4. Guide the attention of the user to indicate important information. There are different mechanisms that draw the attention of the user. If they are overused, the user neglects them. For example, if everything on the display is blinking, the user will ignore it.

5. Since we want that the users look at the displays by choice, also aesthetical considerations can increase the interest for the user to look at the display.

To guide the design of the user interface of our electronic waiting list, we used a technique called "preattentive processing", which we explain now.

Researchers have identified different graphical properties that are processed pre-attentively and grouped them into form, color, motion, and spatial position [12]. Pre-attentive processing elements have the advantage that they are processed (i.e., understood) faster than not pre-attentive elements [12].

An example is provided in figure 2a. If we try to count the number of 3s, we have to process the numbers sequentially, i.e., we have to look at each number separately and decide if it resembles the form of the number 3.



(a) (b)

Figure 2. How many 3s do you see?

It is much easier to count the number of 3s in figure 2b. The numbers get noticed much faster because

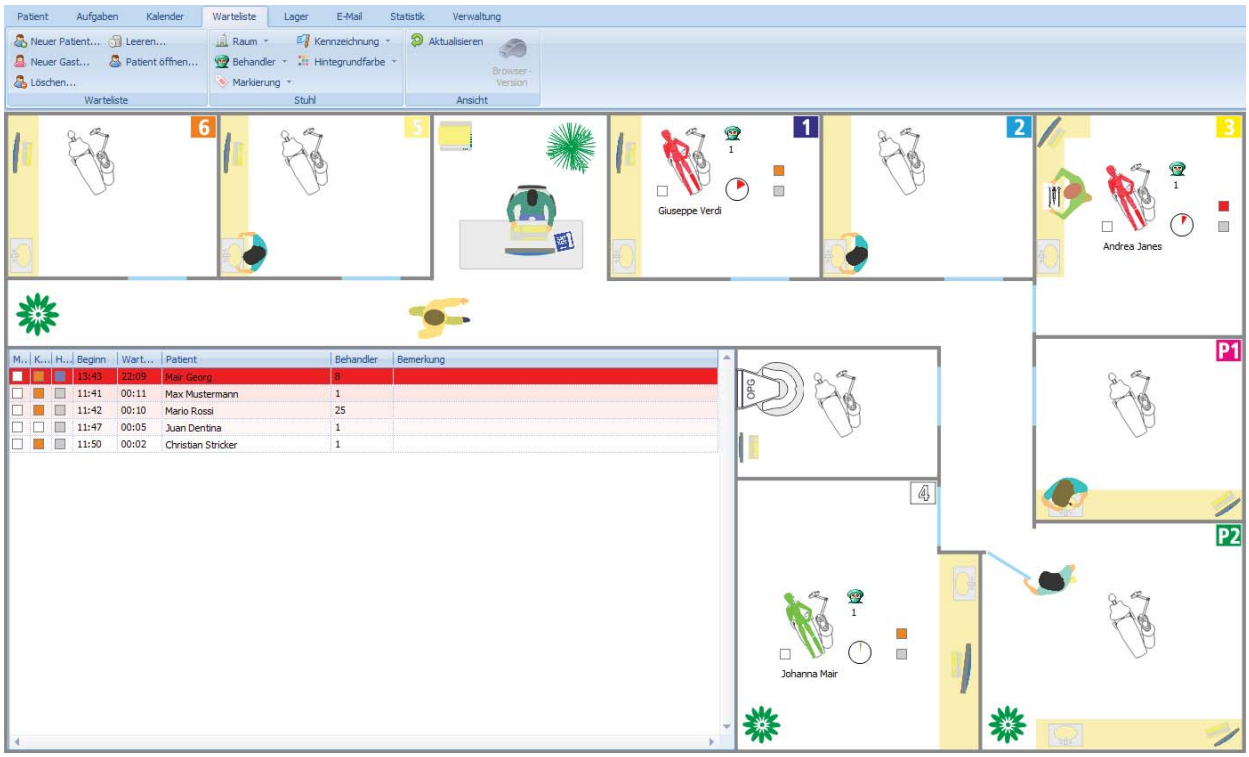


Figure 3. Electronic waiting list of a dental practice with 8 treatment rooms.

of their different color. Moreover, we are able to "filter out" only the 3s from the remaining numbers. We cannot do this if all numbers have the same color.

How strong something is noticed pre-attentively depends from how different the highlighted element is from the others and how different the other elements are among each other.

3. Proof of concept

In this section we present how we developed an Andon board for a software management system for dentists. This Andon board is not intended to be seen by patients, but only doctors and assistants. We implemented this system for two dentists, one with 9 treatment rooms, one with 14. The design of the display is shown in figure 3, we will explain the single parts of it below.

The here presented proof of concept is not just an example, it is the description of an actual implementation that is in use by two dental practices. The figures 3–5 are screenshots of the actual system we implemented.

Typically, dentists organize the visits of their patients in a calendar, and when patients arrive they have to wait in the waiting room.

We implemented an electronic waiting list that allows to create, remove, update, or delete waiting list entries containing the following fields: the treatment priority, two classifications of the upcoming treatment, the time the patient arrived, the name of the patient, the responsible practitioner, and the current room in which the patient is.

If the patient was already registered as patient before, the responsible practitioner is selected automatically, using the favourite practitioner stored in the patient's record.

To use the electronic waiting list as a coordination instrument, i.e., to inform practioners and assistants who is currently waiting, which rooms are taken, etc., we identified the following requirements:

1. Patients do not like to wait [13]. The electronic waiting list must indicate how much the patients are already waiting so that those patients can be treated as soon as possible.

2. Not all patients have the same priority. The electronic waiting list must indicate if some patient has acute pain so that those patients can be treated as soon as possible.

3. Some patients have special conditions that must be taken in to consideration (e.g., a patient might

Figure 4. Patients that just arrived are added to the virtual waiting room

have an allergy or some particular disease). To ensure the safety of the practitioners and the patient, the electronic waiting list must indicate this.

4. The waiting list will be used as a coordination instrument, i.e., it will be visible on the screens of the dental practice and seen not only by the doctors, but inevitably also by the patients. To ensure the privacy of all patients, sensitive information about the patient must be visualized in an encoded way. This avoids that a patient treated in room A sees for example, that the patient in room B—which he or she might know—has some disease.

5. Dentists can treat more than one patient at a time since preparatory or concluding steps can be handled by the assistants. The electronic waiting list has to indicate which room is free so that a new patient can be assigned quickly to an empty room.

We decided to use two pre-attentively processed properties to design the electronic waiting list: color for the requirements 1–4 and spacial position for requirement 5.

We continue now explaining how we designed the user interface of the electronic waiting list.

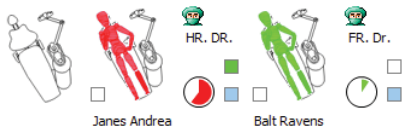
When patients arrive, they are added to the waiting list. Patients wainting in the waiting room are shown as a table (see figure 4). The table contains the color coded treatment priority, the classifications, the time of arrival, the total waiting time, and the name of the patient.

The longer a patient has to wait, the more the line of the patient is colored in red, representing the anger that the patient is accumulating.

This table has three columns in which particular information about the patient is visualized as colored boxes (e.g., the treatment type or treatment priority).

As soon as an assistant has time, he or she will go to the waiting room and pick up a patient and bring him or her to a free room. The assistant can find a free room just looking at the waiting list application in which free rooms are represented as in figure 5a.

If a doctor wants to use the room with the panoramic x-ray machine (shown as OPG in figure 3), he or she can also just look if it is free now and only then decide to ask the patient to accompany him or her to perform an x-ray scan.



(a) (b) (c)

Figure 5. Visualization of free and busy rooms

The assistant accompanies the patient to a treatment room and prepares everything for the treatment. He or she will open the patient's record on the computer and prepare all instruments and drugs.

By opening the patient's record on the computer in the new room, the record in the electronic waiting list and the visualized is automatically updated. The room is now shown as busy as shown as in figure 5b. Moreover, a busy room shows: a colored box indicating the treatment priority, two colored boxes indicating the treatment type 1 and 2, the name of the patient,an acronym of the responsible doctor, and a pie chart indicating how much time the patient is already in the room.

The patient and the pie chart are shown in red, if the doctor did not start the treatment yet. Again, the red pie chart shows the waiting time of the patient. As soon as the treatment begins, the assistant presses the "F12" key on the keyboard of the computer in the treatment room. This changes the status in the electronic waiting list and the patient is shown as green (as in figure 5c). To others this indicates in which room a doctor is currently working and that assistants should enter in a busy room only if this is really necessary to not disturb the patient.

As soon as the patient's record is closed, the record in the waiting list is deleted and the room is shown as free. The time required to sterilize and disinfect the instruments and to prepare them for the next patient is not handled by the waiting list since this can occur also while a new patient is in the room.

Particularly in large practices, the visualization of the free/busy as well as treatment status helps that everybody in the practice obtains a clear picture of what is currently going on, which rooms are busy, and where the doctors are.

3.1. Tracking of the waiting duration

Using an electronic waiting list has the advantage that it is possible to track and analyze waiting times. In the practice depicted in figure 3, we tracked for one year how much patients where waiting. In particular, we were logging time time between when the patient was added to the waiting list to the time the assistant pressed "F12" to indicate that the treatment began.

From the 4th of January 2012 to the 4th of January 2013 we recorded 14,704 visits, with the waiting time depicted in figure 6. To understand where the most data points are, we used a visualization called hexagon binning [14], a form of bivariate histogram useful for visualizing the structure in datasets with large n. The concept of hexagon binning we used is:

1. we tesselate the xy plane over the set (range(x), range(y)) by a regular grid of hexagons,
2. we count the number of points falling in each hexagon,
3. we plot the hexagons using a color gradient in proportion to the counts.

Figure 6 shows that the majority of patients waits under 10 minutes, still it is interesting to evaluate why single patients had to wait more than 20 minutes.

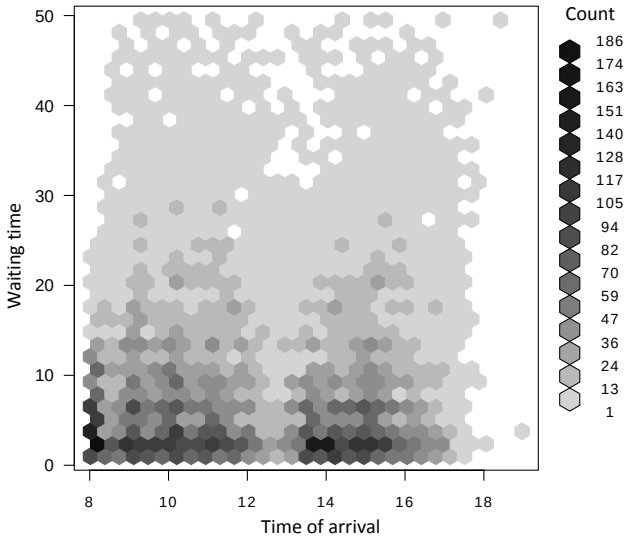


Figure 6. Visualization of waiting time in relation to the time of arrival using hexagon binning.

Moreover, figure 6 shows that during lunch time (12pm-1pm) and in the late afternoon (after 4pm) less patients visit the practice.

Another possible analysis is to study the waiting times in relation to the room to which the patient was assigned to after waiting. Some rooms have special equipment that has to be prepared, other rooms are generic, the interpretation of an analysis as in figure 7 depends on the specific practice.

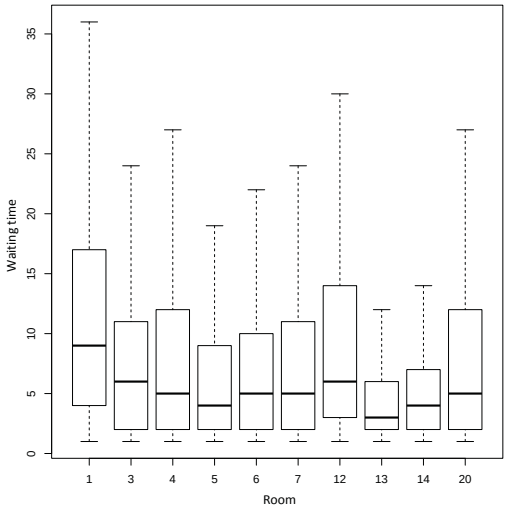


Figure 7. Visualization of waiting time in relation to the room they were assigned to using box plots (excluding outliers).

This case we see that, if the practice wants to improve the waiting times, the preparations in room number 1 should be improved.

4. Conclusion and future work

This paper presented how we applied the Andon concept for two dentists. The visualization of the current status of the practice is not only useful to understand what is going on, but the logged data can be used later to study how to improve.

We implemented this Andon concept only for two dentists, therefore we can consider this implementation just an example of an approach that revealed as useful in two cases.

In both dental practices the dentists found the Andon board so useful to organize the assignment of patients to rooms that they required us to change the software so that the initial screen is not anymore the patient search, but the Andon board.

The Andon board keeps track of how long patients are waiting, which room is free, who is waiting too long. As a by-product the logging of the the visualized data can be used to study the root causes of waiting times in the practice or other upcoming problems [15].

As next steps we plan to extend the Andon concept to the patients. We will study on how to analyze the collected data to understand which types of activities create delays, how patients move within the practice [16], how we can inform patients on-the-fly about delays, for example using an SMS message, and how we can inform patients about their treatment using innovative types of visualizations (e.g., [17]) or analyses (e.g., [18]).

5. Acknowledgements

We want to thank Lorenz and Ute Moser from the Orthodontic Practice Moser and Christian Stricker from the software house CompuNet for their feedback on the developed Andon board.

References

[1] C. Dawson, "Toyota Takes Sales Crown," *The Wall Street Journal (U.S. edition)*, p. B1, January 29 2013.

[2] J. Womack and D. Jones, *Lean thinking: banish waste and create wealth in your corporation*. Simon & Schuster, 1996.

[3] J. B. Austenfeld, "Toyota and why it is so successful," *Papers of the Research Society of Commerce and Economics*, vol. 47, no. 1, pp. 109–173, sep 2006.

[4] T. Ōno, *Toyota Production System: Beyond Large-Scale Production*. Productivity Press, 1988.

[5] I. Rus and M. Lindvall, "Knowledge management in software engineering," *Software, IEEE*, vol. 19, no. 3, pp. 26 –38, May-June 2002.

[6] M. Scotto, A. Sillitti, G. Succi, and T. Vernazza, "A non-invasive approach to product metrics collection," *J. Syst. Archit.*, vol. 52, no. 11, pp. 668–675, Nov. 2006.

[7] A. Janes and G. Succi, "To pull or not to pull," in *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*, ser. OOPSLA '09. New York, NY, USA: ACM, 2009, pp. 889–894.

[8] V. Venkatesh and H. Bala, "Technology acceptance model 3 and a research agenda on interventions," *Decision Sciences*, vol. 39, no. 2, pp. 273–315, 2008.

[9] S. Few, *Information Dashboard Design: The Effective Visual Communication of Data*, ser. Oreilly Series. O'Reilly Media, Incorporated, 2006.

[10] A. A. Janes and G. Succi, "The dark side of agile software development," in *Proceedings of the ACM international symposium on New ideas, new paradigms, and reflections on programming and software*, ser. Onward! '12. New York, NY, USA: ACM, 2012, pp. 215–228.

[11] A. Janes, A. Sillitti, and G. Succi, "Effective dashboard design," *Cutter IT Journal*, January 2013.

[12] C. Ware, *Information Visualization: Perception for Design*, ser. Interactive Technologies. Elsevier Science, 2012.

[13] R. Anderson, F. Camacho, and R. Balkrishnan, "Willing to wait?: the influence of patient wait time on satisfaction with primary care." *BMC Health Serv Research*, vol. 7, 2007.

[14] N. Lewin-Koh, *Hexagon Binning: an Overview*, Apr. 2011. [Online]. Available: http://cran.r-project.org/web/packages/hexbin/vignettes/hexagon_binning.pdf

[15] I. Fronza, A. Sillitti, G. Succi, M. Terho, and J. Vlasenko, "Failure prediction based on log files using random indexing and support vector machines," *Journal of Systems and Software*, vol. 86, no. 1, pp. 2 – 11, 2013.

[16] L. Corral, A. Sillitti, G. Succi, J. Strumpflohner, and J. Vlasenko, "Droidsense: a mobile tool to analyze software development processes by measuring team proximity," in *Proceedings of the 50th international conference on Objects, Models, Components, Patterns*. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 17–33.

[17] I. Fronza, A. Janes, A. Sillitti, G. Succi, and S. Trebeschi, "Cooperation wordle using preattentive processing techniques," in *Proceedings of the 6th International Workshop on Cooperative and Human Aspects of Software Engineering*, 2013.

[18] A. Mahdiraji, B. Rossi, A. Sillitti, and G. Succi, "Knowledge extraction from events flows," in *Methodologies and Technologies for Networked Enterprises*, ser. Lecture Notes in Computer Science, G. Anastasi, E. Bellini, E. Nitto, C. Ghezzi, L. Tanca, and E. Zimeo, Eds. Springer Berlin Heidelberg, 2012, vol. 7200, pp. 221–236.

Identifying Extract Method Opportunities Based on Variable References

Mehmet Kaya
Department of computer Science and Electrical Engineering
Syracuse University
Syracuse, NY, USA
mkaya@syr.edu

James W. Fawcett
Department of computer Science and Electrical Engineering
Syracuse University
Syracuse, NY, USA
jfawcett@twcny.rr.com

***Abstract*— Long methods are usually difficult to read and comprehend due to the length and complexity of the code. As a result, maintenance can be time consuming and costly. One strategy to lower overall cost of software development for large systems is to produce smaller and less complex methods through method refactoring. This paper presents a new technique to automate the selection process of program fragments for refactoring. The soundness of this technique has been demonstrated through experiments on several different software systems. Long method defects can effectively be resolved by extracting code fragments identified with the support of a tool we have developed.**

Keywords-Extract Method Refactoring; Long Method defects; Placement Tree

I. INTRODUCTION

After the delivery or release of a software product, software development enters what is known as the maintenance phase where the focus is "to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment." [9]

For large software systems, effective maintenance is difficult without readable, well-structured, and relatively simple code. If code fails to meet these properties, refactoring is one strategy to improve the internal structure of the code and thus its maintainability. Refactoring is an umbrella term to describe any process that enhances the program's internal structure without changing its external behavior.

Extract Method refactoring resolves the Long Method defect by decomposing long methods into smaller, more meaningful or cohesive ones. For many cases, problems in large software systems can be attributed to large methods [1]. On the other hand, smaller methods are easier to read, comprehend, and maintain. The decomposition process simplifies a program by constructing more meaningful methods. Although shorter methods/functions improve readability, the key to reducing complexity is to identify and extract, as a unit, related code fragments within a function/method.

Methods after refactoring are effective, since they extend the lifetime of programs by making the reader better able to understand the purpose of each method [1] and more effectively test, as well. Extract Method refactoring consists of two major activities: identification of code fragments to be extracted followed by the extraction of the identified code as a new method and replacement of the original fragment with a function call.

The extraction itself is easily achieved using built in functionality within an IDE such as Visual Studio or Eclipse. However, the selection process is less straightforward. There is not a single identification strategy and various heuristics or guidelines for selection of code fragments have been proposed in literature [4,5,6,7]. Implementation of a selection process can prove challenging and the addition of visualization to these techniques is crucial for effective execution.

Most programming languages, such as C++, C#, C and Java, obey a set of inclusion rules that govern how key constructs are placed within source code. In this paper, the constructs that we will focus on are methods/functions and controls. We will see that placement structure of controls within a method is a tree which we will refer to as a placement tree where each element has only one parent. The approach proposed in this paper constructs a placement tree that contains variable reference counts for individual controls or scopes within a given method. The placement tree is then visualized so that each scope node has a color associated with its dominant or most referenced variable. The goal is to create methods/functions with a single color placement tree. Therefore, a node that does not match the color of its parent node is identified as a candidate code to be extracted. In our earlier paper [3], variable references and dependencies were used for extraction of classes out of an existing class. This research showed that reference counts of variables and scope definitions alone can find effective extract method refactoring candidates.

In this paper, Section 2 covers related works, Section 3 explains the placement tree structure, which is key to this approach. In section 4, the concept of dominant variables is explained and the identification process of candidate code fragments is given in Section 5 along with an explanation of how we visualize methods to identify candidate code fragments. The focus of Section 6 is on the extraction process of identified code fragments. Experiments and results are discussed in section 7 and finally concluding remarks about our paper is presented in Section 8.

II. Related Work

Program slicing has been used extensively to identify code fragments for extract method refactoring [4]. However, implementation can be tedious and time-consuming since this technique usually requires the user to manually select the slicing criterion. Furthermore, there is no a priori approach for choosing the slicing criterion for extract method refactoring. In this paper, we introduce a fully automated selection technique and placement tree visualization tool for detecting candidate extraction fragments in large scale systems.

In [4], a program slicing technique is proposed to extract code fragments related to the computation of a given variable and the state of the given object. A mechanism that uses block-based program slicing to automatically extract methods in object-oriented programs is proposed in [6]. A transformation technique to decompose functions into smaller ones, Tuck, is proposed in [7] based on program slicing.

Control Flow Graphs (CFG) are also used widely as a method extraction tool. In [8], an automatic process for extracting methods based on an input CFG of a function and a set of pre-selected nodes is introduced. To find extract method refactoring candidates, an approach based on Data and Structure Dependency (DSD) graph and longest edge removal algorithm is proposed in [5].

```
1  void closeIfsExpectingelse(){
2      if(TrackScopes->size()> 0 && IfsExpectingelse->size()<= 0){
3          closeAll();
4          return ;
5      }
6      while(TrackScopes->size()> 0 && IfsExpectingelse->size()> 0){
7          std::pair < std::string,char > temp = TrackScopes->pop();
8          if(temp.second == 'o'){
9              TrackScopes->push(temp);
10             break ;
11         }
12         CloseBrace();
13         while(TrackScopes->size()> 0){
14             TrackScopes->pop();
15             CloseBrace();
16         }
17     }
18 }
```

Figure 1 Example Source Code

There are some studies that target only the second step in the subject refactoring method: extracting predefined code fragments from original method. Given an arbitrary set of pre-selected statements, in [8], an algorithm is introduced to extract them preserving the semantics of the original code. In [2], a methodology is given to extract a set of marked statements that are difficult to extract due to the presence of some certain key words.

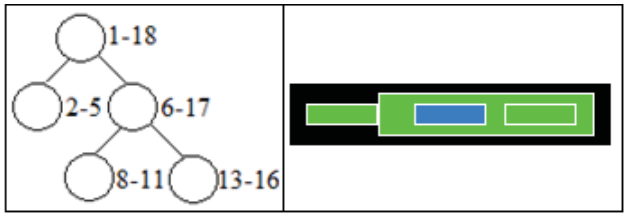


Figure 2 Placement Tree and *Treemap* Representation

The technique proposed in this paper aims to identify scopes that carry out distinct operations. As shown in this paper, density of variable references in code fragments or scopes can be a good indicator for identification of main tasks in large methods with lots of blocks. An effective refactoring method should be supported by a visualization tool or technique for the developers to observe the suggested refactoring better in large scale systems. To visualize the structure or placement tree of the input method, a *treemap* approach is adopted in this research.

Treemap visualization is an effective way of presenting hierarchical information where nodes are represented by nested boxes [10]. T*reemap* visualization in this research represents nodes on the placement tree using boxes where a parent includes all its children. The code fragments chosen for method extraction are decided based on the colors of these boxes which are appointed according to the dominant variables in scopes.

III. Placement Tree

In the proposed technique, an input method is represented with a placement tree where each node represents a different scope. Scope is defined by all code enclosed within braces, e.g., "{" and "}". A scope may include multiple other scopes and this hierarchical placement of scopes constitutes the placement tree.

In this approach, placement tree nodes are classified using eight different scope types. In Table 1, these scope types are shown with their statement coverage. The method scope itself constitutes the root node on the placement tree. If-else blocks are treated as single scopes, since the if part and else part cannot be split into two different methods.

In Figure 1, an explanatory method is shown. Figure 2 shows placement tree for the given example code where each node is identified by its start and end line number. Final visual representation of the placement tree as a *treemap* is also shown in Figure 2.

TABLE I. Scope Types

Type name	Start Line	End Line
Function	Line that includes function name	Line that includes corresponding closing braces
For	Line that includes the key word for this scope. The key word is basically the type name itself, e.g. "for, while, do, switch, if"	Line that includes corresponding closing brace to the one following the respective type name.
While		
Do-While		
Switch		
If		
If-else	Line that includes the key word "if"	Line that includes corresponding closing brace to the one following "else" keyword.
Anonymos	Line that includes open brace that does not have an attached type	Line that includes the closing brace corresponding the anonymous scope opening brace

IV. DOMINANT VARIABLES

After constructing the placement tree for the input method, this approach identifies the dominant variable in each scope. Variable with the highest reference count is identified as the dominant variable for the given scope.

Set theory will be used to better explain how we construct scopes and determine their corresponding dominant variables. *V(F)* is the set of all variable names that appear anywhere in the function. Long Method defects can be detected in Object Oriented Programs, Procedural Programs or Legacy Code using the technique proposed in this paper. All global variables and data members of the class that method under analysis belongs to, may be elements of *V(F)*. Local variables declared in the method are also in the set of *V(F)*. In other words, all variables accessible from within the given method are possible elements of the set *V(F)*, subject to whether they are referenced in the method at least once.

$$V(F)=\{v1,v2,..,vn\} \quad (1)$$

We also have defined several functions to express various properties of program statements, variables, scopes and the method. We will use these functions in our explanations throughout the paper. Table 2 shows the functions and their respective input and output values.

TABLE II. DEFINED PROPERTIES

Name	Input	Output	Use
LN	A program statement, S	Line # of S	LN(S)
RC	A variable name, v; and a scope, B	# of references of v in B	RC(v,B)
SLN	A scope, B	Starting line # of B	SLN(B)
ELN	A scope, B	Ending line # of B	ELN(B)

Let *F* represent the set of all scopes in the given method. And every scope in the given method is represented with a set of statements, *B*. Let the set *V(S)* represent the set of variables that are used for execution of statement *S* and let the set *V(B)* represent the set of all variables that appear in scope *B* at least once.

$$V(S) = \{v | v \text{ is used in } S\} \quad (2)$$
$$B = \{S \mid SLN(B) \le LN(S) \le ELN(B)\} \quad (3)$$
$$\forall S \in B: \nexists B': SLN(B) \le SLN(B') \le LN(S) \le ELN(B') \quad (4)$$
$$V(B) = \{v' \mid \exists S \in B: v' \in V(S)\} \quad (5)$$
$$F = \{B \mid \forall S \in B, B' \in F, B \ne B': S \notin B'\} \quad (6)$$
$$\forall S, B: V(S) \subseteq V(F) \text{ and } V(B) \subseteq V(F) \quad (7)$$

From formulas given above, one can conclude that every statement belongs to only one scope or node in the placement tree. A statement *S'* is and can only be element of one scope *B* that encloses statement *S'*; but *S'* is not and cannot be considered as an element of an ancestor node of *B*, *AB*, at the same time.

After defining our scopes and elements of them, now we find the variable that dominates the computations for every scope. We determine the dominant variable based on their respective reference counts or their respective number of appearances in the subject scope. Let *D(B)* represent the dominant variables in scope *B*,

$$D(B) \subseteq V(B) \text{ such that}$$
$$\forall v \in V(B), v' \in D(B), v'' \in D(B), v \ne v' :$$
$$RC(v', B) > RC(v, B) \text{ and } RC(v', B) = RC(v'', B)$$

The set *D(B)* therefore includes only those variables whose reference counts are highest in block *B*. Every dominant variable name in the whole method is assigned a unique color to represent its power on the scope that it dominates. In our visualization tool, these colors are used to distinguish the nodes that should be extracted.

When the number of dominant variables is zero for the scope *B*, that is |D(B)|=0, in the placement tree, that node is represented with the color of black. On the other hand, when the number of dominant variables is greater than one for the scope *B*, that is |D(B)|>0, two approaches are proposed: Parent Protection, Sibling Collaboration. The idea behind these approaches is to keep the blocks that are dominated by the same variable together as much as we can.

A. Parent Protection

When scope *B* is dominated by more than one variable, according to Parent Protection approach, dominant variable of B's parent node, *BP*, is checked. If dominant variable of the parent node is an element of *D(B)*, then parent node, *BP*, protects its child node *B*, and this dominant variable is assigned to be the dominant variable of node *B*. Otherwise, one of the dominant variable from *D(B)* is randomly selected to be the dominant variable of this scope. Let *dB* and *dBP* represent the dominant variables of the nodes *B* and *BP* respectively.

$$\begin{array}{ll} if\ dBP \in D(B) & dB = dBP, \\ if\ dBP \notin D(B) & dB = v, v \in D(B) \end{array}$$

B. Sibling Collaboration

When scope *B* is dominated by more than one variable, according to Sibling Collaboration approach, dominant variables of its sibling nodes, *SB* and *SA*, are checked. *SB* and *SA* are those nodes that come right before, and after scope *B* in source code (left and right nodes in placement tree respectively). Dominant variable of the sibling *SB* is evaluated first. If dominant variable of the sibling node, *SB*, is an element of *D(B)*, then sibling nodes, *B* and *SB*, collaborate. In this case, dominant variable of *SB* is assigned to be dominant variable of node *B*.

If dominant variable of the sibling node, *SB*, is not an element of *D(B)*, dominant variable(s) of the other sibling node, *SA*, is evaluated. If *SA* has only one dominant variable, that is |D(SA)|=1, and this dominant variable of *SA* is an element of D(B), then sibling nodes, *BP* and *SA* have collaborated, and this dominant variable is assigned to be the dominant variable of node *B* as well.

If *SA* is dominated by more than one variable, that is |D(SA)|>1, a random variable from D(B) ∩ D(SA) is chosen and assigned to be the dominant variable of both *B* and *SA*. If |D(B) ∩ D(SA)|=0, a random variable from *D(B)* is chosen and assigned to be the dominant variable of node *B*. Let *dB, dSB*

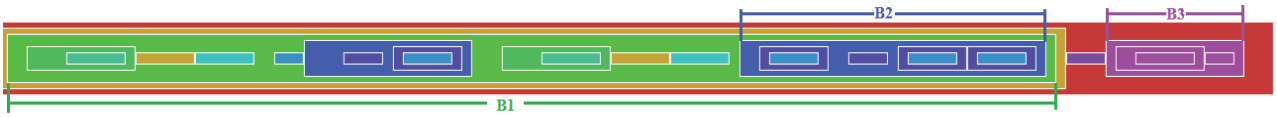


Figure 3 Refactoring Suggestions

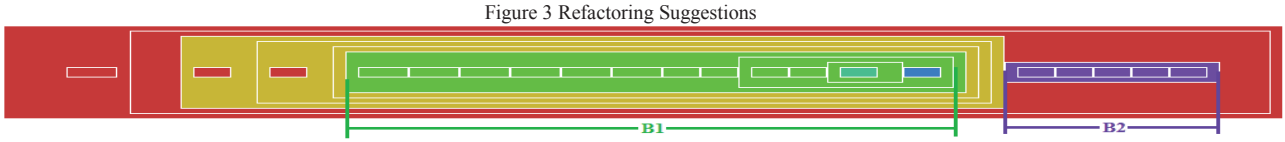


Figure 4 Refactoring Suggestion

and *dSA* represent the dominant variables of the nodes *B*, *SB* and *SA* respectively.

$$if\ dSB\ \in\ D(B)\ \ then\ \ \ dB = dSB,$$

$$if\ dSB\ \notin\ D(B) and\ |D(B)\cap\ D(SA)| > 0$$

$$then\ dB = dSA = v, v \in D(B)\cap\ D(SA)$$

$$if\ dSB\ \notin\ D(B) and\ |D(B)\cap\ D(SA)| = 0$$

$$then\ dB = v, v \in D(B)$$

Adopting one of these two approaches, only one dominant variable is assigned to each node in our placement tree. This variable will be the one that involves in the computation of that scope most. During the resolution of multiple dominant variables, we start from the outmost scope in parent protection approach and leftmost scope in sibling collaboration approach. This guarantees that when a scope is analyzed for dominant variable, its parent node or left sibling node has already been assigned its dominant variable with respect to the adopted approach. User intervention can be used instead of random selection during this process, and this remains as a future work.

V. Identifying Candidate Code Fragments

There are two types of scopes that we suggest to be extracted from original code as new methods.

1. Large code fragments with a color different from parent's color. In Figure 3, we show an example for this case.
2. Consecutive sibling nodes with the same color. In Figure 4, we show an example for this case

In this paper, refactoring suggestions aim to generate methods with minimum number of color diversity. We suggest to extract first the out most scopes with a color different from their parent's color. After refactoring, resulting code should be analyzed again for further refactoring until possibly all generated methods and the original method have only one dominant variable for every scope. Therefore; the resulting code will yield methods that handle only one smaller and less complex task. Figure 3 and Figure 4 show some of the possible scopes that suit our refactoring suggestions.

VI. Extracting Code Fragments

After determining the code fragments for refactoring using our tools, developers are now to extract those fragments as new methods and replace the fragments with function calls to the new methods. Once the code fragments for refactoring are identified, extracting them is usually trivial except a few points that require attention. This section explains how the extracting process should be carried out considering some important cases that, without careful handling, might cause compilation errors or alteration in the behavior of the system.

A. Parameters of Extracted Methods

Analysis and visualization tools described in this paper have been tested on several methods with different sizes. And extract method refactoring is applied on identified fragments. Main motivation of this work is to effectively detect code fragments for extraction as new methods. Number of arguments, that needs to be passed to the extracted methods, have not been considered.

C++ is one of the most difficult programming languages for static code analysis because of its complex syntax and semantic. For this reason, for our experiments, methods that are written in C++ programming language are chosen. In C++ language, there are various ways of passing arguments to methods. By default, arguments to methods are passed by value. When an argument is passed to a method by value, changes that the method does on this argument never affect the value of the argument in the calling method. C++ also provides an option to pass arguments by reference. A reference to a variable is simply an alias for that variable. When an argument is passed by reference, the changes made on the argument within the method are also reflected to the calling method.

If a variable is never used after a code fragment, that is identified as a candidate for extract method refactoring in the original method, this variable can be passed to the method by value. Variables, that have been used after the fragment to be extracted in the original method, have to be passed to the extracted method by reference. Therefore, we suggest to use "pass by reference" whenever the programming language allows, to simplify the process of extract method refactoring.

B. Return Values From Extracted Methods

As stated earlier, IDEs like Eclipse and Visual Studio support extraction of a set of preselected statements as a new method. These IDEs have some limitations when extracting certain types of code. For example, Eclipse requires the selected code fragments not to include any return statement. Visual Studio, similarly, puts some limitation on the code with return statements. When selection contains return statements, all paths are expected to be terminated by a return statement too.

Code fragments with return statements need to be handled carefully especially when not all paths in an identified code fragment are terminated by a return statement. This is one of the greatest challenges and limitations in extraction process of

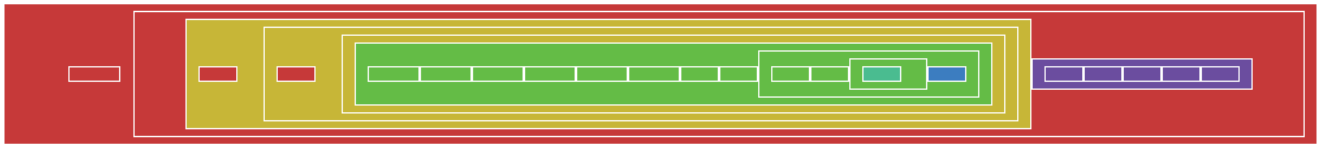


Figure 5 Experiment 1

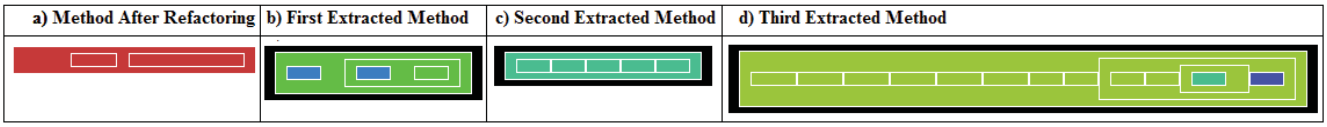


Figure 6 Refactoring Result

method refactoring. We will expand on this problem in a future publication that we are currently working on.

C. Bound Blocks

There are two key words presence of which precludes extraction of the code fragments that they reside in. These key words are "continue" and "break" that respectively carries a loop to next iteration or halts the loop. If a code fragment contains one of these two key words and the corresponding loop is not included in the fragment, then this code fragment cannot be considered for extract method refactoring. In other words, if a code fragment or node in the placement tree contains one of the key words, "continue" and "break", then this node is bound to its ancestor node that represents the loop associated with these keywords.

In Figure 7, we show an example code fragment that cannot be considered for extraction as a new method. The code fragment between lines 601 and 605 is bound to the code fragment between line numbers 591 and 606. Whenever two such scopes are bound, they have to be moved together in case of refactoring.

```
590  int tri=1;
591  for(int j=i+1; j<tc.length(); j++)
592  {
593      if(tc[j]==">")
594      {
595          tri--;
596      }
597      if(tc[j]=="<")
598      {
599          tri++;
600      }
601      if(tri==0)
602      {
603          i=j;
604          break;
605      }
606  }
```

Figure 7 A Code Fragment with "break"

VII. Experiments and Results

We have run our analysis and visualization tools on several methods from different systems. Identified code fragments in these methods are then extracted as new methods following the process explained in Section 5. Throughout the experiments, parent protection approach explained in Section 3 is adopted. Due to page limitation, we cannot put the source code in this paper. Yet, programs before and after refactoring can be found at [12].

A. Experiments from our Analysis Tool

We first applied this technique to a method in our tool. This method basically analyses a statement to find data declaration and references in the statement. Figure 5 shows the placement tree of the original version of the analyzed method.

We extracted three new methods and ran our tool again after refactoring. Figure 6 shows the placement trees for the methods we generated. During refactoring, code fragments with small lengths (usually two lines of codes) were not extracted, as our main focus is on extracting large code fragments.

For all the test cases that we used from our analysis tool, we could easily come up with meaningful names for the extracted methods based on their respective tasks. This demonstrates that our approach, with a high probability, will identify fragments that have a distinct task in the larger operation of the whole method. Other experiments from this domain and their visualizations can be found in [12].

B. Experiments from Open source Projects

Method used for this experiment is taken from a research project written by a group we collaborate. This method basically implements a part of reconstruction process of medical images obtained using cone beam and/or parallel beam collimators. The original function before any processing or refactoring has nearly 400 lines of codes with comments and white spaces. Figure 8 shows a portion of placement tree for the original version of this method.

We extracted nine methods and ran our tool again on these methods after refactoring. Figure 9 shows the placement trees for these methods. After refactoring the method has less than 40 lines. This improves the readability of the code a lot and makes the code more comprehensible reducing its complexity. Hence the overall maintainability of the whole system is improved as the developer has the chance to work on smaller and less complex methods after refactoring.

Another method used for our experiment is one of the longest methods from this research project with nearly 4000 lines of codes with comments and white spaces. Our tool clearly detects many code fragments as extract method refactoring candidates and as shown in Figure 10, one can easily observe candidate code fragments for extraction. Figure 10 shows just a portion of the placement tree for this method. Other parts of the placement tree are not any less diverse in terms of colors of nodes or their dominant variables.

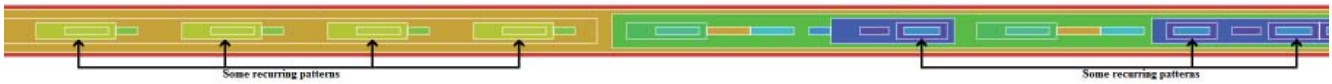


Figure 8 Experiment 2

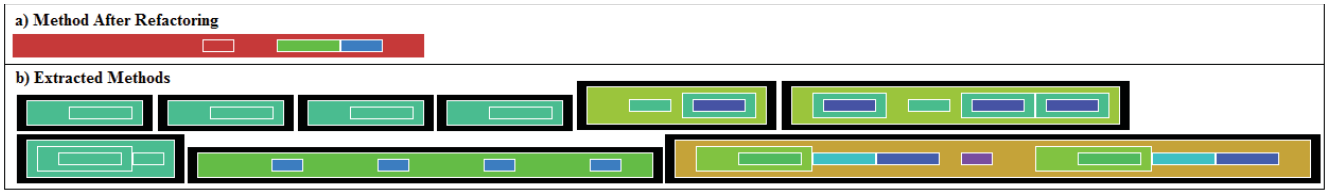


Figure 9 Refactoring result for Experiment 2

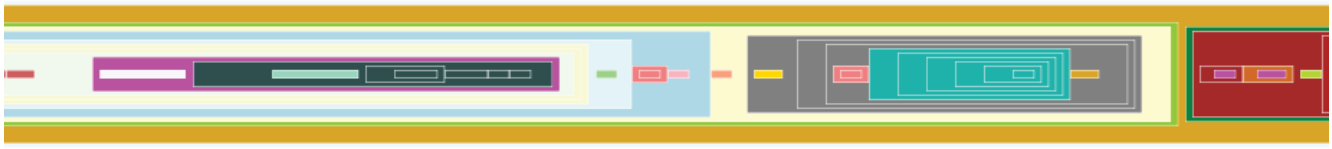


Figure 10 Experiment 3

Figure 11 Experiment 4

Another software that we used for our experiments is called *Notepad++*. *Notepad++* is an open source code editor and Notepad replacement that supports several programming languages and natural languages [11]. Analyzed method, *feedGUIParameters*, has more than 800 lines of code. Figure 11 shows a part of the placement tree for this method. As shown in Figure 11, our tool was able to indentify large code fragments that are candidates for extract method refactoring.

I. Conclusion and Future Work

In this paper we mainly focus on identification of code fragments for extract method refactoring. Our identification process, as stated earlier, is based on placement tree and variable reference counts in each node of this tree. This approach is straightforward to implement and it effectively works in real software systems as shown in the experiments.

In this work, initially we did not target refactoring to reduce total number of statements in systems by detecting and removing duplicated code. Yet, our visualization reveals that variable reference counts can also be used for this purpose. As shown in Figure 8, we have encountered quite a lot recurring patterns in our placement trees. Such recurring patterns can be found in Figure 9 as well in the placement trees of the new methods. When we compared the corresponding code fragments of these recurring nodes, we saw that some of these code fragments were identical to each other, while for some, there was a tremendous similarity between corresponding code fragments, although they were not identical. This shows that our approach with some improvement can be used to detect duplicated code as well. This shapes the future direction of our research together with a study to minimize the number of parameters that extracted methods require.

References

[1] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, "Refactoring: Improving the Design of Existing Code," Addison Wesley, Boston, MA, 1999.

[2] R. Komondoor and S. Horwitz, "Effective, Automatic Procedure Extraction," Proceedings of the 11th IEEE International Workshop on Program Comprehension, pp.33, May 10-11, 2003.

[3] M. Kaya and J. W. Fawcett, "A New Cohesion Metric and Restructuring Technique for Object Oriented Paradigm," Computer Software and Applications Conference Workshops (COMPSACW), 2012 IEEE 36th Annual , pp.296-301, 16-20 July 2012.

[4] N. Tsantalis and A. Chatzigeorgiou, "Identification of Extract Method Refactoring Opportunities," Software Maintenance and Reengineering, 2009. CSMR '09. 13th European Conference on , pp.119-128, 24-27 March 2009.

[5] T. Sharma, "Identifying extract-method refactoring candidates automatically," In Proceedings of the Fifth Workshop on Refactoring Tools (WRT '12). ACM, New York, NY, USA, pp.50-53, 2012.

[6] K. Maruyama, "Automated method-extraction refactoring by using block-based slicing," SIGSOFT Softw. Eng. Notes 26, pp.31-40, May 2001.

[7] A. Lakhotia and J.-C. Deprez, "Restructuring Programs by Tucking Statements into Functions," Information and Software Technology, vol. 40, no. 11-12, pp. 677-690,1998.

[8] R. Komondoor and S. Horwitz, "Semantics-preserving procedure extraction," In Proceedings of the 27th ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL '00). ACM, New York, NY, USA, pp.155-169, 2000.

[9] http://homes.ieu.edu.tr/~kkurtel/Documents/IEEE%20Std%201219-1998%20Software%20Maintenance.pdf

[10] D. Turo and B. Johnson, "Improving the visualization of hierarchies with treemaps: design issues and experimentation," Visualization, 1992. Visualization '92, Proceedings., IEEE Conference on, pp.124-131, 19-23 Oct 1992.

[11] http://notepad-plus-plus.org/

[12] http://www.lcs.syr.edu/faculty/fawcett/handouts/Research/kaya/ap.pdf

Mutation Analysis for JavaScript Web Application Testing

Kazuki Nishiura and Yuta Maezawa
The University of Tokyo
Tokyo, Japan
{*k-nishiura, maezawa*}*@nii.ac.jp*

Hironori Washizaki
Waseda University
Tokyo, Japan
washizaki@waseda.jp

Shinichi Honiden
The University of Tokyo, National Institute of Informatics
Tokyo, Japan
honiden@nii.ac.jp

***Abstract*—When developers test modern web applications that use JavaScript, challenging issues lie in their event-driven, asynchronous, and dynamic features. Many researchers have assessed the adequacy of test cases with code coverage criteria; however, in this paper, we show that the code coverage-based approach possibly misses some faults in the applications. We propose a mutation analysis approach for estimating the fault-finding capability of test cases. We assume that developers can find overlooked fault instances and improve the test cases with the estimated capability. To create a set of faulty programs, i.e., mutants, we classify the JavaScript features in web applications and then define a comprehensive set of mutation operators. We conducted a case study on a real-world application and found that our approach supported the improvement of test cases to expose hand-seeded faults by an extra ten percent.**

***Keywords*-JavaScript; Mutation Analysis and Testing; Web Applications; Test Criteria**

I. Introduction

Developers implement client-side JavaScript programs (henceforth, JS programs) to make their web applications interactive. JavaScript provides APIs for handling user events, requesting asynchronous messages, and dynamically manipulating Document Object Models (DOM)[1] to rewrite the contents on a web page. Ocariza et al. reported that 97 of the 100 most visited web sites utilized JavaScript [1].

Since event-driven, asynchronous, and dynamic features can increase the complexity of web applications using JS programs (henceforth, JSWAs), researchers have sought to address challenging issues of testing JSWAs [2], [3]. In those researches, they have leveraged code coverage criteria to assess the adequacy of test cases. However, these approaches might not show an absence of faults, even if they explore all statements and branches without exception. This is because JS programs have dynamic characteristics; e.g., assigning any values to a non-existent property does not throw any exceptions (See Section II-C).

Mutation analysis is a fault-based technique that provides strong test criteria. The technique injects artificial faults into the software under test. Fault injection is done by applying mutation operators that represent fault types that developers would like to discover. By running test cases on faulty versions of software, developers can estimate the fault-finding capability of test cases.

Researchers have indicated the usefulness of mutation analysis for JSWAs [4]. Recently, some researchers have started applying mutation analysis techniques on JSWAs [5]–[7]. However, they focus on the specific characteristics of applications (e.g., preventing cross-site scripting vulnerability) or heuristically choose mutation operators. Their approach may give a high score to test cases that consider some types of faults but do not consider others.

In this research, we try to define a comprehensive set of mutation operators that cover JavaScript's features by conducting a feature analysis of JavaScript used in web applications. We define ten mutation operators and implement a tool for performing mutation analysis on JSWAs. Using the operators, our tool generates mutants from an original JS program. By executing test cases on mutants, developers can learn the fault-finding capability of the test cases and see unexposed fault instances, i.e., unkilled mutants. Developers can then add or modify the test cases so that more faults can be exposed. We evaluated our tool by surveying real faults from a public bug repository and carrying out a case study on a real-world application.

Our contributions are as follows:

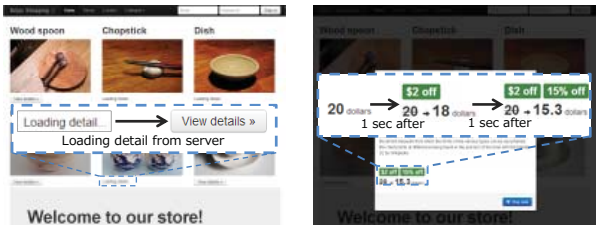
1) Proposal of a comprehensive set of mutation operators focusing on the features of JavaScript in web applications.
2) The AjaxMutator, an implementation of our approach.
3) A short survey on real faults and a case study on a real-world application whose results show that our tool can help developers improve their test cases.

II. Background

A. Mutation analysis

Mutation analysis is a fault-based technique to assess the adequacy of test cases [8], [9]. First, the technique makes a small change to a program under test in order to create faulty programs called *mutants*. The changes depend on fault-seeding rules called *mutation operators*. Then, the technique tests both the original program and each mutant with the given test cases. If one of the mutants gives a different test result from the original, the mutant is said

[1] http://www.w3.org/DOM



(a) An item list page (b) An item detail view

Figure 1: E-commerce web application

to be *killed*. Killed mutants indicate that the test cases can find such faults. Hence, the technique measures the ratio of the number of killed mutants to the number of all created mutants as a *mutation score*. By referring to the mutation score, developers can estimate the adequacy of their test cases.

Although mutation analysis can provide strong criteria for estimating test adequacy [10], its result depends on the definitions of the mutation operators. In this research, we define a comprehensive set of mutation operators that covers all the mandatory features of JSWAs.

B. JavaScript web applications

Modern web applications combine various technologies as described in [11] to realize their interactive web pages. Because JavaScript binds all of these technologies, we argue that JavaScript is a central technology.

To make their applications interactive, developers implement JS programs as follows: When users operate an application, JS programs i) continuously process user requests with event handlers, ii) asynchronously receive the necessary data, and iii) dynamically update web page contents. Analyzing JS programs is a challenging issue because of its event-driven, asynchronous, and dynamic features. Here, we focus on these JavaScript features when defining the mutation operators.

C. Motivating example

We explain the inadequacy of the coverage criteria for testing JSWAs using a typical e-commerce application[2] as our motivating example (Fig. 1). Additionally, we show some of the JS program of the application using jQuery[3] in Figure 2. Here, $(''\#foo'')$ and $(''.bar'')$ are function calls that select DOM elements whose *ID* and *class* attributes correspond to "foo" and "bar", respectively. $Function\#bind(thisArg[, arg1[, arg2[, ...]]])$ is a built-in method of a *Function* object that sets its $this$ keyword and arguments as provided values.

This application first shows an item list page (Fig. 1a). The page does not initially contain item details about these items

[2]http://www.honiden.nii.ac.jp/~k-nishiura/e-commerce-example
[3]http://jquery.com

```
1  $(document).ready(function()  {
2    $(".item").each(function()  {
3      var itemId = $(this).attr("data-item-id");
4      var loadingMsg = $("<span>");
5      loadingMsg.addClass("message" + itemId);
6      loadingMsg.text("Loading detail...");
7      $(this).append(loadingMsg);
8      requestItemDetail(itemId);
9    });
10 });
11
12 function requestItemDetail(itemId) {
13   $.getJSON('item.php?id=' + itemId,
14     function(data) {
15       var button = $("#itemButton" + data.id);
16       button.text("View detail...");
17       button.click(
18           showDetail.bind(null, data.body, data.price,
19           data.discount1, data.discount2));
20       $(".message" + data.id).get(0).textContent='';
21       button.show();
22   });
23 }
24
25 function showDetail(bodyHtml, price, discount1, discount2){
26   // ...........
27   detailBody.html(bodyHtml);
28   // ...........
29   setTimeout(showDiscount.bind(details, discount1), 1000);
30   setTimeout(showDiscount.bind(details, discount2), 2000);
31 }
```

Fault1 …$("#itemButton1"…
Fault2 …data.text…
Fault3 …textConent = '';
Fault4 …2000);
…1000);

Figure 2: JavaScript program and possible faults

```
1  public void testShowingDetail() {
2    WebDriver driver = new FirefoxDriver();
3    driver.get(TARGET_URL);
4    WebDriverWait wait = new WebDriverWait(driver, WAIT_LIMIT_SEC);
5    wait.until(visibilityOfElementLocated(By.id("itemButton10")));
6    WebElement showDetailButton
7        = driver.findElement(By.id("itemButton10"));
8    assertEquals("View detail...", showDetailButton.getText());
9    showDetailButton.click();
10   wait.until(visibilityOfElementLocated(By. className("modal")));
11   driver.findElement(By.className("buy-now")).click();
12 }
```

Figure 3: Test case for JavaScript program in Figure 2 (extracted)

to reduce the amount of data that has to be communicated for fast rendering. After the whole page has been loaded, the JS program asynchronously sends requests for the detail of each item to a server (lines 8, 12-23) while displaying a *loading* message (lines 4-7). When it receives a response containing the discount information about the item, the JS program removes the text message (line 20). Instead, it registers an event handler for displaying the information to a *view detail* button (lines 17-19) and then displays the button (line 21). When the user clicks the button, the program rewrites the DOM to show a detailed view of the item (Fig. 1b) without any page transitions (line 27). In this view, the application displays the information using timer events for a visual effect (lines 29 and 30).

Figure 3 shows the test code implemented with `Selenium WebDriver`[4] for testing the program described above. WebDriver enables test designers to emulate user operations such as mouse clicks by implementing them

[4]http://seleniumhq.org

Table I. Faults that cannot be exposed with code coverage criteria

Fault	Related feature	Unexpected behavior in our motivating example	Lines in Fig. 2
1	User event target	Registers a “click” event without an exception even if there is no “button”.	15 and 17
2	Async. comm. response	Displays an *undefined* value by referring to a non-existent “data.text” property.	18 and 27
3	DOM attr. manipulation	Continuously displays “Loading detail” because of a misspelled “textConent” property.	4-6 and 20
4	Timer event interval	Displays an incorrect price even though both timer events are handled because the intervals are incorrect.	29 and 30

in test code. Given our test code in Figure 3, the framework proceeds as follows: WebDriver first launches Firefox and opens a target web application (lines 2 and 3). The framework waits until the application displays a *view detail* button (line 5). Upon finding the button, the framework clicks the *detail* button (line 9) and finally clicks a *buy* button (line 11). Note that when testing web applications, developers need to consider the timing of the user operations. Therefore, the methods of wait object (lines 5 and 10) can be used as assertions that raise an exception when a given condition is not satisfied within a certain period of time.

The test code provides 100% statement and branch coverage for the JS program shown in Figure 2. However, it is not able to find the faults shown in Figure 2. We argue that this inadequate capability derives from the dynamic features of JavaScript. Table I explains these faults. Regarding **faults 1 and 4**, for example, although the JS program explicitly determines the user event targets and timer event intervals, it does not check for their existence or correctness at runtime. As for **faults 2 and 3**, it does not throw any exceptions even if the running application does not have any corresponding properties or DOM elements. Hence, exploring an entire program without exceptions does not always indicate an absence of faults. In the next section, we propose a mutation analysis considering JavaScript features.

III. Mutation Analysis

Figure 4 shows an overview of our mutation approach:

1) Developers implement test cases to test whether a JSWA runs as expected.
2) Our tool generates mutants of JS programs with our proposing mutation operators.
3) Our tool executes the test cases on the mutants to check if the test cases detect the mutants. Developers can know the adequacy of the test cases (i.e., mutation score) and which mutants remain unkilled with given test cases.
4) Developers add test cases to kill the unkilled mutants. Then, our tool recalculates the mutation score by running additional test cases on unkilled mutants. This process is repeated until the mutation score reaches a certain threshold [12].

In this way, our approach can help developers make test cases with better fault-finding capability.

To expose faults in JS programs, we need to define mutation operators by focusing on JavaScript features. Therefore,

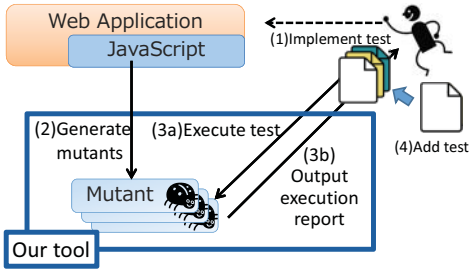


Figure 4: Workflow to improve test suites by mutation analysis

we first conducted a feature analysis on JavaScript. Then, we defined the mutation operators based on the results of the analysis.

A. JavaScript features in web applications

[13] describes three characteristics that distinguish JSWAs from traditional web applications; event driven model, asynchronous communication, and DOM manipulation. We conducted a feature analysis on each feature and developed feature diagrams [14] as shown in Figure 5.

Event driven model: In comparison with traditional web applications, JSWAs leverage JS programs for processing user operations and elapsing time without page transitions. For instance, when a user clicks the ”item detail” button (user event), our motivating example displays the discount information after a second (timer event).

When implementing user events in JS programs, developers determine the `target`, `event type`, and `callback function`. The target corresponds to the built-in Window object or to DOM elements such as buttons. JS programs register a callback function to the event type of the target. As for timer events, JS programs register a `callback function` to an `interval` of elapsed time. Developers can also optionally determine `repeat`, i.e., whether the program repeatedly handles the timer event.

Asynchronous communication: Asynchronous communication enables web applications to continuously accept user operations while waiting for server responses. For instance, our motivating example lets the users browse an item list while it loads the details of each item.

The two main constituents of asynchronous communication are `requests` and `responses`. A request must contain a destination URL and a request method (e.g., GET, POST, etc.). Request parameters such as item IDs are optionally included. A server processes the request and sends a response to the application. A response contains the

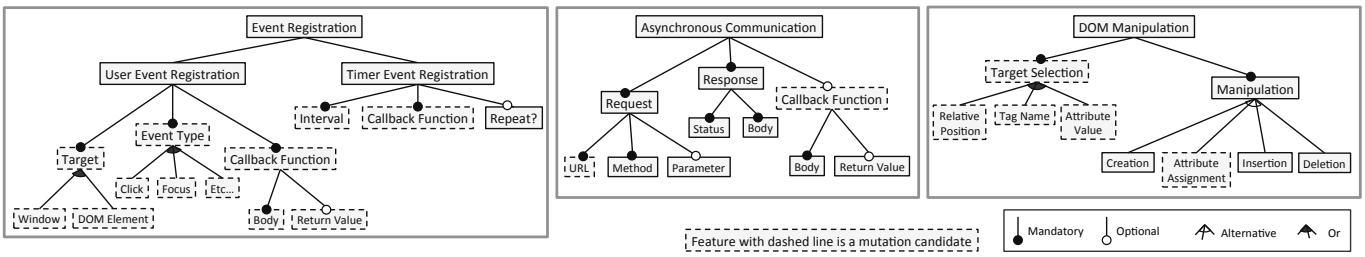


Figure 5: Feature models of JavaScript in web applications

Table II. Proposing mutation operators based on features of JavaScript in web applications and example mutations.

JavaScript feature	Operator name	Original code	Mutated code
User event registration	Event target replacement	buyButton.click(requestBuy)	**cancelButton**.click(requestBuy)
	Event type replacement	button.click(showDetail)	button.**mouseover**(showDetail)
	Event callback replacement	cancelButton.click(closeModal)	cancelButton.click(**requestBuy**)
Timer event registration	Timer interval replacement	setTimeout(callback, 1000)	setTimeout(callback, **2000**)
	Timer callback replacement	setTimeout(showDiscount1, 1000)	setTimeout(**showDiscount2**, 1000)
Asynchronous communications	Request target replacement	$.get('item.php', showItem)	$.get(**'item_list.php'**, showItem)
	Request onsuccess callback replacement	$.get('item.php', showItem)	$.get('item.php', **buyItem**)
DOM manipulation	Nearby DOM element	$("#items").append(newItem)	**$("#items").parent()**.append(newItem)
	Attribute assignment target replacement	element.id = "cancelButton"	element.**textContent** = "cancelButton"
	Attribute assignment value replacement	element.id = "cancelButton"	element.id = **"buyButton"**

status code for signaling success, failure, etc. and a body text. Callback functions are invoked according to the status code if the developers choose to implement them.

Note that JavaScript also provides a means for synchronous communications, but we do not regard it as a feature of JSWAs. Synchronous communication blocks UI threads, and best practice is not to use them.[5]

DOM manipulation: JavaScript provides DOM APIs for manipulating web page elements on the client-side. Such partial updates can make applications more responsive than refreshing whole web pages with page transitions. For instance, our motivating example leverages DOM manipulations to display the item detail view.

DOM manipulations consist of `target` DOM elements and `methods` of manipulating these elements. JS programs select this target element by its position relative to another element, tag name, or ID/class value. As for the method, the programs create, insert, delete the element, or alternatively, assign a value to an attribute of the element.

B. Proposed mutation operators

Here, we describe our ten mutation operators for JS programs (See Table II) and explain how we developed them.

User and timer event registrations: Since developers intentionally implement the optional features of user and timer events, we assume that they typically embed faults in mandatory features, for example, faults 1 and 4 in our example. Hence, we decided to focus on the mandatory features where our approach makes little changes (henceforth, mutation candidates). When mutating a mutation candidate, our approach replaces it with another candidate. For example, consider the `event target` in Table II. Our approach replaces the event target "buyButton" with "cancelButton". Similarly, it also mutates the event types of user events, the timer intervals of timer events, and the callback functions of both.

Asynchronous communications: Although asynchronous communication has two mandatory features, their responses are outside the scope of our study. This study focuses on client-side logic, but the responses depend on the server-side logic. As for the requests, we select only destination URLs as mutation candidates because the differences in the request method should be properly processed by the server-side logic. Additionally, we claim that leveraging responses plays an important role in JSWAs such as when preparing the item details in our motivating example. Therefore, our approach deals with the on-success callback functions as mutation candidates, although this feature is optional.

DOM manipulation: We define a mutation operator called the `nearby DOM element`. This definition is based on our heuristic that developers tend to incorrectly select a DOM element that is near a proper one. Therefore, our approach replaces the target DOM element with its parent or child element.

As for DOM manipulations, our limited implementation does not yet cover creating, inserting, and deleting DOM elements, although such an implementation is planned as future work. Note that our tool can create mutants that insert/delete DOM elements at improper positions and our tool ran as expected to improve test cases in our case study (Section IV-B). As for assigning attributes, we define two mutation operators, one for replacing the attributes

[5]http://blogs.msdn.com/b/wer/archive/2011/08/03/why-you-should-use-xmlhttprequest-asynchronously.aspx

Table III. Real faults in WordPress

Ticket #	JavaScript feature	Brief explanation
1895	User event	Program does not properly register event handlers to user events.
8812		
2184	DOM manipulation	Program creates improper DOM elements
9740		Program selects improper DOM elements

Table IV. Detail of our initial and improved test suites

Test suite	#TC	#A	#W	Cov(%)	MS	FF(%)
Initial	2	6	16	95	45.9	89.5
Improved	+5	+21	+21	100	67.0	100.0

themselves and one for the assigned values.

We implemented our approach in a prototype tool called *AjaxMutator*. This tool is publicly available.[6]

IV. Evaluation Design

To assess the usefulness of our approach, we conducted a short survey about real faults and a case study using our tool. Our research questions are as follows:

- RQ1 Can JavaScript features really cause faults?
- RQ2 Can developers improve test cases with our tool to find faults that remain unexposed by following the code coverage criteria?
- RQ3 Can developers improve the test cases with our tool in a reasonable amount of time?

We first describe design of the survey and case study, and then discuss their results in the next section.

A. Survey: Faults in WordPress

We leveraged the public bug repository of WordPress[7] to survey real faults. In accordance with [15], we searched this repository using the keywords "JavaScript", "js", and "console" and then selected only `closed` bugs from the search results. Next, we manually extracted faults that a JS program clearly caused from the selected bugs. Finally, we extracted the faults related to the JavaScript features discussed in this paper.

B. Case Study: Evaluation of Test Cases for Quizzy

We conducted a case study on a quiz application called Quizzy.[8] This application has 5561 lines of code, including 310 lines of a JS program. We prepared two initial test cases using WebDriver. One represented a normal use case in which users answer quizzes and see their total score. Another is for testing invalid use cases in which users click an answer button before selecting any answer candidates. We conducted the mutation analysis with our tool and added test cases to kill unkilled mutants.

After that, we evaluated how well our tool can assess the fault-detecting capability of the test cases with hand-seeded faults. We asked an undergraduate student with two years industrial experience developing JSWAs to seed typical faults into the application. While he seeded faults, we did not explain our work to him.

[6]https://github.com/knishiura-lab/AjaxMutator
[7]http://core.trac.wordpress.org
[8]http://quizzy.sourceforge.net

The manual setup of the application, the test cases implemented, and the details of the seeded faults are publicly available.[9]

V. Results and discussion

Reality of JavaScript faults (RQ1): In our survey, we found 26 closed bugs that JS programs clearly caused. Among these bugs, we extracted four faults related to our focus in Table III. Note that the other faults were logic faults such as those related to parsing strings, and we expect that the existing approaches can deal with them. However, the faults related to JavaScript features are less studied, and they are more difficult to find, as we discussed in Section II-C. Hence, we claim that JavaScript features can cause real faults that should be exposed.

Improving test cases (RQ2): In our case study, the participant seeded 20 faults in the Quizzy application. Note that we dealt with 19 faults in total, because one seeded fault did not change the behaviour of the applications. These 19 faults consisted of one user event fault, one asynchronous communication fault, two DOM manipulation faults, and other faults such as typos.

Table IV shows the details of the initial and improved test suites. To compare the sizes of the test suites, we list the number of test cases (`#TC`), assertion statements (`#A`), and wait statements (`#W`). To evaluate the adequacy of the test cases, we measured statement coverage using jsCoverage[10] (`Cov`), mutation score (`MS`), and the ratio of found faults to the 19 seeded faults (`FF`).

Although the initial test suite covered all of the statements where faults were seeded, it exposed only 17 faults among the 19 faults (about 10% of the seeded faults were unfound). In addition, their mutation score was low. For instance, in this application, users can choose an option in two ways: by clicking a radio button or clicking a label. Initial test cased only care for clicking a radio button, so they did not kill the mutants that only affected labels. Additional test cases were implemented to kill such unkilled mutants. By adding five test cases, we could improve mutation score by about 20 and it was able to expose all hand-seeded faults. These results suggest that developers can find unexposed faults by increasing the mutation score with our mutation analysis.

As for the mutation analysis, our tool generated 109 mutants, and we divided them into four groups (Fig. 6). The blue bar in the figure stands for mutants killed by our

[9]https://github.com/knishiura-lab/
[10]http://siliconforks.com/jscoverage

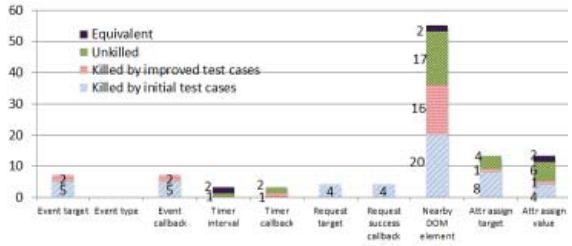


Figure 6: Details of mutants generated by our tool.

initial test cases. The red bar indicates the mutants which the initial test cases could not kill but the improved test cases could. The green and purple bars represent unkilled mutants with both test cases and equivalent mutants.

Note that eliminating equivalent mutants is a challenging issue in the field of mutation analysis [9]. As for the unkilled mutants with either test case, we found another challenging issue for mutation analysis caused by the robustness of modern web browsers; browsers could automatically infer and set proper values at runtime independently of the JS programs. Despite these issues, we believe that our tool can generate enough mutants for developers to improve test cases in order to find unexposed faults.

Reasonable time (RQ3): It took us an hour to prepare test cases and another three hours to improve them. We argue that a few hours is a reasonably short among all cost required to assure the quality of JSWAs that may contain unexposed faults. Additionally, we can explore applying automated test case generation techniques [2], [3], [16] to our tool in order to reduce the manual cost. As for the execution cost, it took the tool 20 minutes to conduct the mutation analysis on the initial test cases. After that, we added test cases to increment mutation score, and recalculated mutation score, which took another 20 minutes. Although this was a reasonably short time, because mutants are independent, we can further reduce this time by modifying the tool to test several mutants in parallel.

Internal validity threats: In a case study we use a real-world application that authors of [17] have also leveraged in their experiments. Moreover, the faults were seeded by a student who did not know about our study. However, we cannot assure these faults represent real faults on JSWAs. Additionally, we prepared the initial test cases by considering possible use cases of the application and improved them by referring unkilled mutants, however, our knowledge of the proposed method could have potentially affected the result of our case study. To avoid these threats, we plan to conduct additional case studies on a real-world development.

External validity threats: Although our set of mutation operators covers the mandatory JavaScript features, we consider to define operators for the optional features in our future work. Moreover, we selected the WordPress bug repository for our survey, and although there are only a few public repositories containing JavaScript bugs, we can define other mutation operators by modeling more real faults from other repositories. We used a single application in our case study, and it would be interesting to investigate how the mutation operators developed here work for other applications. Note that the Quizzy application has all of the JavaScript features discussed in this paper.

VI. Related Work

Mutation analysis for web applications or JavaScript programs: Mutation analysis has been widely studied since it was first introduced in the 1970s [8], [9]. Praphamontripong and Offutt argued that existing mutation analysis techniques do not consider the characteristics of web applications. They proposed mutation operators for HTML and Java Server Pages [6]. Shahriar and Zulkernine defined mutation operators for PHP and JavaScript to evaluate the adequacy of tests for avoiding cross site scripting vulnerability [5]. Alshraideh conducted mutation analysis on JS programs by applying basic mutation operators such as rewriting arithmetic operands to automate unit testing [18]. Our study developed mutation operators by focusing on characteristics of JavaScript in web applications. It has a different focus from and is complementary to the previous research.

Recently, Mirshokraie et al. have proposed some mutation operators based on the common mistakes that inexperienced engineers make [7]. They also proposed some mutation operators related to DOM and XMLHttpRequest[11] as summarized in Table V. Note that we do not consider synchronous communication because developers should avoid using it not to block the UI thread. Implementing mutation operators regarding element insertion is included in our future work. In addition, Mirshokarie et al. dealt only with specific mistake examples and some features of JSWAs whereas we try to define comprehensive set of mutation operators. In fact, they did not consider mutation operators about event-driven nature that is one of the main features of JSWAs in our analysis.

Classification of real world faults: Marchetto et al. [19] and Guo et al. [20] surveyed typical faults in web applications. Because these surveys targeted any faults related to web applications, we argue that faults related to JavaScript are not sufficiently studied.

Ocariza et al. studied run-time JavaScript exceptions by automatically exploring popular web applications in the Alexa ranking [1]. They showed that the current quality assurance for JSWAs was so insufficient that even widely used applications threw run-time exceptions. Since their study utilized run-time exceptions as perceived by users, we claim that the actual faults behind such exceptions are not always clear.

[11] JavaScript object that provides a means for HTTP communications

Category	Mutation target	Our proposal	Mirshokraie's [7]
User event	3 targets (target, type, callback)	✓	NA
Timer event	2 kinds (interval, callback)	✓	NA
Asynchronous communications	Request destination URL	✓	✓
	Asynchronous or synchronous	NA (Out of scope)	✓
	Callback for a response	✓(replace on-success callback)	✓(change conditions on which callback is invoked)
DOM	DOM element selection	✓(replace with nearby DOM element)	✓(rewrite attribute of selection method)
	DOM attribute assignment	✓	✓
	Element insertion	NA (Future work)	✓
Other JavaScript-related	9 kinds (common mistakes for inexperienced programmers)	NA	✓

✓: Implemented, NA: Not Available

Table V. Comparison of mutation operators by our proposal and those by Mirshokraie et al. [7]

Ferrari et al. took a bottom-up approach that defined mutation operators based on real faults [21]. However, there are only few public bug repositories for JSWAs [15]. In addition, public bug reports are typically written by users. Thus, it is difficult to discern the actual fault causing the reported application behavior. Therefore, we chose a top-down approach that analyzes the features of JavaScript in web applications and then defined mutation operators corresponding to these features.

VII. Conclusion and Future Work

In this paper, we classified the features of JavaScript in web applications and defined mutation operators for them. By using our tool, developers can estimate the fault-finding capability of their test cases. We conducted a brief survey on real faults and a case study using a real-world application. The results of our evaluations indicated that our tool could expose faults including ones missed by coverage criteria. We conclude that our approach can help developers improve their test cases and find more faults.

Our future work will be in three main directions. First, we plan to refine the mutation operators for JavaScript web applications by surveying more real faults and conducting additional case studies. Second, we will use our approach to evaluate other methods that support tests such as automated testing. Third, we plan to investigate a new automated testing algorithm that can kill mutants efficiently.

References

[1] F. S. Ocariza Jr., K. Pattabiraman, and B. Zorn, "Javascript errors in the wild: An empirical study," in *Proc. Int'l Sym. on Softw. Reliability Eng. (ISSRE)*, 2011, pp. 100 –109.

[2] S. Artzi, J. Dolby, S. H. Jensen, A. Møller, and F. Tip, "A framework for automated testing of javascript web applications," in *Proc. Int'l Conf. on Softw. Eng. (ICSE)*. ACM, 2011, pp. 571–580.

[3] A. Mesbah and A. van Deursen, "Invariant-based automatic testing of ajax user interfaces," in *Proc. Int'l Conf. on Softw. Eng. (ICSE)*. IEEE Computer Society, 2009, pp. 210–220.

[4] A. Marchetto, P. Tonella, and F. Ricca, "Testing techniques applied to ajax web applications," in *Proc. WS on Web Quality, Verification and Validation*, 2007.

[5] H. Shahriar and M. Zulkernine, "Mutec: Mutation-based testing of cross site scripting," in *Proc. ICSE WS on Softw. Eng. for Secure Systems (IWSESS)*. IEEE Computer Society, 2009, pp. 47–53.

[6] U. Praphamontripong and J. Offutt, "Applying mutation testing to web applications," in *Proc. Int'l Conf. on Softw. Testing, Verification, and Validation WSs (ICSTW)*. IEEE Computer Society, 2010, pp. 132–141.

[7] A. Mirshokraie, Shabnam. Mesbah and K. Pattabiraman, "Efficient javascript mutation testing," in *Proc. Int'l Conf. on Softw. Testing, Verification and Validation (ICST)*. IEEE Computer Society, 2013, p. to appear.

[8] R. A. DeMillo, R. J. Lipton, and F. G. Sayward, "Hints on test data selection: Help for the practicing programmer," *Computer*, vol. 11, no. 4, pp. 34 –41, april 1978.

[9] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *IEEE Trans. on Softw. Eng.*, vol. 37, no. 5, pp. 649–678, Sep. 2011.

[10] A. J. Offutt, J. Pan, K. Tewary, and T. Zhang, "An experimental evaluation of data flow and mutation testing," *Softw. Pract. Exper.*, vol. 26, no. 2, pp. 165–176, Feb. 1996.

[11] Garrett, Jesse James. (2005, Feb.) Ajax: A new approach to web applications. [Online]. Available: www.adaptivepath.com/ideas/ajax-new-approach-web-applications

[12] A. J. Offutt and R. H. Untch, "Mutation testing for the new century," W. E. Wong, Ed. Norwell, MA, USA: Kluwer Academic Publishers, 2001, ch. Mutation 2000: uniting the orthogonal, pp. 34–44.

[13] D. A. Justin Gehtland, Ben Galbraith, *Pragmatic Ajax: A Web 2.0 Primer*. O'REILLY, 2006, ch. Ajax Explained, pp. 61–77.

[14] D. Batory, "Feature models, grammars, and propositional formulas," in *Proc. Int'l Conf. on Softw. Product Lines*. Springer-Verlag, 2005, pp. 7–20.

[15] F. S. Ocariza Jr., K. Pattabiraman, and A. Mesbah, "Autoflox: An automatic fault localizer for client-side javascript," in *Proc. Int'l Conf. on Softw. Testing, Verification and Validation (ICST)*. IEEE Computer Society, 2012, pp. 31–40.

[16] R. A. DeMillo and A. J. Offutt, "Constraint-based automatic test data generation," *IEEE Trans. on Softw. Eng.*, vol. 17, no. 9, pp. 900–910, Sep. 1991.

[17] Y. Zheng, T. Bao, X. Zhang, and W. Lafayette, "Statically locating web application bugs caused by asynchronous calls," in *Proc. World Wide Web (WWW)*, 2011, pp. 805–814.

[18] M. Alshraideh, "Complete automation of unit testing for javascript programs," *Computer Science*, vol. 4, no. 12, pp. 1012–1019, 2008.

[19] A. Marchetto, F. Ricca, and P. Tonella, "Empirical validation of a web fault taxonomy and its usage for fault seeding," in *Proc. Int'l WS on Web Site Evolution (WSE)*. IEEE Computer Society, 2007, pp. 31–38.

[20] Y. Guo and S. Sampath, "Web application fault classification - an exploratory study," in *Proc. Int'l Sym. on Empirical Softw. Eng. and Measurement (ESEM)*. ACM, 2008, pp. 303–305.

[21] F. C. Ferrari, J. Maldonado, and A. Rashid, "Mutation testing for aspect-oriented programs," *Proc. Int'l Conf. on Softw. Testing, Verification, and Validation (ICST)*, pp. 52–61, 2008.

A Knowledge-based Approach for Generating Test Scenarios for Web Applications

Rogene Lacanienta, Shingo Takada
Graduate School of Science and Technology
Keio University
Yokohama, Japan

Haruto Tanno, Morihide Oinuma
Software Innovation Center
NTT CORPORATION
Tokyo, Japan

***Abstract*— The popularity and complexity of current Web applications continue to increase, and thus, their quality and reliability are increasingly becoming more important. Software testing is an important part of validating the software, but it needs a large number of good quality test scenarios and test cases. This can be done manually, but it would be very time consuming. They can also be generated automatically, but previous work only considers artifacts of the software-under-test. Given that there are already a huge number of Web applications that have gone through rigorous testing, we believe that the information from these previous tests can be a good source of information for testing future Web applications. This paper proposes an approach to automatically produce test scenarios by leveraging on the knowledge gathered from existing Web applications that have already undergone software testing. Our approach is based on a database containing test scenarios that are collected from previous applications. When a tester wants to produce test scenarios for a new application, he/she first creates a "base" scenario of the application, and then searches the database for related scenarios. The retrieved scenarios are then used to test the applications. A case study showed that our approach can retrieve useful test scenarios that a professional tester was not able to build manually.**

Keywords-software testing; scenario generation; knowledge-based engineering; web applications

I. INTRODUCTION

Software testing is an important part of evaluating the validity and correctness of a software product, but it is widely known that software testing is both costly and time-consuming. Thus, many methods for automating tests have been proposed [1][2][3][4][5][6][7]. Their main goal is to reduce the cost of testing, effectively reducing the entire cost and time consumed for the entire software development cycle. However, they only consider information that is available from the software-under-test.

The world, being now very reliant on software, has produced countless working software of good quality that has gone under extensive (and expensive) testing. Each such software has various software artifacts, such as design documents, source code, test scenarios, test cases. In the past, such artifacts have been reused, e.g., code reuse and design reuse. We consider reuse of information concerned with testing.

We take a knowledge-based approach that taps into the potential of already-available, high-quality software artifacts (e.g., design documents, source code, test scenarios, and test cases) to produce robust test scenarios for future software. Kaner described a test scenario as a test based on how the program is used [8]. In this paper, we focus on a Web page, and view a test scenario as a sequence of steps that are taken to transition from one Web page to another. For example, a test scenario for registration may include steps such as input username, input password, and submit information. We will explain this further in subsection III.B.

Our approach is based on a database containing test scenarios that are collected from previous applications. A tester creates a base scenario of the software-under-test and uses that to query for related test scenarios.

The rest of this paper first starts with a discussion of related work. Section III then describes our approach. We give an overview and then give details on the test scenario database as well as the querying. Section IV describes a case study we conducted to compare our approach to a manually generated set of test scenarios. Section V makes concluding remarks.

II. RELATED WORK

Much work has been done on generating test scenarios and test cases. Xu, et al used adaptive agents that traverse a UML diagram representing a test scenario [1]. During traversal, the agents perform manipulation of the nodes based on a fixed algorithm. Andrews, et al. proposed a method to use finite state machines to model Web applications, which are then used with corresponding constraints in order to output test cases with minimal state space explosion [2]. Fujiwara, et al. introduced an approach that analyzes UML diagrams and uses a combination of Object Constraint Language (OCL) and Satisfiability Modulo Theories (SMT) solver to produce test cases [3]. Artzi, et al. used symbolic execution to dynamically create test cases and find bugs [4]. Halfond, et al. conducted static analysis to discover Web application interfaces and help generate test inputs [5]. Although not intended for Web applications, Pacheco, et al. developed Randoop which conducts random testing with feedback [6], and Gross, et al. proposed a genetic algorithm based approach based on GUI to generate high coverage test suite while avoiding false failures [7].

Works such as these effectively increase the coverage of test cases and/or their bug detection ability. But, they fail to consider the knowledge available from other possible sources of test information – that is, already deployed Web systems and their corresponding software artifacts. We consider how such information can be used to generate useful test scenarios.

III. KNOWLEDGE-BASED TEST SCENARIO GENERATION

A. Overview

Our approach is based on building a database of test scenarios, to which the tester queries to obtain test scenarios that could be used. The basic steps are as follows:

(1) The tester inputs a base test scenario.

(2) The base test scenario is analyzed. Each node in the scenario is used to query for related nodes in the database.

(3) Combinations of the nodes are made to generate scenarios that are related to the base scenario.

(4) The generated scenarios are ranked and returned to the tester.

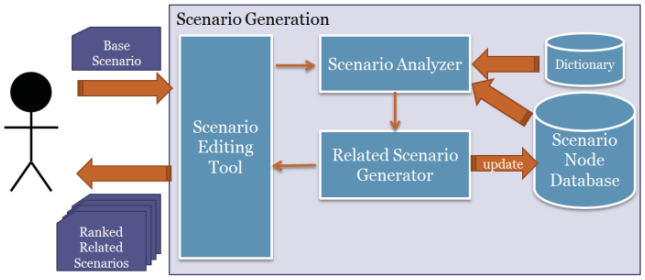


Figure 1. Overview of test scenario generation tool

Fig. 1 shows an overview of our test scenario generation tool. The tool is based on UMLet Tool [9], an open-sourced Java-based application for creating UML diagrams.

In the remainder of this section, we first describe the test scenario used in our approach. We next describe the database. Then, we describe the node matching procedure as well as related scenario generation. Finally, we describe the ranking of related scenarios.

B. Test Scenario

A test scenario is basically a sequence of steps that are taken to transition from one Web page to another. A tester can use our tool to draw a diagram of a test scenario, similarly to a UML activity diagram, to simplify and visualize the scenario easily. The idea is that a developer, barring the experience they have, can easily be familiarized in using the tool.

There are three basic types of nodes: 1) *Processing* is a node indicating an internal process in the Web page, 2) *Input* is a node indicating the information the Web page requires from the user, 3) *Output* is a node indicating some output by the Web page. Of course, there are also start and end nodes.

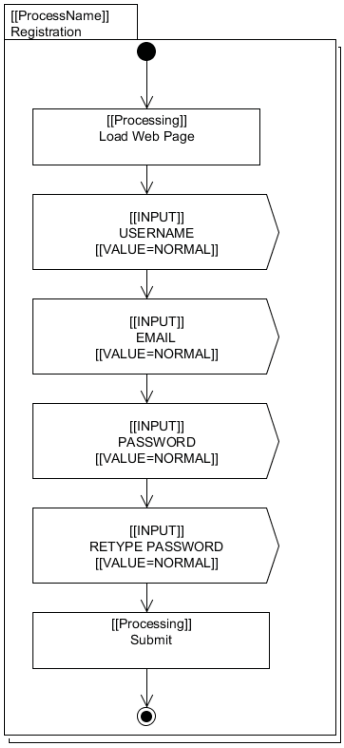


Figure 2. Example test scenario

Fig. 2 shows an example of a test scenario for the Registration process. It has 8 nodes (including the start and end nodes). The second node is a *Processing* node where a Web page is loaded. The following four nodes are all *Input* nodes, which require user input. Each *Input* node has an attribute and a value associated with it. For example, the third node has "USERNAME" as the attribute and "NORMAL" as the value.

C. Database

Each entry in the database corresponds to a node rather than a test scenario. The following information are included in each entry:

- Process Type: The type of operation where the node appeared. For example, this may be "Registration", "Log In", "Search," and "New Content".
- Node Type: Either *Processing*, *Input*, or *Output*. This paper mainly focuses on *Input*, as they make up the majority of nodes.
- Attribute Name: The name of the attribute.
- Node Value: The value of the node.
- Frequency: The number of times a node occurs in the various applications. This value plays an important role when ranking the generated related scenarios.

D. Node matching and related scenario generation

A related scenario is a scenario where one node in the base scenario has been replaced with one "matching" node from the

database. Ideally, we would not limit ourselves to one node and allow multiple nodes to be replaced. However, this may result in combinatorial explosion. Thus, we only consider related scenarios that differ from a base scenario by only one node.

For each node in the base scenario, the Scenario Analyzer searches for "matching" nodes in the database. The matching algorithm considers a hit when the node information and the database entry information match; more concretely, when the Process Type, Node Type, and Attribute Name all match. In case of *Input* type, the Scenario Analyzer searches for *Input* type nodes in the database with a "matching" attribute. There are two types of "matches": (1) exact match and (2) similar match.

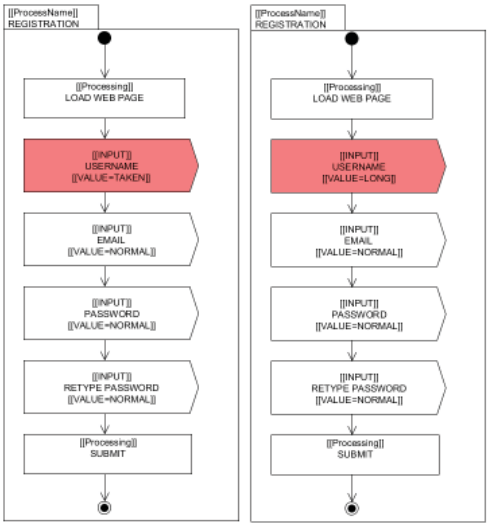
An exact match literally means that the attribute names for the nodes are the same. For example, a base scenario *Input* node with "Username" as an attribute, and an *Input* node in the database with "Username" as an attribute are considered to be exact matches. A similar match means that the two attribute names can be considered as synonyms. For example, an *Input* node in the database with "UserID" as an attribute can be considered as similar to a "Username" *Input* node. A dictionary is used to determine if two attribute names can be considered to be synonyms.

The resulting matching nodes are now sent to the Related Scenario Generator. Note that the matching nodes also have the values for the attributes. For example, the result of searching for an Input node with "Username" attribute and "valid" value (henceforth denoted as "Username=valid") may result in matches such as "Username=taken", "UserID=does not exist", etc.

Also note that each node will have a frequency value, which will be used for ranking the final related scenarios. In the case of a similar node, the original frequency value is weighted to take into consideration the factor that this particular node is not an exact match, but a close one. The default value of the weight is 0.75.

The Related Scenario Generator now takes each node in the base scenario, and replaces it with a matching node to form a related scenario. Note that each related scenario will only have one matching node; all other nodes in the related scenario are the same as the base scenario. Fig. 3 shows an example of two related scenarios given the base scenario in Fig. 2. In Fig. 3 (a), the value for the "USERNAME" node is changed from "NORMAL" to "TAKEN", meaning that the username has already been taken by someone and cannot be used for a new member. In Fig. 3 (b), it is changed to "LONG", meaning that the username is too long and need to be shortened.

The Related Scenario Generator also attaches a score to each related scenario. Scoring will be described in the next subsection.



(a) (b)

Figure 3. Example related test scenarios

E. *Scoring and ranking*

As we described in subsection III.C, each entry (node) in the database has a frequency value attached to it. The following formula is used to compute a score for each related scenario.

$$Score = \alpha * freq$$

where α is a coefficient taking a value between 0 to 1, and *freq* is the frequency value of node that was replaced.

When the match is an exact match, the value of α is 1. If it is a similar match using a dictionary, we have set the value to 0.75.

The ranking of the related scenarios are done based on the above score. A higher score basically means that the node occurs in many applications, and can thus be considered to be more important.

IV. CASE STUDY

We conducted a case study to evaluate our approach. The case study involved two parts: (1) database population and dictionary creation, and (2) comparison of manual vs. automatic test scenario generation.

A. *Database Population and Dictionary Creation*

As the proposed approach needs a database of scenario nodes, we first describe how we populated it. We also describe how the dictionary was created so that "similar" matches can be allowed.

We first conducted a survey of available open-source programs with "good" documentations. We concentrated on four types of Web application processes: 1) Registration of a new user, 2) Log In where a user inputs his/her credentials to enter the Web application, 3) Search where an element of the Web application can be searched, and 4) New Content where a new entry in the web application's database can be input. We

also concentrated on these six types of Web applications: a) CMS systems, b) Wiki systems, c) Forum applications, d) Blogging applications, e) Shopping and Hotel Management Systems, and f) Online Educational Systems. This resulted in a set of twelve open-sourced web applications.

The documentations and the source code were used to create the scenarios for the Web pages, and the database was updated with them. Each scenario node was either input into the database as a new node if it did not exist, or the frequency was updated if the same entry already exists.

The dictionary was created in parallel with the database population. Here, the database populator helps determine synonymous terms by selecting possible synonyms for attribute names of nodes if it has not been input yet. The resulting dictionary from the entire database population was then reviewed, and any mistakes were corrected.

The resulting knowledge database contains about 900 unique scenario steps and 180 dictionary terms.

The cost of constructing the database took about 30 man-hours. This included analysis, encoding, and saving the Web application scenarios.

B. Manual vs automatic test scenario generation

Test scenarios were generated for two open-sourced Web applications, both of them separate from the set used for database population. The two Web applications are as follows:

- Application 1: Tapestry Hotel Booking System, an open-sourced Java-based web application for hotel booking management [10]
- Application 2: Online Learning And Training (OLAT), an open-sourced Java-based web application for online learning [11]

We aimed to compare our approach (Part 1 below) with manual generation (Part 2 below) by measuring the quality, quantity, and time for creating test scenarios.

- Part 1 (Tool generated scenarios):

  A student with minimal degree of experience in software testing created one base test scenario for each target Web page of the two applications. Our tool then took the base scenarios and generated related scenarios.

- Part 2 (Manually created scenarios):

  Three professional software testers (one for Application 1 and two for Application 2) examined and analyzed the documentation, source code, and run time of the applications under test and created test cases for each target Web page. All three testers had at least five years of industry experience.

Both Part 1 and Part 2 involved a review process to verify the output test scenario. In Part 1, post-processing was performed to account for duplicates. This will be discussed further in the next subsection. In Part 2, review was also conducted to account for duplicates and identify inapplicable test scenarios.

The output (generated scenarios) of Part 1 and Part 2 are compared to see how effective our tool can come up with useful test scenarios.

C. Results And Analysis

We now evaluate our approach in terms of time, quantity, and quality of generated test scenarios.

1) Time

Table I compares the overall time taken to generate test scenarios by engineers and the student using the tool.

TABLE I. TIME FOR SCENARIO GENERATION

	Application 1	***Application 2***
Engineers	About 60 minutes	About 90 minutes
Student + Tool	25 minutes [base scenario creation = 23] [related scenario generation = 2]	27 minutes [base scenario creation = 25] [related scenario generation = 2]

Here, it can be seen that the student + tool combination generated test scenarios quickly. The main reason is that the student only needed 23 to 25 minutes to create the most basic scenario without dwelling too much into the documentation and source code. The tool also enabled the quick generation of test cases (about 2 minutes). On the other hand, the professional engineers needed to get a general idea of the Web page before being able to create test scenarios, and then manually write the scenarios. This resulted in the engineers taking two to three more times to generate the test scenarios.

2) Quantity and Quality

We compare the generated test scenarios and consider the quantity (number of scenarios) and quality (usefulness of scenarios).

Table II shows the number of generated scenarios. “App1” and “App2” are Application 1 and Application 2, respectively. “Org” stands for the original scenarios generated by our tool, while “Norm” stands for the normalized number of scenarios generated by our tool. “Eng” are the scenarios created by the engineers. “Registration”, “Login”, etc are the type of Web pages that were targeted for testing.

TABLE II. NUMBER OF GENERATED SCENARIOS

		Registration	**Login**	**Search**	**New Content**	**Total**
App1	Org	47	18	9	34	108
	Norm	24	13	8	13	58
	Eng	15	4	3	8	30
App2	Org	60	21	22	13	116
	Norm	41	18	14	8	81
	Eng	18	18	4	4	44

Scenarios were “normalized” due to duplicates. Duplicates occur when two particular node *values* use different wording to express the same concept. One example that occurred in the Registration process is when the values “taken” and “exists” were used for the attribute “username”. These duplicates occurred because the values for attributes were not strictly controlled during database population. Note that this is not an

issue that is currently handled by the dictionary, which only considers node attributes and not attribute values. Thus the normalization procedure consisted of manually checking the results and removing any duplicates. Note that the number of duplicates was significant (46% for Application 1 and 30% in Application 2).

From these data, we looked at the following:

(1) How many tool-generated scenarios correspond to the engineers' output and how many were missing?
(2) How many tool-generated scenarios were missed by the engineers?
(3) For (2), how many are actually significant and useful for testing the applications under test?

Tables III and IV show the results for the above questions (1) and (2). "TO" stands for tool-output, while "EO" stands for engineer-output. Also, the tool-output numbers are based on the normalized values.

TABLE III. COMPARISON OF EO AND TO FOR APPLICATION 1

	TO	EO	Both in EO and TO	In EO, not in TO	In TO, not in EO
Registration	24	15	14	1	10
Log In	13	4	4	0	9
Search	8	3	3	0	5
New Content	13	8	7	1	6
Total	58	30	28	2	30

TABLE IV. COMPARISON OF EO AND TO FOR APPLICATION 2

	TO	EO	Both in EO and TO	In EO, not in TO	In TO, not in EO
Registration	41	18	13	5	28
Log In	14	4	4	0	10
Search	18	18	13	5	5
New Content	8	4	2	2	6
Total	81	44	32	12	49

Our tool was able to generate a majority of the test scenarios that the engineers created (81%=(28+32)/(30+44)). The test scenarios that were not generated included cases where the wordings of a particular input field in the Web application did not correspond to any database or dictionary entry. For example, in the Registration process of Application 1, an "Enter Text Display" field asks for a security text (a captcha). In this case, the knowledge stored in the database and dictionary did not match the entered information, even if, technically, relevant related scenario nodes were available. If the attribute name was set to "captcha", "security text", or any similar entry, then relevant nodes would have matched and a related scenario could have been generated.

For question (3), we looked at the scenarios generated by the tool but not present in the engineers' output set. These scenarios were manually analyzed, using assistance from the engineers, to determine which tool-generated test scenarios were useful and which were not ("noise").

Table V shows the result of analyzing the usefulness of the generated scenarios for two applications. It breaks down the scenarios generated only by our tool (and not by the engineers) into either useful or noise.

TABLE V. ANALYSIS OF GENERATED SCENARIO USEFULNESS

		In TO, not in EO	Useful	Noise
App 1	Registration	10	4	6
	Log In	9	3	6
	Search	5	0	5
	New Content	6	6	0
	Total	30	13	17
App 2	Registration	28	20	8
	Log In	10	4	6
	Search	5	1	4
	New Content	6	4	2
	Total	49	29	20

For all the generated scenarios that were not in the engineer's output, 53% were found to be useful and 47% were not. However, the number of useful scenarios varied significantly among the Web pages. In some cases, the tool output a large number of useful scenarios not generated by the engineer. However, there were also instances that the tool failed to produce any useful ones.

The most common source of noisy scenario occurred due to differences in input type. For example, suppose that the base scenario has an *Input* node with "description" as the attribute name, and it exactly matches an *Input* node in the database. However, if the *Input* node in the base scenario is a text field requiring a text input, while the one in the database is a checkbox which can take a value of "true" or "false", then the generated scenario will not be useful. This can be solved in the future by adding input type information to the node.

Another common cause of noise is when synonyms were erroneously added to the dictionary. For example, "Search" and "Search in Product Description" were treated as synonyms during the dictionary creation, but these were actually completely different entities. This issue can be solved by improving the content and quality of the entries in the dictionary.

3) Test Scenario Reduction

The result of the case study indicates that the tool can generate quite a large number of scenarios which may contain some non-useful ones. As with any test case generation method, reducing the number of test cases is important. We now analyze the results based on the ranking of the generated scenarios. Specifically, we consider how dropping the lower 25% and lower 50% of the ranked scenarios affect the results, especially the noisy scenario.

First, Table VI shows the results from dropping the lower 25% of the ranked scenarios. Due to space, we only show the aggregate for Application 2. The values in parentheses indicate the original values, prior to reduction. Cells without parentheses indicate that the new value is the same as the original value. "U" stands for useful and "N" stands for noise. Note that the lower 25% were dropped for each of the Web page, so the aggregate of all four pages actually has more than 25% dropped.

TABLE VI. TOP 75% SCENARIOS

		TO	EO	Both in EO and TO	In EO, not in TO	In TO, not in EO	
App 1	Regist.	18 (24)	15	14	1	4 (10)	U = 3 (4)
							N = 1 (6)
	Log In	9 (13)	4	4	0	5 (9)	U = 3
							N = 2 (6)
	Search	6 (8)	3	3	0	3 (5)	U = 0
							N = 3 (5)
	New Content	9 (13)	8	4 (7)	4 (1)	5 (6)	U = 5 (6)
							N = 0
	Total	42 (58)	30	25 (28)	5 (2)	17 (30)	U = 11 (13)
							N = 6 (17)
Total (App2)		58 (81)	44	28 (32)	16 (12)	30 (49)	U = 24 (29)
							N = 6 (20)

The results show that removal of the lower 25% of the generated scenarios effectively removes noise (68% = (11+14)/(17+20)). Furthermore, 16% (= (2+5)/(13+29)) of the useful scenarios were also dropped as well as 12% (= (3+4)/(28+32)) of the scenarios that were both in the tool output and the engineer's scenario set.

To further see the effect of ranked reduction of test scenarios, we next dropped the lower 50% of the ranked scenarios. Table VII shows the results.

TABLE VII. TOP 50% SCENARIOS

		TO	EO	Both in EO and TO	In EO, not in TO	In TO, not in EO	
App 1	Regist.	12 (24)	15	10 (14)	5 (1)	2 (10)	U = 1 (4)
							N = 1 (6)
	Log In	6 (13)	4	4	0	2 (9)	U = 1 (3)
							N = 1 (6)
	Search	4 (8)	3	3	0	1 (5)	U = 0
							N = 1 (5)
	New Content	6 (13)	8	2 (7)	6 (1)	4 (6)	U = 4 (6)
							N = 0
	Total	28 (58)	30	19 (28)	11 (2)	9 (30)	U = 6 (13)
							N = 3 (17)
Total (App2)		40 (81)	44	24 (32)	20 (12)	16 (49)	U = 11 (29)
							N = 5 (20)

When the lower 50% is removed, 78% (= (14+15)/(17+20)) of the noisy scenarios are removed. However, there is also a significant drop in the useful scenarios with a drop of 60% (= (7+18)/(13+29)).

The results indicate that noisy scenarios appear far more often in the lower rankings. When reducing the number of test scenarios, there is a clear trade-off between reducing the noise and retaining the useful scenarios.

V. CONCLUSION AND FUTURE WORK

We proposed a novel knowledge-based approach to generate test scenarios for Web applications. Our approach is based on a database containing the test scenarios from previous software tests. The tester creates a base scenario, and then searches for related scenarios in the database.

We conducted a case study on two Web applications, comparing the results of manually created test scenarios by professional software testers and test scenarios generated by our tool based on base scenarios created by students. We found that (1) our tool can produce more scenarios more quickly, (2) our tool can generate scenarios that the professional did not create, (3) noisy scenarios were also generated but they tended to be due to dictionary (synonym) issues and inconsideration of input type, and (4) ranking information can be used to reduce the number of test scenarios, and effectively reduce noise up to a certain point.

Future work includes the following. First, we need to conduct a controlled experiment, as our evaluation was based on a case study. Many factors could have affected the result, especially the contents of the database. Second, we need to consider how we can better build a dictionary, as well as other techniques for determining synonyms. Finally, we need a way to identify noisy scenarios.

VI. REFERENCES

[1] D. Xu, H. Li, and C. P. Lam, "Using adaptive agents to automatically generate test scenarios from the UML activity diagrams," Proc. of APSEC '05, pp. 385-392, 2005.

[2] A. Andrews, J. Offut, and R. Alexander, "Testing Web applications by modeling with FSMs," Software & Systems Modeling, vol.4, no.3, pp.326-245, 2005.

[3] S. Fujiwara, K. Munakata, Y. Maeda, A. Katayama, and T. Uehara, "Test data generation for web application using a UML class diagram with OCL constraints", Journal of Innovations in Systems and Software Engineering, vol. 7, no. 4, pp. 275–282, 2011.

[4] S. Artzi, A. Kiezun, J. Dolby, F. Tip, D. Dig, A. Paradkar, and M. Ernst, "Finding Bugs in Web Applications Using Dynamic Test Generation and Explicit-State Model Checking," IEEE Trans. on Software Engineering, vol.36, no.4, pp.474-494, 2010.

[5] W. Halfond and A. Orso, "Improving test case generation for web applications using automated interface discovery", Proc. of ESEC-FSE '07, pp.145-154, 2007.

[6] C. Pacheco and M. Ernst, "Randoop: Feedback-directed Random Testing for Java", Proc. of OOPSLA 2007 Companion, 2007.

[7] F. Gross, G. Fraser, and A. Zeller, "Search-based system testing: high coverage, no false alarms", Proc. of ISSTA 2012, pp.67-77, 2012.

[8] C. Kaner, "Cem Kaner on Scenario Testing: The Power of 'What-If...' and Nine Ways to Fuel Your Imagination", Software Testing and Quality Engineering Magazine, vol.5, no.5, pp.16-22, 2003.

[9] UMLet, http://www.umlet.com [accessed Feb. 20, 2013].

[10] Tapestry5 Hotel Booking, http://tapestry.zones.apache.org:8180/tapestry5-hotel-booking/ [accessed Feb. 20, 2013].

[11] OLAT, http://www.olat.org/ [accessed Feb. 20, 2013].

Improving Usability Inspection Technologies for Web Mockups through Empirical Studies

Luis Rivero and Tayana Conte
Instituto de Computação, Universidade Federal do Amazonas (UFAM)
Manaus, AM - Brazil
{luisrivero,tayana}@icomp.ufam.edu.br

***Abstract*— Usability is the absence of obstacles that stop the user from carrying out his/her tasks in the system with efficiency, effectiveness and satisfaction. In Web applications, usability is crucial because it determines their acceptance. Researchers have proposed many Usability Inspection Methods (UIMs) to improve the usability of Web applications. However, we identified that these UIMs are mostly applied in later stages of the development process, which can increase the cost of correcting the identified usability problems. Correcting problems in later stages can be costly since most of the source code will have already been written. To improve the quality of Web applications by ensuring usability in earlier stages of their development process, we proposed the Web Design Usability Evaluation technique and its tool support, which allow the evaluation of low fidelity prototypes or mockups. In this paper, we present the results from empirical studies evaluating these technologies. The quantitative and qualitative results provided indicators of their feasibility and also allowed us to identify improvement opportunities. We have further analyzed and then integrated the identified suggestions to generate newer and improved versions of both the technique and its tool support; and to develop a new tool suitable for novice inspectors.**

Keywords-Usability Inspection Method; Tool Support; Web Application; Mockups; Empirical Studies

I. INTRODUCTION

Over the past years, our dependence and reliance on Web applications have significantly increased [1]. Furthermore, in this type of applications, usability becomes a key factor since it determines their acceptance and improves their quality [2].

In order to evaluate the usability of Web applications, it is necessary to verify if the user interface is friendly, direct and easy to understand [3]. Users expect Web applications to allow them to carry out their tasks with accuracy, within short time and using few resources. It is possible to verify all these features by carrying out usability inspections.

Usability Inspection Methods (UIMs) are procedures in which inspectors verify the system's level of achievement of usability attributes. The obtained results can be used to predict whether there will be a usability problem. Since the cost of performing usability inspections is lower than other types of usability evaluation methods (such as user testing), the software development industry has invested in the proposal and improvement of a variety of UIMs for Web applications to address Web usability issues [4].

In our previous work [3], we identified that despite this increasing number of UIMs for the Web, few of them can be applied in earlier stages of the development process. Finding usability related problems in later stages can lead in an increase in time and cost, since their correction might require significant changes in the source code of the already developed application [5]. Therefore, there is a need for UIMs able to find problems during the design phases of Web applications. Furthermore, our results also showed that only a few of the usability inspection techniques provided tool support, and that most of the proposed tools were not able to support the identification of all types of usability problems. These results indicated the need for usability inspection tools able to support inspectors during the application of an UIM.

To meet these needs, we proposed the Web Design Usability Evaluation (Web DUE) technique [3] and its tool support, the Mockup Design Usability Evaluation (Mockup DUE) tool. The Web DUE technique allows inspectors to identify usability problems in earlier stages of the development process by evaluating Web low fidelity prototypes or mockups. A mockup is a model that shows images of how the software would look like after its implementation. The Mockup DUE tool assists inspectors using the Web DUE technique by allowing them to: (a) interact with mockups as if they were a real application, and (b) use the Web DUE technique to find usability problems.

In this paper, we analyze and discuss the results of two empirical studies in order to evolve the Web DUE technique and the Mockup DUE tool. First we summarize the quantitative and qualitative results that show indicators of the feasibility of the proposed technologies. Then, analyzing and integrating the suggestions from the inspectors who participated in the studies, we created newer and improved versions of the technologies. Some modifications include: (a) rewriting instructions, (b) thoroughly describing usability verification items, and (c) modifying the overall layout of the tool. Finally, in order to assist novice inspectors using the Web DUE technique, we developed a new tool to guide them through the overall inspection process of Web mockups.

This paper is organized as follows. Section II presents the background to Usability Inspection Methods while Section III shows the proposal of the Web DUE technique and the Mockup DUE tool. In Section IV we summarize the empirical studies and their overall results. Section V shows how we used the results from the empirical studies to create the new

versions of the proposed technologies. In Section VI, we present the Mockup Guiding Design Usability Evaluation tool for novice inspectors in the Web DUE technique. Finally, Section VII presents our conclusions and future work.

II. Related Work on UIMs for the Web

Usability Evaluation Methods (UEMs) are procedures composed by a set of well-defined activities that are used to evaluate the system's usability [4]. Usability Inspection Methods (UIMs) are a subset of these UEMs, in which inspectors review the usability aspects of the software artifacts. The main advantage of UIMs is that they can lower the cost of finding usability problems since they require fewer resources. A UIM does not need any special equipment or laboratory to be performed.

The most commonly used UIMs are the Heuristic Evaluation (HE) [6] and the Cognitive Walkthrough (CW) [7]. The HE assists inspectors in finding usability problems through the usage of a set of rules which seek to describe common properties of usable interfaces. The CW, on the other hand, allows inspectors to analyze if a user can make sense of interaction steps as they proceed in a pre-defined task.

Regarding UIMs for the Web, in our previous research [3] we identified that there has been an effort in evaluating the usability of Web applications to improve their quality. There have been several proposals of UIMs that are specific for the inspection of Web applications. These UIMs are adapting generic techniques like the HE and the CW to the Web domain. Furthermore, some of the techniques are creating their own specific set of rules for the Web domain.

We also identified that around 77% of the reviewed papers reported UIMs analyzing completely developed Web applications or at least functional prototypes [3]. Moreover, the remaining papers described automated techniques in which HTML code was verified, or in which UIMs evaluated if the application model met interaction rules within the Web domain. These results showed that despite the proposal of new UIMs for the Web, these UIMs are mostly applied to find problems in finished applications or versions that are previous to their release.

Finally, besides not being used in the last stages of the development process; most of the proposed UIMs for the Web did not provide any tool support [3]. Furthermore, the automated tools for usability inspection of Web applications could only be used to point usability issues regarding colors and patterns. These results show that there is also a need for tool support in order to enhance the results of the inspection of Web applications, and that such tools should be able to assist the inspectors in finding problems related to the overall design of their interaction.

III. A Set of Technologies for Usability Inspection of Web Mockups

In [3], we proposed the Web Design Usability Evaluation (Web DUE) technique and the Mockup Design Usability Evaluation (Mockup DUE) tool to meet the actual needs of the software development industry regarding UIMs for the Web.

A. *The Web Design Usability Evaluation Technique*

The Web DUE technique allows the identification of usability problems in early stages of the development process. Consequently, it allows the inspection of Web mockups, which are software prototypes that can be created earlier in the development. Furthermore, mockups cost less than other types of prototypes and, software engineers can use them to validate the software product with its end users [8].

To assists inspectors in finding more usability problems of Web applications, the Web DUE technique uses the concept of Web page zones [9] to guide inspectors. Web page zones are pieces of Web pages with specific types of contents. For each Web page zone, the Web DUE technique suggests a set usability verification items that must be checked to verify if the Web application meets usability principals. These items are based on the Web Design Perspective (WDP) based usability evaluation technique [10]. The WDP technique evolves the Heuristic Evaluation for the usability inspection of Web applications by using the Navigation, Concept and Presentation perspectives. Table I shows some of the usability verification items for one of the Web page zones. Interested readers can find the complete list of Web page zones and their usability verification items at our previous work [3].

In order to carry out a usability inspection using the Web DUE technique, we must: (a) divide the mockups in Web page zones; (b) verify if the mockups adhere to the checklists for the Web page zones they contain; and (c) point any nonconformities in the mockup as a usability problem. Figure 1 shows a brief example of the inspection process of the Web DUE technique over a mockup from the SiON Web application[1] that provides indicators of science and technology in Brazil. We have divided the mockup in the following zones: (1) Institution, (2) Navigation, (3) Information, (4) Direct Access, (5) Services and (6) Data Entry.

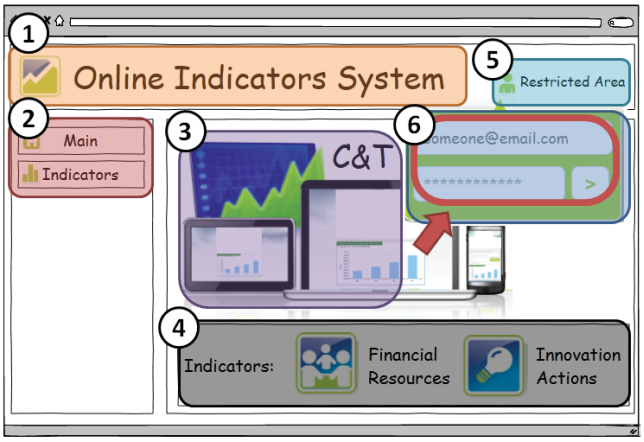


Figure 1. *A mockup being evaluated with the Web DUE technique.*

Table I also shows some of the usability verification items we have evaluated for the data entry zone within this mockup (See Figure 1 Element 6). First, Item A (indicating the correct format of the data) has been respected, since the mockup

[1] http://sion.secti.am.gov.br/

shows examples of how the user must enter the data. Moreover, Item B (ability to understand terms) could cause a usability problem because: (a) the interface does not show the labels of the requested data, and (b) the symbol used to confirm the action is not very clear. Finally, Item C (indicating mandatory data) is a usability problem, since there is no information about which of the fields are mandatory.

TABLE I. DATA ENTRY ZONE: DESCRIPTION, USABILITY VERIFICATION ITEMS AND EXAMPLES / EXPLANATIONS.

Data Entry Zone		
Description: This zone is responsible for providing the user with a form to input data in order to execute certain operations. Then, a submit-like button links the input data with the associated functionality.		
ID	**Usability Verification Items**	**Examples / Explanations**
A	The interface indicates the correct format for a determined data entrance.	The data fields must contain/provide hints on how to fill them. For example, a "Date" entry field could have the next hint: "mm/dd/yy".
B	It is possible to understand the meaning of the terms (words and symbols) that are being used.	The user must be able to identify which data are being requested without difficulty. For instance, a date field should not have ambiguous names, such as "time", "age", among others.
C	The interface indicates which data must be mandatory filled.	For example, the system indicates mandatory input data with a "*" or a "mandatory" next to the field.

In order to perform a complete inspection, all the mockups within the Web application must be evaluated. However, inspectors will only need to check the usability verification items for the zones that compose the evaluated mockup.

B. *The Mockup Design Usability Evaluation Tool*

In [3], we also identified that emerging UIMs for the Web should provide tool support so that they could enhance the results and performance of the inspection process. Therefore, we developed the Mockup DUE tool which provides assistance in two different stages: (a) inspection planning, and (b) detection process. During the planning stage, the moderator of the inspection, who prepares the materials for an inspection, loads a set of mockups and maps them to simulate interaction. For instance, Figure 2 shows a print screen in which the moderator adds a link in a mockup and then, by clicking on it, he is automatically taken to the destination mockup of that link.

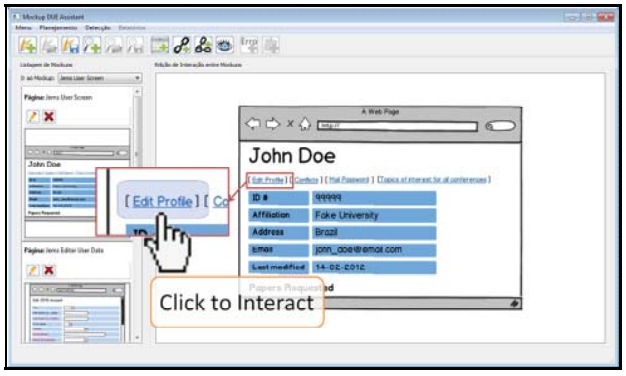


Figure 2. *The Mockup DUE tool being used to interact with a mockup.*

The Mockup DUE tool can also be used to identify usability problems. In this stage, the inspector, who identifies problems, inputs his/her name and opens a file containing the previously mapped mockups. Then, the tool shows the Web DUE technique, describing the Web page zones and their usability verification items. The inspectors check if the mockups adhere to the usability principles from the technique, and if they detect a problem they can point it in the mockup or add suggestions by adding notes. Furthermore, in order to verify if the interaction model is adequate, the inspectors can use the previously added links to simulate and evaluate the provided interaction.

IV. EVALUATION THROUGH EMPIRICAL STUDIES

A. *1st Empirical Study: Feasibility of the Web DUE Technique*

This empirical study aimed at answering the following research question: "*Is the Web DUE technique feasible regarding the number of detected defects and is time well spent?*" It is noteworthy that the results from this study have been published in [11] and interested readers in its description and evaluation artifacts can refer to that paper for further information.

1) *Description*

We compared the Web DUE technique with its predecessor, the WDP technique analyzing two indicators: (a) effectiveness, which is the ratio between the number of detected problems and the total of existing problems, and (b) efficiency, which is the ratio between the number of detected problems and the time spent in finding them. These indicators have also been used in other empirical studies [1][11].

There were a total of eight subjects who agreed to participate in this study. All subjects answered objective questions regarding their degree of knowledge and professional experience in Human Computer Interaction (HCI), Usability Inspections and Design. Then, we classified them as having Low, Medium or High experience according to the information they provided. For instance, regarding their experience in HCI, we considered: (a) Low, if the subject had no practical experience in HCI nor had studied HCI; (b) Medium, if the subject had studied HCI, but had poor practical experience in usability evaluations; and (c) High, if the subject had studied HCI in books and class, and had participated in projects involving usability evaluation.

After classifying the subjects, each of them was assigned a technique, either the Web DUE or the WDP technique (see the classification in Table III). We balanced each group according to the experience of the subjects to avoid bias. Then each group received training in the assigned technique and carried out a usability inspection over a set of mockups based on a Coupon Website. It is noteworthy that the inspectors manually simulated the interaction among the mockups, and manually pointed errors and notes since the Mockup DUE tool was not yet available.

After finishing the inspection, each subject delivered an inspection report containing discrepancies describing possible usability problems. We combined the discrepancies in a

unique report without repetitions. Finally, this report was analyzed by experienced inspectors, outside the study, who classified the discrepancies as false positives or real defects.

2) Quantitative and Qualitative Results

Table II shows the overall quantitative results from this study which suggest that the Web DUE technique was more effective and efficient than the WDP technique. However, due to the small sample used, our results can only be considered indicators of the feasibility of the Web DUE.

TABLE II. QUANTITATIVE RESULTS PER INSPECTOR AND TECHNIQUE.

	Group							
	Web DUE				WDP			
Subjects	1	2	3	4	5	6	7	8
HCI Exp.	M	L	L	H	L	L	H	M
Inspections Exp.	L	L	L	L	L	L	L	H
Design Exp.	L	M	L	H	L	M	M	L
Time Spent (min)	100	67	200	160	147	132	124	279
Discrepancies	38	23	27	22	12	33	9	30
False Positives	19	2	5	4	1	5	0	8
Real Defects	19	21	22	18	11	28	9	22
Effectiveness (%)	24.1	26.6	27.9	22.8	14.0	35.4	11.4	27.9
Efficiency (Defects per Hour)	11.4	18.8	6.6	6.8	4.5	12.7	4.4	4.7
Av. Effectiveness	25.32				22.15			
Av. Efficiency	10.89				6.58			

To better understand the obtained results, we asked the inspectors' opinion towards the use of the Web DUE technique. All subjects answered a questionnaire aiming to identify the difficulties and advantages of using the technique. This is the list of our findings which are also available at [11]:

- Some inspectors indicated that there were items that were too broad, too ambiguous or not clear enough to assist in the identification of usability problems.
- All inspectors agreed that it was easy to understand and identify all Web page zones within the mockups.
- All inspectors agreed that the technique could aid in finding usability problems in Web application mockups.
- The inspectors agreed that the usability verification items were helpful when combined with the Web page zones.
- The inspectors stated that the inspection process becomes very tiring because it is necessary to manually simulate the interaction between the user and the system.

B. 2nd Empirical Study: Evaluation of the Mockup DUE Tool

In this second empirical study, we aimed at answering the following research question: "*Is the Mockup DUE tool easy to use and does it satisfies its users?*" It is noteworthy that the results from this study have also been published in [11] and interested readers in its description and evaluation artifacts can refer to that paper for further information.

1) Description

We performed a cooperative evaluation to obtain qualitative data about the ease of use and the degree of user satisfaction of the tool. During this evaluation, experienced inspectors, both in HCI and usability inspections, used the tool and evaluated it to identify usability issues and their solutions.

There were a total of four inspectors who agreed to participate. These inspectors had high knowledge and practical experience in usability evaluations. During the cooperative evaluation, all subjects performed two activities using the tool: (a) map mockups, and (b) identify usability problems. While carrying out the activities, the inspectors would think out loud their opinions regarding the tool and anything they thought made the inspection process difficult. Finally, all inspectors answered a questionnaire aiming to identify the difficulties and advantages of using the tool.

To ensure the certainty of the data, we recorded all of the cooperative evaluations. We have used the recordings to identify improvement opportunities in the design of the tool.

2) Qualitative Results

The inspectors stated that they were pleased using the tool. However, they also pointed out that the tool's design could be improved a lot. We used the answers to the questionnaire and the recordings of the cooperative evaluation to identify the reason for these opinions. This is the list of our findings:

- All inspectors would use the tool to carry out a usability inspection of Web mockups.
- The opinions were divided, some inspectors liked it very much and others thought the tool needed improvement.
- Some inspectors thought it was necessary to provide more information and hints on how to use the tool.
- The layout had a direct effect over how easy it was to execute the functionalities. Furthermore, it was necessary to rename some buttons and relocate some elements.

V. IMPROVING THE TECHNOLOGIES

A. The Web DUE Technique v2.0

If we look at Table II it is possible to see that the number of false positives from the Web DUE technique was much higher than the number of false positives from the WDP technique. This meant that, despite allowing the identification of more usability problems in lesser time, the Web DUE technique led inspectors to point discrepancies that would not affect the usability of the evaluated Web application. Furthermore, the inspectors corroborated this statement within the answers to their questionnaires (see example quote below):

"… However, some verification items judge features, which are not totally wrong or right." - Inspector 3.

It is necessary to improve the accuracy of the Web DUE technique by reducing the number of false positives. This can offer inspectors a higher degree of certainty that the usability evaluation will help find real usability problems that, if corrected, can improve the quality of the Web application.

We analyzed the descriptions that the inspectors wrote for each false positive. Table III shows an example of three false positives and their corresponding description that motivated us

to make changes in the Web DUE technique. We will now relate the changes to the false positives in Table III.

TABLE III. FALSE POSITIVES AND THEIR DESCRIPTION.

ID	Usability Verification Item	Description of the Problem
1	There is a data entry zone in the system.	*There is no such zone.*
2	The user interface makes it easy to differentiate between different data.	*There is no different data.*
3	The system offers a navigation menu.	*I don't see a menu.*

We realized that asking the inspectors to point any nonconformity with the usability verification items made them identify unrelated errors. For instance, an inspector marked Item 1 even though providing a data entry zone depends on the functionality of the application. Therefore, we changed an instruction from the training as shown in Figure 3.

Web DUE v1.0	Web DUE v2.0
If the mockup does not adhere to the verification item, a new usability error was found.	If the mockup does not adhere to the verification item, **evaluate if such nonconformity could cause a usability problem** in the Web application.

Figure 3. *Changes in the instructions of the Web DUE technique v2.0.*

We identified that the inspectors were using the usability verification items as rules and not as guidelines for the inspection. For instance, Item 2 was pointed because the inspector thought that providing different data was mandatory. This type of false positive allowed us to realize that there are some items that should only be evaluated if a precondition is met. Consequently, we renamed this type of items and added a new instruction to fix this problem (see example in Figure 4).

New Instruction for the Web DUE v2.0	
Evaluate if the usability verification item applies to the evaluated mockup (**if the necessary preconditions are being respected by the mockup**).	
Web DUE v1.0	**Web DUE v2.0**
The interface makes it easy to differentiate between distinct data.	**If distinct data are requested:** Evaluate if the interface makes it easy to differentiate them.

Figure 4. *Adding preconditions in the Web DUE technique v2.0.*

Finally, we also came across usability verification items and examples/explanations that did not offer enough information to aid in the identification of usability problems. For instance, Item 3 was pointed as an error because it was not clear enough. Even though the evaluated application provided accurate navigation options, the inspector thought that since they were not menus, there was a usability problem. Therefore, we renamed inaccurate items to provide more insight and assistant to the inspectors (see Figure 5).

Web DUE v1.0	Web DUE v2.0
The system offers a navigation menu.	The system offers a **set of navigation links**, that can be activated to **move from one page to another**.

Figure 5. *Thoroughly describing items in the Web DUE technique v2.0.*

B. *The Mockup DUE Tool v2.0*

We have improved the Mockup DUE tool by analyzing and incorporating the suggestions from the second study:

- **Use of Databases:** As the Mockup DUE tool used XML files to save the inspection results, we noticed there was a risk of losing information if one of the mockups was accidentally deleted. Therefore, we integrated SQL to the tool, allowing it to read all data (mockups, links, notes and defects) from a single file. This also allows the tool to save and load the planning and detection activities at any time.
- **System State:** In order to inform users of the background actions of the tool, we added more labels and tooltips.
- **Renaming Misleading Elements:** Some buttons and labels confused the inspectors. For instance, the "edit" button in a mockup suggested that the inspectors could edit their mockups instead of editing the interaction. Therefore, we renamed all the confusing labels, buttons, and any other similar interaction elements. For instance, we renamed the previously mentioned "edit" button as "edit mockup mapping".
- **Confirmation:** Sometimes, the tool performed actions that the user could regret, such as deleting a note or a pointed defect. To avoid this problem, the Mockup DUE v2.0 always asks for the confirmation of undoable actions.
- **Report:** The first version of the Mockup DUE only generated text files with the data of the inspection. Now, inspectors can create a report using the Mockup DUE tool. Figure 6 shows an example of this report which contains: (a) the name of the inspector; (b) the time spent during the inspection; and (c) a list containing all the errors and notes within the evaluated mockups and their exact location.

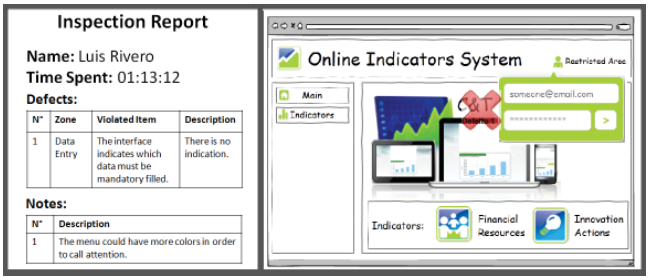


Figure 6. *Mockup DUE tool v2.0 – Generating inspection report.*

Figure 7 shows the current version of the Mockup DUE tool. In this figure, for instance, Element 1 shows how the tool informs the current state in the inspection process, while Element 2 shows how the tool asks for confirmation when deleting a defect. Furthermore, since using different UIMs in the evaluation of an application can improve the performance of the inspection [3], we have integrated the Heuristic Evaluation [6] to the tool. Consequently, inspectors using the tool can now perform an inspection using two different UIMs.

VI. AN INSPECTION TOOL FOR NOVICE INSPECTORS

Our results from the second empirical study showed that despite being an expert in usability inspections, an inspector without previous experience in the Web DUE technique could find the tool difficult to use. Therefore, we made adjustments in the tool to guide inspectors through the overall inspection process of the mockups using the Web DUE technique.

Figure 8 shows a new version of the Mockup DUE tool, the Mockup Guiding Design Usability Evaluation (Mockup G-DUE) tool. In this new tool we made changes in the detection process, particularly replacing the part in Figure 7 Element 3. In this new version, the novice inspectors are asked to indicate which zones are present in the mockup. Then, the tool asks them if there is any nonconformity with the usability verification items. If the inspector indicates that an item has not been respected (by leaving it unmarked), the tool asks him/her if he/she considers it a usability problem. If it is considered a problem, the tool asks for its description. Finally, the problem is added to the mockup and the inspector can locate it in the desired area as seen in Figure 7 Element 4.

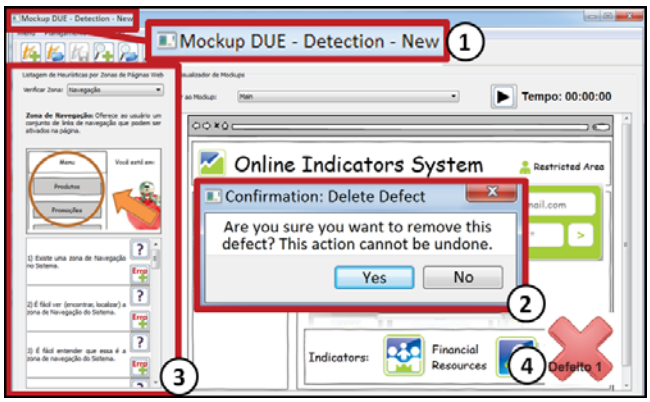


Figure 7. *Mockup DUE tool v2.0 – Print Screen.*

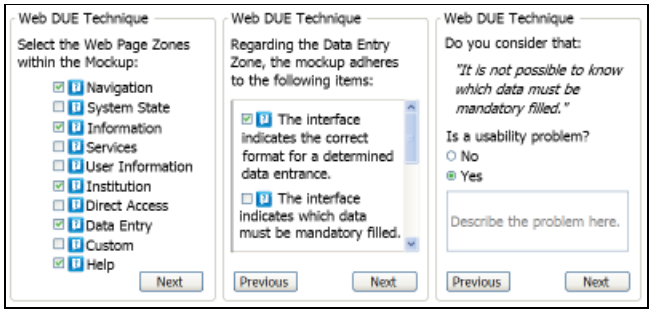


Figure 8. *Changes in the proposal of the new Mockup G-DUE tool.*

VII. Conclusions and Future Work

With this research we were able to develop a new set of technologies for the inspection of Web application mockups. Our results from the empirical studies indicated that the Web DUE technique was more effective and efficient than its predecessor. Furthermore, the qualitative data showed that inspectors found the technique and the Mockup DUE tool useful and appropriate for the inspection of mockups.

The qualitative data also suggested improvements in the technologies. Therefore, we evolved the Web DUE and the Mockup DUE by: (a) rewriting the instructions and usability verification items from the Web DUE technique to make it clearer and more understandable; (b) incorporating further functionalities in the Mockup DUE tool like the creation of reports and support for other techniques; (c) enhancing the usability of the tool by adding labels and information; and (d) proposing an enhanced tool, the Mockup G-DUE, that could assist novice inspectors in the evaluation of mockups of Web applications when using the Web DUE technique.

We are currently carrying out a new empirical study to evaluate how inspectors use the Mockup DUE tool to perform an inspection with the Web DUE technique. In this study we have also increased the number of subjects to augment the statistical power of the results and increase the degree of certainty about the performance of the technologies. Finally, we will also evaluate the Mockup G-DUE to verify its feasibility, and we will carry out an empirical study in an industrial environment to observe any obstacles in adopting the proposed technologies in the software industry.

Acknowledgments

We would like to acknowledge the financial support granted by FAPEAM process PAPE 055/2013. We would also like to thank CNPq and SUFRAMA for the provided scholarship to the first author of this paper.

References

[1] S. Murugesan, "Web Application Development: Challenges and the Role of Web Engineering," In G. Rossi, D. Schwabe, L. Olsina, and O. Pastor, "Web Engineering: Modeling and Implementing Web Applications," Springer, 2008.

[2] M. Matera, F. Rizzo, and G. Carughi, "Web Usability: Principles and Evaluation Methods," In: E. Mendes and N. Mosley, "Web Engineering," Springer, 2006.

[3] L. Rivero, R. Barreto, and T. Conte, "Characterizing Usability Inspection Methods through the Analysis of a Systematic Mapping Study Extension," Latin-american Center for Informatics Studies Electronic Journal, Volume 16, Issue 1, 2013.

[4] A. Fernandez, E. Insfran, and S. Abrahao, "Usability evaluation methods for the Web: A systematic mapping study," Information and Software Technology, Volume 53, Issue 8, 2011.

[5] G. Travassos, F. Shull, M. Fredericks, and V. Basili, "Detecting defects in object-oriented designs: using reading techniques to increase software quality," Proc. 14th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, USA, 1999.

[6] J. Nielsen, "Finding usability problems through heuristic evaluation," Proc. CHI'92, UK, 1992, pp. 373-380.

[7] P. Polson, C. Lewis, J. Rieman, and C. Wharton, "Cognitive walk-throughs: a method for theory-based evaluation of user interfaces," International Journal of Man-Machine Studies, Volume 36, Issue 5, 1992.

[8] F. Ricca, G. Scanniello, M. Torchiano, G. Reggio, and E. Astesiano, "On the effort of augmenting use cases with screen mockups: results from a preliminary empirical study," Proc. 4th ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, Italy, 2010.

[9] J. Fons, V. Pelechano, O. Pastor, P. Valderas, and V. Torres, "Applying the OOWS model-driven approach for developing Web applications: The internet movie database case study," In G. Rossi, D. Schwabe, L. Olsina, and O. Pastor, "Web Engineering: Modeling and Implementing Web Applications," Springer, 2008.

[10] T. Conte, J. Massollar, E. Mendes, and G. Travassos, "Web usability inspection technique based on design perspectives," IET Software, Volume 3, Issue 2, 2009.

[11] L. Rivero, and T. Conte, "Using an Empirical Study to Evaluate the Feasibility of a New Usability Inspection Technique for Paper Based Prototypes of Web Applications," Journal of Software Engineering Research and Development, 2013. (*Submitted for Evaluation*)

[12] A. Fernandez, S. Abrahao, and E. Insfran, "Towards to the validation of a usability evaluation method for model-driven Web development," Proc. IV International Symposium on Empirical Software Engineering and Measurement, USA, 2010, pp 54-57.

Random Visual GUI Testing: Proof of Concept

Emil Alégroth
Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg, Sweden
Emil.Alegroth@Chalmers.se

***Abstract*—Market demands for higher quality software and shorter time-to-market delivery have resulted in a need for new automated software testing techniques. Most automated testing techniques are designed for regression testing that limit their fault finding ability to faults explicitly tested in scenarios/scripts. To overcome this limitation, companies define test processes with several test techniques, e.g. unit testing and random testing (RT). RT is a technique that can be performed manually or automatically with tools such as Fuzz, DART and Quickcheck. However, these tools operate on lower levels of system abstraction, leaving a gap for a Graphical User Interface (GUI), bitmap level, automated RT technique.**

In this paper we present proof of concept for Random Visual GUI testing (RVGT), a novel automated test technique that combines GUI based testing, Visual GUI Testing, with random testing. Proof of concept for RVGT is evaluated in a three phase study with results that show that RVGT is applicable for both functional and non-functional/quality requirement conformance testing. Furthermore, results from a survey performed in industry indicate that there is industrial need for the technique. These pivotal results show that further research into RVGT is warranted.

***Keywords*-Visual GUI testing; Random testing; Proof of Concept**

I. Introduction

The demands for higher software quality and faster time to market delivery are continuously increasing in software industry. These demands force software development companies to focus a larger percentage of time on development, leaving less time for software quality assurance. Quality assurance that can constitute more than 40 percent of the development cost of a system [1]. To aid companies to reach higher quality and lower development time a plethora of automated testing techniques have been developed, e.g. unit testing [2], record and replay [3], and Visual GUI Testing [4], [5]. These automated testing techniques are used for regression testing, i.e. to ensure that the system under test (SUT) still conforms to the system requirements after change or maintenance of the system [6].

Most test tools focus on functional requirement conformance testing but there are also tools that can test a system's non-functional/quality requirement (NFR) conformance. However, most of these tools focus on performance, availability and other quantifiable types of NFRs, whilst NFRs like usability and user experience are left without support. This is one reason why automated testing is not considered a replacement to manual testing. Another reason is because most automated testing techniques are only able to find faults, functional or NFR related, which have been specified in the scripted test scenarios. Therefore, a common industrial practice is to complement the automated scenario-based testing with manual test techniques such as random or exploratory testing. Both these techniques are generally performed through interaction with the SUT's GUI but with the important distinction that with exploratory testing the practitioner aims at identifying the cause of a fault, not just that there is a fault, which is the case with random testing [7], [8]. However, in contrast to exploratory testing, random testing can be performed automatically with tools such as Direct Automated Random Testing (DART) [9], Fuzz [10] and Quickcheck [11]. Automated random testing is however generally performed on lower levels of system abstraction. A gap therefore exists for a high-level technique that can perform user emulated random testing through the SUT's GUI.

In our previous work we have evaluated an automated test technique, referred to as Visual GUI Testing (VGT) [4], [5]. VGT uses image recognition to interact with a SUT through its GUI bitmap layer, i.e. what is shown to the human user on the computer monitor. The image recognition is what differentiates VGT from previous GUI based test techniques, e.g. coordinate- or component/widget-based record and replay [3]. These techniques require access to the SUT's components, underlying APIs or source code, which limits their applicability dependent on SUT implementation. In contrast, VGT is more flexible since the image recognition makes it non-intrusive and independent of SUT implementation, operating system or even platform, e.g. desktop, mobile or cloud. Furthermore, the image recognition allows the technique to emulate human user behavior since all interactions with the SUT are performed through, and with, the same interfaces a human uses, i.e. GUI bitmaps and the operating system's mouse and keyboard operations. Therefore, VGT also has the potential to perform automated GUI bitmap based random testing, which in the continuation of this paper will be referred to as Random Visual GUI Testing (RVGT). However, to the author's best knowledge, no research has been conducted on RVGT, either for conformance testing of functional or non-function/quality requirements.

In this paper we aim to bridge this gap in research with results from a three phase study with the goal of providing proof of concept that VGT is applicable for automated random functional requirement and NFR conformance testing. Furthermore the study will present a survey performed at the

Swedish safety-critical software development company, Saab AB, which shows that there is a need for further research into RVGT and that it perceivably can have positive impact on industrial testing. Thus, the research questions that this study aims to answer are,

1) RQ1: Can random testing be combined with Visual GUI Testing to perform automated, GUI bitmap based, random testing?
2) RQ2: Can random Visual GUI Testing be used to verify system conformance to non-functional/quality requirements?
3) RQ3: Is there a need for/interest in random Visual GUI Testing in industrial practice?

The continuation of this paper is structured as follows. Section II will present related work regarding random testing, automated random testing and VGT, followed by the research methodology used in this study in Section III. In Section IV the results of the study will be presented. Finally the paper vill be concluded in Section V.

II. Related Work

Random testing (RT) is a technique that is commonly used to complement automated testing in industry. The technique is performed through random generation of, and/or random execution, of test cases with the overall aim to cover the input space of a system. Thus, providing test coverage of both common and uncommon cases that appear during system usage and which may be faulty. Furthermore, the technique can be performed manually, e.g. through random interaction with the system under test (SUT) to force it into a faulty state, or automatically by using tools, e.g. Direct Automated Random Testing (DART) [9], Fuzz [10] and Quickcheck [11]. In addition, even though RT is based on random execution of, often mutually exclusive, SUT interactions, studies have shown that RT has equal or even higher fault finding ability than structured test techniques because of higher input coverage [11], [12].

Furthermore, RT allows the user to quantify the significance that a test will not fail and formulate statements like "it is certain that program P will not fail more than once in 10.000 calculations". The conventional theory behind such a statement comes from the equation,

$$\frac{1}{\phi} = \frac{1}{1-(1-e)^{1/N}}$$

where ϕ is the failure rate, $1/\phi$ is the Mean Time To Failure (MTTF), e is the probability that one failure will be observed and N the number of test runs [8]. Hence, the number of tests required to acquire a confidence of 1-e for a given MTTF is,

$$\frac{log(1-e)}{log(1-\phi)}$$

This value can for instance be used as a quality metric of the tested software. However, as with all techniques, RT has drawbacks including, but not limited to, that it can be difficult to define the input space, observe/evaluate the RT output, develop proper oracles to support RT, etc.

Another technique similar to RT that is often used in industry is exploratory testing (ET), defined as *simultaneous learning, test design, and test execution* [13]. ET is based on random input to identify faults but instead of using mutually exclusive interactions this technique relies on cognitive decision making and the user to use previous test results to narrow in on the cause of a fault. Because of this cognitive element, ET is primarily a manual practice that human testers perform without knowing about it, i.e. if a human finds a fault, he/she naturally tries to find its cause [7]. In summary, ET focuses on depth, i.e. finding the cause of a fault, whilst RT focuses on breath, i.e. finding many faults but necessarily not what causes them.

RT and ET are primarily performed manually on a GUI level because there is a gap in terms of tool-support. However, this gap could potentially be filled by Visual GUI Testing (VGT) [4], a novel test technique that is currently emerging in industry. VGT is a tool supported technique that uses scenario-based scripts and image recognition to interact with the system under test (SUT) through its GUI on a bitmap level, i.e. interaction against what is actually shown to the user on the computer monitor. The image recognition allows VGT to emulate human user behavior and could perceivably therefore be used for automated RT and ET. This hypothesis is supported by previous work into VGT that has shown the technique's industrial applicability for scenario-based system and acceptance test automation, resulting in modular scripts that could be randomized on a test suite level of abstraction [4], [5]. However, to the author's best knowledge, there are no studies that use image recognition to perform automated GUI based RT.

As stated, VGT is a novel automation technique, and as with any new automated test technique it requires verification. One approach that is common to verify test techniques, oracles, etc., is mutation testing. Mutantion testing is conducted through fault injection into the SUT, e.g. by randomly modifying input to, or operations in, the SUT to produce faulty output that the tested technique should be able to capture. Even though these faults are artificially created, studies have shown that these faults are equally difficult, or even more difficult, to identify than industrial grade faults [14]. In this study, mutation testing is used to verify RVGT's ability to find faults related to a system's functional requirements.

III. Methodology

The study presented in this paper consists of three phases. In the first phase, a proof of concept study was performed to evaluate VGT's applicability for random testing of functional requirement conformance. The evaluation was performed on two calculator applications, tested with a RVGT script written in the open source VGT tool, Sikuli [15]. In the second phase, another RVGT script was developed to evaluate VGT's applicability for random testing of non-functional/quality requirement (NFR) conformance. This evaluation was performed with a commercial Swedish bus-travel application to show RVGT's applicability on real-world applications. Both phase

one and two were performed on a computer with an 3.07GHz Intel(R) Core(TM) i7 CPU, 6GB of RAM, two GeForce GTX 460 graphic cards with SLi support and the Windows 7 Professional Operating System. In the third phase of the study a survey was performed in industry, at the company Saab AB, to evaluate the industrial need of RVGT. Thus, this study follows the principles of the circular knowledge transfer model described by Gorschek et al. [16]. The model highlights the importance of research knowledge transfer/development in incremental parts. It also states that laboratory experimentation is a good idea before industrial deployment to catch research problems and thereby save unnecessary costs for industry. Hence, this study aims to be an initial building block for future research into RVGT, such as industrial evaluation of the technique, further technical improvement of random testing algorithms, etc. The continuation of this section will describe each phase of the study in more detail.

A. Phase One: Calculator evaluation

In the first phase of the study a RVGT script was written to test the functional requirement conformance of calculator applications. Thus, this phase was performed with simplistic applications that have limited generalizability. However, this choice of applications was motivated by the exploratory nature of this phase of the study since it was unknown if VGT was at all applicable for GUI based random testing. Therefore, a SUT was required that could receive random input but for which it was possible to construct a dynamic oracle. Hence, an oracle that based on random input values could evaluate an expected output for comparison to the actual output from the SUT. The RVGT script was written using Sikuli [15], an open source VGT tool, which uses Python as a scripting language. Python is an object-oriented programming language that supports all aspect of a conventional programming language such as loops, branching, randomization, etc. However, in Sikuli the Python language has been extended with a set of methods that use the tool's image recognition capabilities. These additional methods allow the user to write scripts that can interact with any bitmap displayed on the computer monitor and through scenarios that exactly emulate human user interaction with the SUT.

The RVGT script was developed using a modularized architecture, consisting of four main parts. The first part of the script contained a set of variables to configure the script's speed, the tool's image recognition sensitivity, etc. Furthermore, this part of the script contained the GUI bitmap components to allow Sikuli's image recognition algorithm to interact with the calculators. The bitmaps were stored in lists which made them simple to change in order to migrate the RVGT script from one calculator to another. The second part of the script contained the script oracle that based on the randomized input numbers, generated in the range of -100.000 to 100.000 and a randomized calculator operation, calculated the expected output. The third part of the script defined the GUI interaction with the SUT, i.e. translated the randomized input numbers into click interactions that Sikuli could perform using the mouse-cursor. Input to the calculators was given as

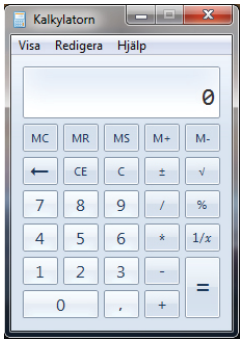


Fig. 1. Windows calculator interface.

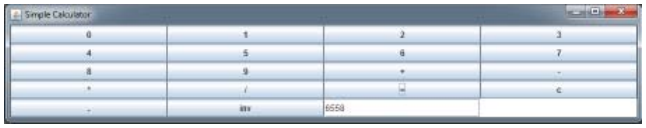


Fig. 2. Simple Java calculator interface

mathematical operations following the pattern,

$$Clearcalculator,$$
$$Input1[0, 100.000], (Optional)Negation,$$
$$Operation[addition, subtraction, multiplication, division],$$
$$Input2[0, 100.000], (Optional)Negation,$$
$$Equals.$$

Hence, if two positive single digit numbers were randomly generated the script would perform five interactions with the SUT whilst two negative six digit random numbers would require 17 interactions. Each mathematical operation was followed by an assertion to verify that the output from the calculator was equal to the expected output. The fourth and final part of the script produced output. Output that consisted of what the randomized numbers were, what mathematical operation that was used, what expected value the oracle had calculated, what the actual value from the calculator was and finally a verdict of the performed assertion.

In order to evaluate the RVGT script it was applied on two calculators, the standard Windows calculator and a custom Java calculator, GUI's shown in Figure 1 and Figure 2. The RVGT script was developed for the Windows calculator and then migrated to the custom Java calculator. To test the RVGT script's capabilities for long-term testing they were executed 10.000 times for each calculator during which the execution time and number of faults were measured. Due to the simplicity of the SUTs, the hypothesis was that there would be no faults in either the Windows or the custom Java calculator. Hence, any fault found by the RVGT script would most likely be a false positive caused either by the image recognition failure or faulty implementation of the RVGT script itself.

Consequently, since the RVGT script, as expected, did not find any faults in the calculators, other than false positives, the

custom Java calculator was modified by introducing mutants into it, for mutantion testing [14]. These mutants caused the calculator to randomly calculate a faulty value in 10 percent of all calculations, also when calculations were performed by a human. In addition, the Java calculator was modified with an additional output module to track which of the calculations were faulty. The RVGT script was then executed against the mutated Java calculator during which 1000 tests were generated. After the execution the result logs from the RVGT script and the calculator were analyzed through visual inspection to determine how many of the randomly infused mutants the RVGT script had been able to kill. The visual inspection was performed by comparing the output logs from the calculator and the RVGT script.

It must however be stressed that a calculator is a quite simplistic system in comparison with most industrial grade systems. However, it is not completely irrelevant to showcase proof of concept since many industrial systems build on basic GUIs with only a few buttons and minimalistic output to the user even though the functionality of the SUT's backend might be quite extensive and/or complex. An example of such a system is the airport management system presented in [4], which had a shallow simplistic GUI, but which controlled safety-critical functionality.

In summary, phase one aimed to provide the proof of concept support for RVGT's applicability to test functional requirement conformance. This was achieved through development of a RVGT script to test calculator applications.

B. Phase Two: Commercial web-application

In phase two, the evaluation of RVGT was expanded also to verification of non-functional/quality requirement (NFR) conformance. Furthermore, to gain a broader view of RVGT's applicability, the evaluation in this phase was performed with a commercial web-service, which in the continuation of this paper will be referred to as the travel planner. The travel planner, screenshot of its GUI shown in Figure 3, is a web-application that allows the user to schedule bus-travel in parts of Sweden. Bus-travel is calculated based on user input that should consist of a start location, an end location and timing information. These input can be given by the user either as text strings or by clicking on a map. Output from the application is presented in a list of different bus travel alternatives, including departure times, if the bus fare will be late, etc. One feature of the travel planner, which is relevant for the evaluation, is that it automatically suggests a list of alternative locations based on the first three letters that the user inputs to the application.

The choice to perform the evaluation on this application was motivated by its representativeness for commercial web-applications and thus of importance/interest for the knowledge transfer to industry, as expressed by Gorschek et al. [16]. However, since access to the actual non-functional requirements of the application could not be acquired during the study, they had to be reverse-engineered based on industrial best practice. Ten NFRs were created for the evaluation, presented in Table I, based on their representativeness for actual NFRs used in industry. Representativeness was evaluated through comparison to industrial NFRs, i.e. best practice, which were available during the study. Furthermore, the NFRs were chosen to cover different types of NFRs, including both qualitative and quantitative requirement types, i.e. Performance, Reliability, Usability, Availability and Portability.

Fig. 3. The traveler planner text input GUI.

Nr	NFR Description	Type
1	The "travel planner" shall provide the user with alternative bus stops within 1 second after the first three letters of a bus stop, or location, has been given as input.	Performance
2	No trips before the current, or user selected, time shall be presented as available.	Reliability
3	After a user clicks the "search-trip" button it shall take maximum 5 seconds before a result is presented.	Performance
4	Late fares shall be presented in a clear way that they are late.	Usability
5	When a user clicks the update button, for a previous search, he/she shall receive new data from the server regarding the trip.	Availability
6	The service shall be available 90 percent of the time.	Availability
7	It shall be possible for a user to provide traveling input in different ways, as textual input or by clicking on a map.	Usability
8	The "travel planner" application shall work in several different browsers.	Portability
9	It shall be possible for a blind user to tab through the entire interface without getting stuck in a "sink-hole".	Usability
10	The "travel planner" shall not accept negative time input.	Usability

TABLE I
THE NFRS THAT WERE TESTED BY THE NFR RVGT SCRIPT.

Once the NFRs had been defined, a RVGT script was developed to test them. Similar to the script in phase one, the travel planner RVGT script was based on a modular architecture with one part for setting up properties of the script, one part that contained the actual tests, one part for output, etc. The largest difference between the scripts developed in phase one and phase two was that randomization in phase two was performed on two different levels of abstraction, i.e. on both an input level but also in what order the tests were executed. Hence, during script execution against the

travel planner, which was performed with 1000 tests, each of the test cases were chosen randomly with a 1/10 probability. Furthermore, all textual and numeric input to the application was randomized. For the textual input the randomization was achieved by randomly inputting three letters to the application and then choosing the top alternative that was suggested by the application. A similar scheme was used for numeric input, e.g. time.

In summary, phase two aimed to identify support for RVGT's applicability for automated random testing of NFR conformance, evaluated on a commercial web-application using constructed, yet realistic, NFRs. However, it should once again be stressed that due to the small number of NFR tests the evaluation only has limited representativeness for industrial NFR testing.

C. Phase Three: Industrial survey

In the third phase of the study, a questionnaire survey was performed at the company Saab AB to evaluate if there is a need for RVGT in practice. The questionnaire was anonymous, voluntary and was handed out to 13 industrial practitioners that were chosen through convenient sampling. Ten people answered the survey, leaving a response rate of 77 percent. The industrial need for RVGT was evaluated with the question, "What effect would the introduction of RVGT perceivably have at Saab?", which was presented as a forced choice question with the three alternative answers,

1) Make current testing worse
2) Not change anything
3) Improve current testing

Included in the questionnaire was a one page summary of RVGT and the proof of concept evaluations performed in phase one and phase two.

IV. Results

In phase one of the evaluation, a RVGT script was developed to randomly test the functionality of two calculators. Results of the first and second part of phase one found no faults in either the Windows calculator or the custom Java calculator that was developed for the evaluation. However, the script still reported that 0.59 percent (less than one percent) of the tests failed with false positive test results. The false positives occurred either because of a mismatch between how the calculator and the oracle rounded fractions, i.e. the precision of the fractions were different which caused the result comparison to fail, or due to image recognition failure. The image recognition failure occurred when an input number with two subsequent nines was generated, e.g. 199 or 991, since the mouse curser changed the GUI state such that the image recognition algorithm could not find the second nine. This problem only appeared for nines and could easily have been solved by replacing the image for the button in the script. However, both these issues show the difficulties of constructing a perfectly functioning oracle, even for simplistic applications.

In the third part of phase one, mutants were introduced into the Java calculator, causing it to produce faulty output 10 percent of the time. Analysis of the output from the calculator showed that 112 mutants were introduced, for the 1000 generated tests, of which 100 percent could be identified and killed by the RVGT script. However, the script reported 114 failures, i.e. 2 false positives. Analysis of these two failures showed that they were caused by the same oracle problem as in the first two parts of phase one. In summary, phase one provided proof of concept of RVGT's applicability to test functional requirement conformance but that oracle creation can be difficult, which caused the RVGT script to have a trustworthiness of 99 percent.

In phase two, 1000 tests were randomly performed with random input to test 10 quality attributes, summarized in Table I, for the travel planner application. Out of the 1000 generated tests, two threw exceptions of which one was a fatal exception that caused that particular test to fail. Hence, 999 tests were completed and one failed due to an exception. Figure 4 visualizes the test results and also shows that the test cases were chosen with an even distribution. Further analysis of the results shows that the tests had overall high success-rate but with test number 4 being an outlier. Figure 4 shows that the travel planner primarily struggled with performance and availability tests. Worth noting is that the 10 test cases, due to the eighth NFR conformance test (See Table I), were performed in two different web-browsers, i.e. every time this test case was randomly chosen the script switched from one web-browser to the other. Analysis of the test case success-rate showed no significant difference between the two browsers since the 114 browser switches were uniformly distributed over the 1000 generated tests. This test also showcases VGT's flexibility and ability to work seamlessly with several GUI based application once.

Analysis of the outlier, test 4, showed that its low success-rate was caused by faulty oracle implementation. If the resulting list of available bus fares from a search did not include any late fare the test failed. Thus, once again showing the difficulties of developing a perfect oracle. Further analysis of the results showed that most failures were caused by high network latency that caused the scripts' assertions to time out when the travel planner application did not respond within the specified time frame. The result that was the most puzzling was that the travel planner in some cases accepted negative time input in test 10. No explanation was found for this behavior since it could not be replicated manually.

In phase three, a survey was performed with 10 industrial practitioners to evaluate if there is an industrial need for RVGT. Results of the survey provide support for a need of the technique since most of the industrial practitioners stated that RVGT would perceivably improve the company's current testing (Median value 3, see question in Section III-C).

Consequently, the results from the evaluations performed in phase one, two and three provide proof of concept that RVGT is applicable for conformance testing of both functional and non-functional/quality requirements. However, this research is only pivotal, showing that the technique is at all applicable and that there is an industrial need for the technique. Future

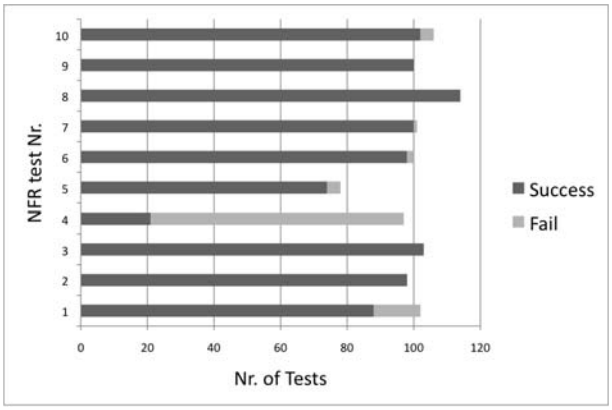


Fig. 4. Summary of the results from experiment 4. Each bar shows the number of successful and failed NFR test cases.

RQ	RQ answer	Summary
1	Yes	The random test script, developed in phase one to test calculators, provides proof of concept for random Visual GUI Testing's (RVGT) applicability to test functional requirements.
2	Yes	The random test script, developed in phase two for the commercial "travel planner" application provides proof of concept for random Visual GUI testing's applicability to test non-functional/quality requirements of different types, i.e. performance, availability, reliability and usability requirements.
3	Yes	The results from the questionnaire survey provides support that there is an industrial need for random Visual GUI Testing.

TABLE II
SUMMARY OF THE COLLECTED RESULTS FROM EACH EXPERIMENT AND THEIR CONNECTION TO THE RESEARCH QUESTIONS.

work therefore includes evaluation of RVGT on more types of software systems, web, desktop, industrial software, etc., as well as other requirement types, e.g. robustness, safety and security.

V. CONCLUSION

Random Visual GUI Testing (RVGT) takes the strengths from Visual GUI Testing (VGT), e.g. flexibility and user emulation, and combines it with the practices of random testing to create an automated GUI based random testing technique. To the author's best knowledge there is no previous work that focuses on the combination of random testing and image recognition-based system under test (SUT) interaction. However, there are other tools available for automated random testing but these tools interact with the SUT on a lower level of system abstraction, which limit their capabilities for testing, for instance, non-functional/quality requirement (NFR) conformance, e.g. SUT usability. These tests are instead performed manually in industry.

In this paper we have presented a three phase evaluation study with the goal of providing initial proof of concept for RVGT, i.e. that the technique is applicable for both functional and NFR conformance testing. Phase one of the study was performed with two calculator applications for which a test script was written in the open-source tool Sikuli [15] that generated and inputted random numbers through GUI interaction and then compared the visual output automatically. To test the fault-finding ability of RVGT, mutation testing was used to introduce faults in a calculator's operations that caused it to produce faulty output 10 percent of the time in a long-term test with 1000 generated RVGT test cases. 112 mutants were generated that could all be identified by the RVGT script. In phase two, a script was written for a commercial web-application to test its NFR conformance. Results showed that RVGT works for test cases executed in random order, with random textual and numeric input, asserted through automated visual inspection. Furthermore, in phase three, a questionnaire survey was performed with 10 industrial practitioners at the company Saab AB that showed a need for the technique.

In summary, this study provides initial proof of concept that RVGT can be applied for GUI based random testing of functional as well as non-functional/quality requirement conformance testing. Furthermore, the study shows that there is a need for the technique in industry which also shows that further research into RVGT is warranted.

REFERENCES

[1] I. Jovanović, "Software testing methods and techniques," *The IPSI BgD Transactions on Internet Research*, p. 30, 2009.
[2] M. Olan, "Unit testing: test early, test often," *Journal of Computing Sciences in Colleges*, vol. 19, no. 2, pp. 319–328, 2003.
[3] A. Adamoli, D. Zaparanuks, M. Jovic, and M. Hauswirth, "Automated gui performance testing," *Software Quality Journal*, pp. 1–39, 2011.
[4] E. Börjesson and R. Feldt, "Automated system testing using visual gui testing tools: A comparative study in industry," *ICST*, 2012.
[5] E. Alegroth, R. Feldt, and H. Olsson, "Transitioning manual system test suites to automated testing: An industrial case study," *ICST*, 2012.
[6] G. Rothermel, R. Untch, C. Chu, and M. Harrold, "Prioritizing test cases for regression testing," *Software Engineering, IEEE Transactions on*, vol. 27, no. 10, pp. 929–948, 2001.
[7] J. Itkonen and K. Rautiainen, "Exploratory testing: a multiple case study," in *2005 International Symposium on Empirical Software Engineering, 2005.* IEEE, 2005, p. 10.
[8] R. Hamlet, "Random testing," *Encyclopedia of software Engineering*, 1994.
[9] P. Godefroid, N. Klarlund, and K. Sen, "Dart: directed automated random testing," in *ACM Sigplan Notices*, vol. 40, no. 6. ACM, 2005, pp. 213–223.
[10] J. Forrester and B. Miller, "An empirical study of the robustness of windows nt applications using random testing," in *Proceedings of the 4th conference on USENIX Windows Systems Symposium-Volume 4*. USENIX Association, 2000, pp. 6–6.
[11] K. Claessen and J. Hughes, "Quickcheck: a lightweight tool for random testing of haskell programs," in *Acm sigplan notices*, vol. 35, no. 9. ACM, 2000, pp. 268–279.
[12] W. Gutjahr, "Partition testing vs. random testing: The influence of uncertainty," *Software Engineering, IEEE Transactions on*, vol. 25, no. 5, pp. 661–674, 1999.
[13] J. Bach, "Exploratory testing explained," *Online: http://www. satisfice. com/articles/et-article. pdf*, 2003.
[14] J. Andrews, L. Briand, and Y. Labiche, "Is mutation an appropriate tool for testing experiments?[software testing]," in *Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on*. IEEE, 2005, pp. 402–411.
[15] T. Yeh, T. Chang, and R. Miller, "Sikuli: using gui screenshots for search and automation," in *Proceedings of the 22nd annual ACM symposium on User interface software and technology*. ACM, 2009, pp. 183–192.
[16] T. Gorschek, C. Wohlin, P. Carre, and S. Larsson, "A model for technology transfer in practice," *Software, IEEE*, vol. 23, no. 6, pp. 88–95, 2006.

Using Change Entries to Collect Software Project Information

Hazeline U. Asuncion[1], Macneil Shonle[1], Robert Porter[2], Karen Potts[1], Nathan Duncan[1], William Joseph Matthies Jr.[1]

[1] Computing and Software Systems
University of Washington, Bothell
Bothell, WA 98011 USA
{hazeline, mshonle, pottsk2, njd91, wjoem}
@ u.washington.edu

[2] Department of Computer Science
The University of Texas at San Antonio
San Antonio, TX 78249 USA
robertportercs@gmail.com

***Abstract*—Confronted with tight project deadlines, a development team is often under pressure to make decisions regarding the project (e.g., Which features can be included in the next release? Is the software product ready for release?). In order to make these decisions, it is necessary to obtain information from multiple sources, including source code in different languages and documentation in different formats. In this paper, we present a technique that uses change entries to obtain relevant project information. Our technique, FACTS PT, automatically extracts, traces, aggregates, and visualizes change entries along with other software metrics to provide project information. Results from our case study at the ABC Organization* suggest that the information provided by the FACTS PT is useful to project managers and developers. We also offer lessons learned regarding collecting and presenting information to a team in a proprietary and regulated software development context.**

Keywords-Software traceability; Software analytics; information management

I. INTRODUCTION

Connecting related information during the course of software development, referred as software traceability, is necessary for various software lifecycle tasks such as determining conformance to requirements or assessing the quality of design [7, 14]. However, connecting related information found in different artifacts to answer project management questions has received little attention [9]. (We define an artifact as any human-produced file during the software lifecycle.) Ideally, a team should be able to answer questions related to project status, quality of software artifacts, and compliance to requirements. To answer these types of questions and to make informed decisions, information from various sources must be collected.

Collecting project information has been referred to as software metrics [16], or software telemetry [18]. Techniques have been developed to support the automated collection of both process and product metrics [13, 18, 22, 26]. These techniques, however, are generally focused on collecting process information to aid developers assess their own skills and productivity. Project managers, meanwhile, require process and product information that provide them an understanding of the overall state of the code, the progress toward a milestone, and the skills and productivity of each member of the team. Buse and Zimmerman recently used the term software analytics to refer to the type of data and data analysis necessary to support managers in their decision-making tasks [9]. This paper provides a technique to support software analytics for project managers and developers.

Agile software development methods also have techniques for collecting project information. Burn down charts are used to track the velocity of a team over several iterations [11]. These techniques are built-in to the agile process and are difficult to adapt to teams which use other lifecycle models, such as the waterfall or spiral models. Incidentally, the concept of software analytics is also being used in agile projects [25].

We present a technique for tracing related information from various sources. Our technique, Flexible Artifact Change and Traceability Support for Project Team (FACTS PT), uses change entries as a level of abstraction by which changes across various artifacts can be uniformly represented and monitored. Previous approaches to collecting project information include tracking number of errors detected or number of lines of code added [16]. We posit that change entries do not only serve as a useful metric for project progress, but they also provide more meaningful information regarding the *reason* behind specific changes. Our contributions are: 1) change entries as a means of gathering scattered project information, 2) a set of tools for extracting, tracing, aggregating, and visualizing change entries and other metrics, 3) evaluation in a real-world setting which suggests the utility of our approach, and 4) lessons learned from the study.

This paper is organized as follows. The next section provides a motivation behind our technique. Section 3 presents our FACTS PT technique, followed by tool support in Section 4. Section 5 covers evaluation of our technique with lessons learned. Section 6 discusses related work. We close the paper with avenues of future work.

II. MOTIVATION

We now provide an overview of the challenges faced by a development team working in a proprietary and regulated software development context, such as the ABC Organization* where we conducted a case study of our technique.

This work is supported by the US National Science Foundation under Grant No. 1218266.

* Kept anonymous here due to a non-disclosure agreement.

The ABC Organization is a research institute which engages in scientific research and develops software for various applied science domains. The organization has several thousand employees with branches in the United States and around the world. Because the organization works on software projects that must adhere to government standards and regulations, and because these software products may be deployed on critical systems, the software must pass a rigorous quality assurance standard. The organization also uses the Capability Maturity Model Integration (CMMI) level as a means of demonstrating its maturity level [2].

The software development team which we studied uses a hybrid waterfall and iterative software development lifecycle. The team comprises of a project manager, a lead software engineer, 3-4 developers, 1-2 test engineers, and a documentation engineer. The team releases software in roughly 10-month cycles. Not only must the organization be able to assess the quality of their code, they must also demonstrate process maturity to reach the next CMMI level.

Since the group uses an iterative development, it is important for the development group to continually monitor changes performed on the code and accompanying artifacts for each release cycle. In addition, the team must be able to quickly obtain information from various artifacts, which are often scattered among the various tools used by the team. Finally, the team must be able to answer questions such as "What is the status of the project?" and "Are the source code and documentation meeting quality standards?"

III. FACTS PT TECHNIQUE

To address the challenges faced by a development team in tracing relevant information from heterogeneous sources, we used the FACTS PT technique. This technique consists of the following steps: select a tracing unit to connect heterogeneous sources of information, embed concepts within change entries, extract change entries and other project metrics, aggregate and visualize extracted data. We now discuss each step in detail.

A. *Select a Tracing Unit to Connect Heterogeneous Sources of Information: Change Entry*

Since a software team is confronted with large amounts of information from multiple sources, it is necessary to only trace the relevant information. One of the challenges with many traceability recovery techniques is the generation of potentially large number of false positive links along with accurate links [19]. This can be addressed by identifying a tracing unit that can connect heterogeneous information across the project lifecycle. The tracing unit serves as a means of "marking" artifacts, or parts of the artifacts, to trace.

Previous techniques use requirements [7], architecture concepts [5], or events as tracing units [24]. We connect information via change entries, which are descriptions of changes made in the artifacts (e.g., source code, documentation) (see Figure 1). A change entry can be mapped to specific deletions or additions in a file. Since change entries can be uniformly represented across heterogeneous artifacts, we use change entries as our tracing unit.

B. *When Possible, Embed Project Concepts within Change Entries*

Once the tracing unit is selected, the next step is to embed it with concepts understandable by a team. In our context, an important higher level concept is a task. A task is a software update to be performed by a software engineer. A task may entail implementing a new feature, performing a bug fix, or carrying out a code maintenance activity.

Since artifacts are at different levels of abstractions (e.g., requirements document contain abstract concepts while source code contain concrete concepts), project concepts may only be embedded in some change entries. For example, the concept of task is a higher level concept by which source code changes can be aggregated. Meanwhile, requirements document changes may not be related to tasks since concepts at the requirements document are at a higher level of abstraction than tasks. It is certainly possible to embed requirement change entries with another set of concepts, such as requirement IDs, and then develop a mapping between requirements and tasks.

C. *Extract Change Entries and other Project Metrics*

The next step is to extract change entries and other relevant project metrics to gather project information. To address the challenge of extracting change entries from heterogeneous artifacts, we use artifact-specific extractors, such as an extractor for each source code language, an extractor for specification documents in PDF, an extractor for test documents in PDF, etc.

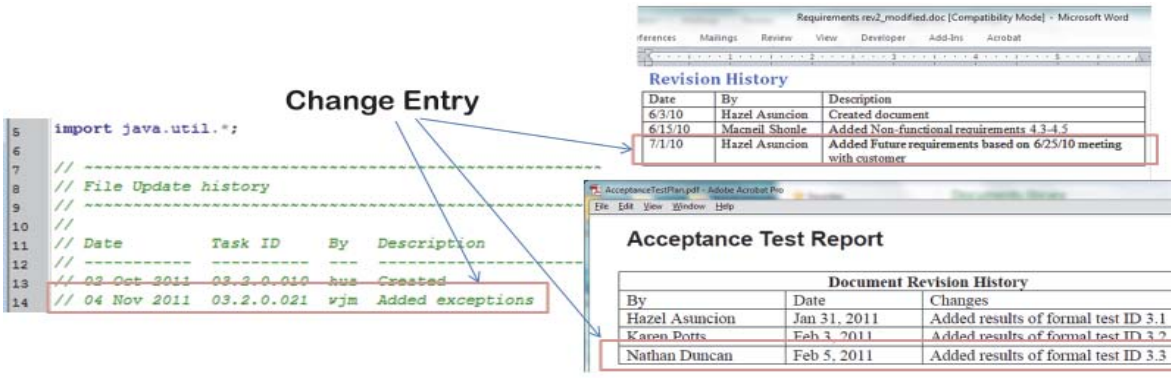


Figure 1. Examples of change entries on various software artifacts in different formats (source code, requirements documentation, and test report).

The extracted change entries are then represented as a uniform change model. We use XML to represent our change model, which contains the following information: a unique identifier, author, date of the change, task ID (optional), description, and path. All of these attributes, except the path, are obtained from change entries. The description is a free-form text that describes the change. To support access to the artifacts that were changed, it is also necessary to determine the location of the artifact–whether on the local machine, on the local network, or on the Internet. The path may also point to a specific location within the artifact to support accessibility at different levels of granularity [6]. The path is automatically determined relative to the project root folder. An XML file contains multiple change entries and multiple XML files may be used to encapsulate groups of change entries.

We also collect pertinent project metrics associated with managing the project. These include length of time spent by a developer on a task and number of lines changed for source code files. To minimize the overhead in collecting metrics, each developer simply reports the total number of hours spent on each task. The project manager compiles these self-reported hours into a Software Tasks spreadsheet, which includes additional task information. Once the information is in a spreadsheet, we can automatically extract the self-reported hours. Metrics regarding number of lines changed for source code files are obtained using a diff tool (e.g., SVN diff [12]).

D. Aggregate and Visualize the Extracted Data

Once the change entries, which contain task IDs, and other project metrics have been obtained, the data can then be aggregated and visualized at different levels of granularity. At a high level, an overview of project progress and potential problem areas in the code can be provided. At a detailed level, one can view how the aggregated information was derived, which parts of the documentation or source code has been changed, which parts of the code are non-compliant to coding standards, and which tests have passed or failed.

We generated the following types of visualizations: Change Lookup by Developer, Release Comparison Report, Change Distribution Chart, Timeline of Change Entries, Gantt Chart Actuals, and Task-Author-Artifact Report. The Change Lookup by Developer allows developers to search for their tasks and all the changes they performed for the current iteration. The Release Comparison Report shows a listing of all the source code files that have been changed between two specified releases. Within this report, additional information such as detailed diff and coding standard reports are provided for each file and are accessible via hyperlinks. The Change Distribution Report, meanwhile, visualizes changes along various dimensions, by software engineer, by file type, by subsystem, by hours spent, or a combination of these dimensions. The Timeline of Change Entries provides a chronological ordering of change entries. The change entries

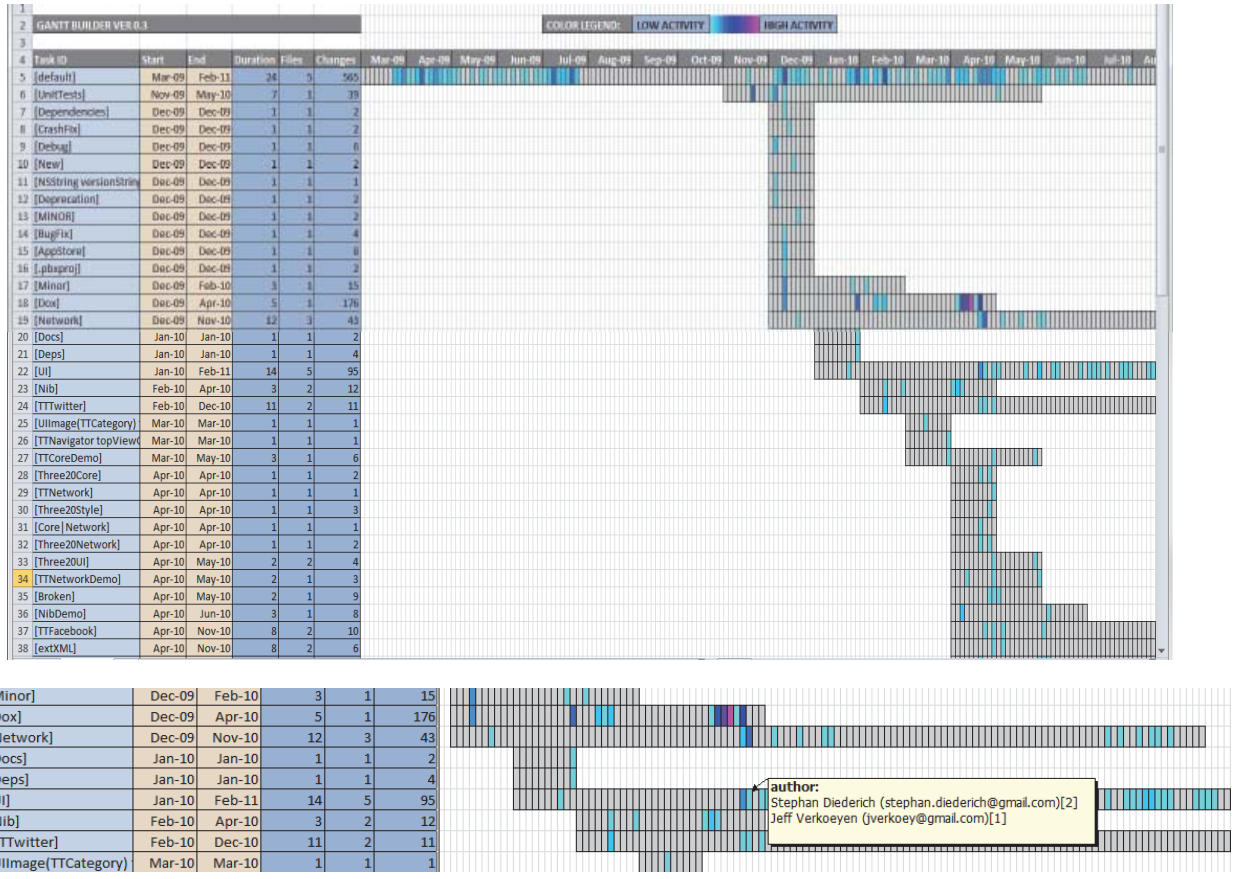


Figure 2. Gantt Chart Actuals using data from the jEdit open source project. Top figure shows zoomed out view of the chart. Bottom figure is a zoomed-in view of the UI Task with a comment indicating the authors who performed changes for the selected date along with the number of change entries.

are color-coded according to various dimensions (e.g., author, file type, subsystem). We discuss in detail the rest of the visualizations.

Gantt Charts are generally used for project planning to determine when a software product can be released [27]. We have adapted the Gantt Chart to show actual project information based on change entries (see Figure 2, top screenshot). Similar to a Gantt Chart, the Gantt Chart Actuals lists tasks (or task IDs) on the left side of the chart. Each Gantt row spans a period of n months and is color-coded based on the number of change entries. Upon hovering over a colored heat map cell, additional metadata such as list of authors that performed changes for that date and number of change entries in brackets, are shown (see Figure 2, bottom screenshot).

Change entries, along with its mapping to a more abstract concept (e.g., task) and more concrete concepts (e.g., lines added or deleted, time spent), are shown in the Task-Author-Artifact Report. In this report, a user can select a task ID. A list of software engineers who are working on the specified task is shown, along with summary data for the number of change entries and hours spent for the selected task. Within each developer, detailed information is shown, such as files that were changed, the percentage of the total lines that are comments, number of lines added and deleted. This report also allows a user to assess the complexity of a task by combining self-reported information (number of hours) along with automatically generated metrics, such as the number of files modified and number of lines added or deleted.

IV. Tool Support

We built artifact-specific change extractors using Perl and Python scripts to extract change entries from requirements and design specifications (in PDF and Word) and from source code written in Java and JSP. Since each artifact type follows company standard formatting, it is straightforward to locate the section of the file that contains the change entries. Task IDs are included with each source code change entry (see Figure 1), and are included in the extraction process. Once the change entries are found, the change extractors write each change entry to an XML file with the appropriate XML tags. All generated XML files are then combined into one file.

The change entries and the project metrics, both in XML formats, are then fed into the visualizations. The Timeline of Change Entries was built using Piccolo 2D. The spreadsheet visualizations were built on top of Microsoft Excel 2010 spreadsheet in the .NET 4.0 Windows environment using C# and the Excel API. Spreadsheet visualizations were used since the team was comfortable with analyzing project data in the spreadsheet environment. The Gantt Chart Actuals and timeline visualizations have been fully implemented, while the other spreadsheet visualizations are partially implemented. For the Release Comparison Report, we used an SVN diff [12] to identify the differences between two releases. We also built a script that analyzes high priority coding standard violations and outputs a report for each source code file. Alternatively, an off-the-shelf coding standard checker could also be used.

V. Evaluation

The FACTS PT technique and tool support was evaluated in the context of an ongoing software project that has over 200K total lines of code and implemented in five different languages and scripts. We analyzed the change entries from the Java and JSP codebase which covers about half of the total codebase. The project also has numerous artifacts including requirements specifications, design documents, test plans, test documents, checklists, tasks, and change requests. These artifacts are in different file formats (e.g., spreadsheets, documents, PDF files, diagrams). FACTS PT was used to extract change entries from a subset of these artifacts, to relate tasks and change entries to developers, and to support tracking project progress. The artifacts from three major releases were studied, with thousands of change entries. The change entries spanned the period of January 2009 to May 2011.

We were primarily focused on determining the utility of change entries and their visualizations to developers or project managers for their respective tasks. Thus, we sought answers to the following research questions:

Q1: Does the FACTS PT technique assist you in development or management tasks? If so, in what way?

Q2: How useful are the FACTS PT visualizations?

We solicited information from various members of the team: a project manager who has 15 years of experience as a software project manager and three programmers who have about 10-15 years of experience. The subjects were presented with the visualizations after the releases and were asked to provide feedback via questionnaires and semi-structured interviews. We conducted four iterations of the study (and improved the FACTS PT tool support after each iteration).

A. Results

Q1: In the early iterations of the study, both the project manager and developers concurred that the FACTS PT technique did not assist them in their tasks.

In later iterations, both project manager and developers stated that FACTS PT can assist them in their tasks. The developers stated that the FACTS PT technique allowed them to quickly identify which files have changed and to understand the source code changes. The project manager stated that in its current state, the FACTS PT technique can assist him with project management tasks by identifying areas of improvement from the generated visualizations and reports after a major release. These areas of improvement can then be addressed in the next software development iteration.

Q2: In early iterations of the study, the development team stated that the FACTS PT visualizations were not useful.

In later iterations, the visualizations were useful to the development team. The programmers were able to quickly locate the changes they made with the Change Lookup by Developer and were able to reflect upon their own productivity. According to the project manager, the Gantt Chart Actuals (Figure 2) and the Task-Author-Artifact Report were rated as providing highly useful information because they provide

summary information. The other visualizations require further changes to be rated as highly useful.

B. Discussion

Q1: Throughout the four iterations, we followed the same general steps of extracting, aggregating, and visualizing change entries, except for the additional steps of selecting and embedding project concepts within change entries and including project metrics in later iterations. It turned out that these additional steps were keys in transforming the FACTS PT into a technique that can assist project managers and developers with their tasks. Viewing changes at the granularity of change entries was acceptable to all the subjects.

Q2: Throughout the study, we visualized change entries. It was interesting to learn that the type of visualization can largely affect the perceived usefulness of the change entries. Although the Timeline of Change Entries we initially used provided some insight into the project, all subjects had difficulties navigating it and was unable to quickly obtain aggregate information across different dimensions (e.g., by developer). Meanwhile, all subjects found the tabular visualization format as most useful, especially when it contained information extracted from various sources, as in the Task-Author-Artifact Report or the Change Lookup by Developer. The project manager added that when the FACTS PT tool support becomes more mature, it can also be used during a project iteration, as opposed to simply being used at the end of an iteration as part of a post-release analysis. The developers also expressed interest in visualizing their changes from other projects. Doing so would allow them to cross-reference the changes they make across different projects.

C. Limitations

Our approach makes the following assumptions:

Change entries are present in the files to trace. Development teams which produce formal specifications often have a history log as part of the document template (see IEEE Std 830-1998 [1]). In addition, many development teams also use a configuration management (CM) system which contains commit records. These commit records can be used as a source of change entries if history logs are not used. Open source projects also maintain a change log [10]. Time spent on tasks may be estimated from CM check-out/check-in timestamps.

It is possible that the developer-entered information, such as hours spent on tasks and change entries may contain incorrect information, or even missing information [10]. This inaccuracy would be fairly straightforward to detect since the developer-entered information is presented with the automatically generated metrics. In addition, if developers are incentivized for demonstrating a higher level of activity, via the change entries, it is less likely they will neglect providing change entries each time they make a change to a file.

With regards to limitations with our evaluation, we focused on whether tracing, aggregating, and visualizing change entries with other project metrics is useful to project managers and developers. We did not examine the overhead involved with processing the artifacts. This is a subject of future work.

D. Lessons Learned

1. A software development team is not keen on using new tools or technology unless they have a direct benefit. This finding is consistent with the adoption of software traceability techniques in industry [7]. Thus, even though we also presented change entries in the earlier iterations, the team was not willing to use the tool because the information was not accessible to them. Later on, when we presented the change entries in both the aggregated and detailed level, along with other project metrics, the team was more willing to use our tool and technique.

2. Aesthetically pleasing visualizations do not necessarily provide usable information. Since a development team is constantly under time pressure, a visualization must enable them to obtain information quickly. Thus, support for easy navigation, filtering, searching, and data manipulation are key requirements for a usable visualization. This is one of the reasons why tabular formatted data is preferred by the subjects. The information can easily be aggregated (by invoking the sum function), filtered by an attribute, or searched by a keyword. Our timeline visualization, while aesthetically pleasing and classifies changes according to author or file type, does not provide capabilities for fast data manipulation, and thus, was not useful to them.

3. Using a combination of self-reported and automated metrics can lower the overhead for collecting metrics, while minimizing privacy issues that may be associated with fully automated data collection techniques. By leveraging existing company practices in extracting metrics, more applicable metrics can be obtained. Moreover, some of the fully automated techniques in metric collection may under-report the actual time spent on an activity. Since the automated techniques are based on engineer interaction with tools [18, 26], these techniques do not measure the time away from the computer (e.g., face-to-face meetings with teammates).

On the other hand, manually tracking time can be a time consuming process [16] and may potentially distract engineers from their task since they are required to context switch between tracking their time and performing development tasks [17]. In a company setting, a balance can be achieved by tracking course-grained activities and tracking time at 10 or 15 minute increments. Recording time spent on activities can be performed at the same time as engineers report their timesheets (e.g., once a day), to avoid the context switching problem. In our context, the engineers track their time at the task level and the reported times were generally accurate.

VI. Related Work

We now compare our work to related research areas.

Software Traceability: Software traceability research is concerned with identifying relationships between various software artifacts [5]. Traceability techniques and approaches have generally been developed to support an analyst or a requirements engineer [15], an architect [5], or a developer [4], but not project managers. One case study describes the use of bug tracking as a tracing unit to support developers [21]. If a tracing unit is not embedded into the artifacts to trace, then

other techniques like link recovery techniques [4, 15] can be used to identify possible connections between artifacts. Limited traceability support for project management tasks was described in another study [7]. Jazz is a tool that supports mapping of information across various artifacts that reside within the Jazz platform [3]. Our work, however, uses change entries as a tracing unit, and aggregates and visualizes them with other project metrics to support developers and managers. Our work is also not constrained to a specific tool or platform.

Process metrics: Several process metrics have been previously proposed to support project management, including time spent on activities and number of defects discovered during code inspection [16]. Goal-Question-Metric paradigm provides guidance on which metrics to collect [8], while CQMM is a technique that collects metrics from different static analyses tools to monitor and assess code quality [23]. In agile development contexts, story points are used to track the amount of work performed in each iteration [11]. Using change entries as a metric is complementary to these techniques.

Metric Collection: There are tools that collect process and product metrics. One particular category of tools, Software Project Control Centers (SPCCs) are used to collect, interpret, and visualize project metrics to provide context-, purpose-, and role-oriented information for various members of the development team [20]. Other tools use different techniques to collect metrics, such as using sensors attached to various tools (e.g., development editors, build tools) [18], tracking evolution of classes, methods, invocations [22], or tracking personal software process (PSP) data [26]. Another tool allows users to plug-in their custom metrics into a provided infrastructure [13]. Our technique, meanwhile, combines self-reported metrics from developers (i.e., time spent on tasks) with extracted change entries and product metrics.

VII. Conclusion

In this paper, we presented FACTS PT, a technique that traces change entries across heterogeneous artifacts to collect project information. We developed a set of tools that automatically extracts, traces, aggregates, and visualizes change entries along with other project metrics. Our case study at a proprietary and regulated software development context indicates that our approach is useful to project managers and developers. We also offered lessons learned regarding collecting and presenting accessible information to a software development team.

We plan to continue improving the FACTS PT tool support. We will also solicit feedback of other members of the development team, including QA engineers and documentation engineers, to determine how our technique can also assist them in their tasks. Finally, we plan to analyze the description of change entries to automatically group together related changes.

VIII. Acknowledgements

We thank the project manager and developers at the ABC Organization for valuable insights and feedback. Dang Nguyen at UTSA developed the initial timeline visualization.

References

[1] IEEE Recommended practice for software requirements specifications. *IEEE Std 830-1998*, 1998.

[2] CMMI Institute. http://cmmiinstitute.com/, Jan 2013.

[3] The Jazz Project. http://jazz.net, Jan 2013.

[4] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo. Recovering traceability links between code and documentation. *IEEE Transactions on Software Engineering (TSE)*, 28(10):970–983, 2002.

[5] H. U. Asuncion, A. U. Asuncion, and R. N. Taylor. Software traceability with topic modeling. In *Proc of 32nd Int'l Conference on Software Engineering (ICSE)*, 2010.

[6] H. U. Asuncion and R. N. Taylor. *Software and Systems Traceability*, chapter Automated Techniques for Capturing Custom Traceability Links across Heterogeneous Artifacts, pages 129–146. Springer London, 2012.

[7] H.U. Asuncion, F. François, and R. N. Taylor. An end-to-end industrial software traceability tool. In *Proc of the 6th Joint Meeting of the European Software Eng Conf and the ACM SIGSOFT Int'l Symp on the Foundations of Software Engineering (ESEC/FSE)*, 2007.

[8] V. Basili and S. Green. Software process evolution at the SEL. *IEEE Software*, 11(4):58 –66, 1994.

[9] R.P.L. Buse and T. Zimmermann. Information needs for software development analytics. In *Proc of ICSE*, 2012.

[10] K. Chen, S. R. Schach, L. Yu, J. Offutt, and G. Z. Heller. Open-source change logs. *Empirical Software Engineering*, 9(3):197–210, 2004.

[11] M. Cohn. *Agile Estimating and Planning*. Prentice Hall, 2006.

[12] CollabNet. TortoiseSVN. http://tortoisesvn.tigris.org/, Jan 2013.

[13] G. Gousios and D. Spinellis. A platform for software engineering research. In *Proc of Int'l Working Conf on Mining Software Repositories*, 2009.

[14] V. L. Hamilton and M. L. Beeby. Issues of traceability in integrating tools. In *IEE Colloquium on Tools and Techniques for Maintaining Traceability During Design*, 1991.

[15] 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.

[16] W. Humphrey. *A Discipline for Software Engineering*. Addison-Wesley, 1995.

[17] P. M. Johnson and A. M. Disney. A critical analysis of PSP data quality: Results from a case study. *Emp Software Engr*, 4(4):317–349, 1999.

[18] P. M. Johnson, H. Kou, M. Paulding, Q. Zhang, A. Kagawa, and T. Yamashita. Improving software development management through software project telemetry. *IEEE Software*, 22(4):76 – 85, 2005.

[19] J. Leuser. Challenges for semi-automatic trace recovery in the automotive domain. In *Proc of the 5th Int'l Workshop on Traceability in Emerging Forms of Software Engineering*, 2009.

[20] J. Münch and J. Heidrich. Software project control centers: concepts and approaches. *Journal of Systems and Software*, 70(1–2):3 – 19, 2004.

[21] C. Neumüller and P. Grünbacher. Automating software traceability in very small companies - a case study and lessons learned. In *Proc of the 21st Int'l Conference on Automated Software Engineering*, 2006.

[22] J. Oosterman, W. Irwin, and N. Churcher. EvoJava: A tool for measuring evolving software. In *Proc of the Australasian Computer Science Conference*, 2011.

[23] R. Plösch, H. Gruber, C. Körner, and M. Saft. A method for continuous code quality management using static analysis. In *Proc of the Int'l Conf on Quality of Information and Communications Technology*, 2010.

[24] W. Poncin, A. Serebrenik, and M. van den Brand. Process mining software repositories. In *Proc of the European Conference on Software Maintenance and Reengineering*, 2011.

[25] Rally Software. Advanced analytics. http://www.rallydev.com/platform-products/advanced-analytics, Jan 2013.

[26] A. Sillitti, A. Janes, G. Succi, and T. Vernazza. Collecting, integrating and analyzing software metrics and personal software process data. In *Proc of the Euromicro Conference*, 2003.

[27] D. White and J. Fortune. Current practice in project management — an empirical study. *Int'l Journal of Project Mgmt*, 20(1):1 – 11, 2002.

Improving Software Engineers' Skills through the Simulation of Distributed Software Development in Academic Environments

Luiz Leandro Fortaleza, Olavo Olimpio Matos Júnior, Tayana Conte
Federal University of Amazonas - UFAM
Manaus, Brazil
{luizfortaleza,olavomatos,tayana}@icomp.ufam.edu.br

Sérgio Roberto Costa Vieira
Fucapi
Manaus, Brazil
sergio.vieira@fucapi.br

Rafael Prikladnicki
PUCRS
Porto Alegre, Brazil
rafaelp@pucrs.br

***Abstract*—The software industry needs practitioners with communication and collaboration skills. By stimulating the improvement of these skills in Software Engineering courses, software engineers can be better equipped for the software development market. The use of Distributed Software Development (DSD) in academic environments can lead students to improve their skills. However, implementing a DSD course can be costly. A cost-effective alternative to include DSD is to simulate it. In this paper, we present an empirical study aiming to verify the use of DSD simulation in academic environments as a way of improving communication and collaboration skills. In our controlled experiment we encountered similar results to the ones of studies in which the subjects were distributed. Therefore, the simulation was efficient in the context of this experiment. By showing indicators of its efficiency, we intend to stimulate the use of DSD simulation as a cost effective way to prepare software engineers so that they can meet the needs of the software development industry.**

Keywords-DSD simulation; software engineering education; communication and collaboration skills; controlled experiment

I. INTRODUCTION

Communication and collaboration skills are essential in the software development industry since they can provide competitive advantages to software development companies [1]. However, despite the fact that Software Engineering courses generally provide technical knowledge, few of these courses focus on improving the students' skills. This affects the improvement of communication and collaboration skills since they are often improved by gaining experience at industrial environments.

By stimulating the development of skills during undergraduate courses, it is possible that the students become better practitioners in lesser time. Having better prepared practitioners could reduce the cost in training programs in the software development industry. It is possible to facilitate the development of skills in academic environments by using real-world scenarios [2].

In this context, the adoption of real software industry practices in academic environments can lead the students to develop their skills. One of these practices is the Distributed Software Development (DSD) in which the development team is geographically distributed. The use of DSD aims to obtain competitive advantages, such as: speed-to-market, cost reduction, and resources availability [3], [4].

When an individual participates in a DSD project he/she can have some improvement in his/her skills [5], [6]. Hence, the adoption of DSD in academic environments can be an effective way to lead Software Engineering students to become better trained practitioners regarding their skills.

By including DSD projects in Software Engineering courses, we can allow students to experience a practice that has become common in the software industry [2], and at the same time stimulate the improvement of their skills. Nevertheless, the implementation of DSD in regular Software Engineering courses can be a challenging task [7], [8]. It generally requires (a) the alignment of the pedagogical goals from two or more institutions, and (b) a high coordination effort to manage the activities within the involved institutions.

An alternative to bypass the difficulties of adopting DSD is to simulate geographical distribution. The use of simulation can lead to results similar to the ones of a real distribution of the software team, but with a lower cost. In this paper, we report the results of a controlled experiment that simulates a DSD scenario in the classroom. In this empirical study we use simulation with avatars in order to verify the subjects' perception regarding the improvement of their communication and collaboration skills. Furthermore, to achieve a better understanding of the studied phenomenon, we performed a

qualitative analysis using Grounded Theory (GT) procedures [9].

The results from this empirical study showed that the students perceived an improvement in their communication and collaboration skills, which reinforces that the use of DSD can improve the practitioners' skills [5], [6]. Moreover, we identified that the simulation of DSD obtained similar results as real DSD practices. Furthermore, as DSD simulation requires less effort to be performed, we identified indicators that the simulation of DSD scenarios can be a lower cost alternative to include DSD practices in software engineering courses, particularly in institutions that do not have previous experience with DSD projects.

Although we used DSD simulation to stimulate the development of communication and collaboration skills, we reported the results regarding skills development in our previous work [10]. In this paper, we focus in the simulation environment and the simulation-based teaching approach. Consequently, by describing how we simulated DSD, we intend to provide an outline for other researchers willing to replicate the simulated scenario within this paper. As a result, the scientific community can benefit from new studies in the area. Moreover, as simulating DSD development environments requires lesser effort than the actual distribution of students, we intend to stimulate the use of such approach for training better prepared software engineers.

This paper is organized as follows. In Section II we present the background on DSD simulation. Section III presents the method that we use in our research while in Section IV we discuss our results. Finally, we present our final remarks in Section V.

II. BACKGROUND

Although Software Engineering has many characteristics from other engineering disciplines, it presents new challenges regarding software developers' knowledge and skills [11]. Therefore, in order to be prepared for the software development market, software engineers must possess skills that go beyond technical knowledge.

Adopting industry practices into the academic environment can be an efficient way to promote the improvement of skills by Software Engineering students [1], which can lead to software engineers that are better prepared for the expectations of the software development industry. The development of software with distributed teams is one of these practices.

The Distributed Software Development (DSD) is characterized by the geographic distribution of the development team. Such distribution can influence the interactions among the team members, and it can lead them to a scenario in which communication and collaboration processes have even greater importance.

Begel and Nagappan [5] state that software engineers who participated in DSD projects had perceived a significant improvement in their skills, which allowed them to become better prepared practitioners. An experiment using DSD in an academic environment was carried out with students distributed between two cities from Brazil's south. This experiment showed that the involved students also perceived an improvement on their skills [6].

The use of DSD has become a common practice in the software development industry [3]. Furthermore, the number of universities that use geographical distribution of teams in Software Engineering courses has increased [12]. Such approach shows the academy's interest in providing graduate students with knowledge and experience in DSD.

When adopting DSD projects in the classroom, we need to overcome some difficulties ranging from the search for a partner institution to the alignment of coordination strategies [7]. These challenges can lead to project failure. The use of simulations of DSD environments is a way to mitigate this risk [13], [14].

Simulation can be understood as the process of using one or more agents embedded in operational scenarios in an observational environment to measure characteristics of a particular phenomenon using a fictional representation of the reality [15]. Furthermore, the literature presents some examples of the use of simulation of DSD scenarios:

Keenan et al. [14] simulated a DSD industrial scenario through a partnership with another research institution. In this simulation, the students' teams were separated. In this case the simulation referred to the industrial scenario, while the distribution of the teams was real.

Kroll et al. [16] carried out a controlled experiment where the distribution of teams was simulated. The authors investigated the possible benefits brought by the use of a specific DSD method.

In our research, we use a simulated DSD scenario to stimulate the improvement of communication and collaboration skills of students from Software Engineering courses. We collected qualitative data on the subjects' perception. The analysis of these data provided us with some evidences of the improvement of the students' communication and collaboration skills. Through the qualitative analysis we have also observed the students' satisfaction regarding the developed activity.

We believe that using simulation can be particularly interesting for institutions that do not have previous experience with distributed projects, which is the case of the institution that participated in this experiment. We will now describe the research method in the following section.

III. RESEARCH METHOD

We carried out an empirical study with undergraduate students. These students were from the Information Systems and Computer Science courses from a university located in Manaus, a city from Brazil's North Region. The institution and the students had never participated in DSD projects and our empirical study lasted one academic semester.

As mentioned in Section I, the results regarding communication and collaboration skills are thoroughly described in our previous work [10]. In this paper, our main goal is to provide an overview of the simulation environment.

We intend to verify if the use of DSD simulation can be as useful as the real distribution of teams to stimulate the improvement of communication and collaboration skills. Furthermore, it can be a cost-effective alternative for some institutions with lack of experience in DSD projects.

Therefore, in this study we aim to observe the students' perceptions regarding the improvement of their communication and collaboration skills through the use of a DSD simulated environment. We obtained similar results compared to those in which real distribution was used. Such results indicate the efficiency of the simulation-based teaching approach, which can stimulate the use of simulation in other institutions. Table 1 presents our research goal according to GQM paradigm [17].

TABLE I. RESEARCH'S GOAL ACCORDING TO GQM PARADIGM

Analyze	The students' perception provided through interaction logs and questionnaires
For the purpose of	Understanding
With respect to	- The improvement of communication and collaboration skills; - The satisfaction of carrying out the developed activity;
From the viewpoint of	The researchers
In the context of	A controlled experiment with undergraduate student who had never used DSD.

The students interacted with avatars. These avatars were operated by one of the researchers who had never had any previous contact with these students. Aiming to provide more realism to the simulated scenario, the students did not receive any information regarding the geographical location of the avatars' operator, thus they could imagine a possible location.

A total of 17 subjects participated in this study, however only 14 of them provided all data in the post-experiment questionnaire. All the subjects signed a consent form authorizing the use of collected data in this research.

The subjects were divided into groups according to their experience in real software development projects. This occurred in the following way:

- The subjects were divided into three main experience categories: low, medium, and high;
- We randomly selected students from each category to form the groups. However, we balanced the groups based on their experience. There were a total of five groups: two with four subjects and three with three subjects each.

Each group of students received a document with the requirements specification of a system for the management of scientific events. The specification document had intentional errors and ambiguities in order to stimulate the interaction among the students and with the virtual team to solve the problems.

Each team had a leader responsible for the communication with the correspondent virtual team. These leaders were instructed to interact with the local team, and they were responsible for expressing the opinion of the whole group. The teacher, who supervised the class, noted that in each group, the members interacted with each other. This is an indicator that all students participated actively in the experiment.

There were three stages in this activity:

- First, the students received the system's description, documents and interacted for the first time with the avatars;
- Then, the students had a day to interact and solve misunderstandings;
- Finally, the students sent the final version of the requested artifacts: a document with the use cases and the use cases diagram;

Figure 1 shows the representation of the configuration of the teams and their interactions. The avatars' operator was a researcher whose identity was unknown to the subjects. Furthermore, even though we also stimulate the use of e-mails, the students did not use them because they preferred to interact thought the instant messaging programs.

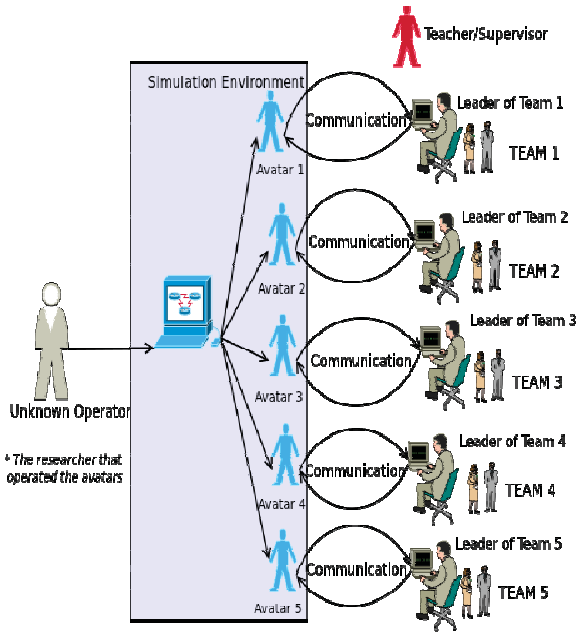


Figure 1. Configuration of the experiment.

The avatars' operator used two computers accessing 5 browsers simultaneously. Therefore, he could operate the five avatars at the same time. This provided more realism to the experiment because the teams were together at the computing lab and the operator could interact with all the teams in the same time.

We collected data from different sources: chat logs, post-experiment questionnaires, and interview with the teacher. We performed a data crossing in order to provide greater reliability to the results. Figure 2 shows an example of the interaction between a team and one of the avatars.

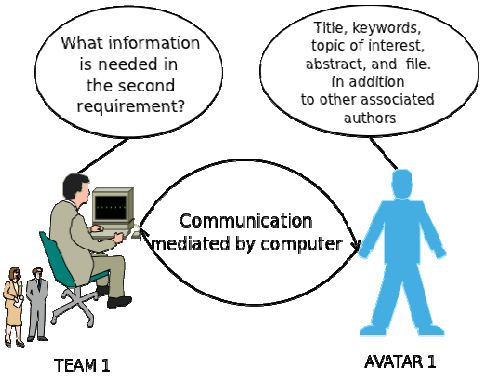


Figure 2. Open coding process

During the academic semester we had 4 meetings with the teacher of this class. These meetings were useful to monitor the students' performance, because it provided us with a vision of what was happening in the computing lab during the experiment.

A. *Data Analysis Procedures*

To analyze the collected data, we use Grounded Theory (GT) procedures. The coding process is the GT's basis [9], [18]. During this process the researches identify concepts, also called codes, as well as create categories that represent a grouping of the identified concepts [19]. The GT method can be divided into three phases [9]:

- Open Coding: identification of initial codes and categories;
- Axial Coding: identification of possible relationships between the categories identified during the first phase;
- Selective Coding: refining of the analysis. In this phase we identify the core category that groups all the other categories.

In this research we only used the open and axial coding. The purpose of the selective coding is the creation of a theory, while our goal is to verify if the use of simulation of DSD in academic environments can stimulate the improvement of communication and collaboration skills. Therefore, we decided not to use the selective coding. Moreover, in order to use the selective coding it would be necessary to achieve the theoretical saturation of the studied phenomenon.

The analysis procedures were performed by one of the researchers and reviewed by the others, thus we had many rounds of data analysis and discussion. The reviews aimed to reduce the bias of having the opinion of a single researcher, and to increase the reliability of the results.

Figure 3 shows an example of the open coding process. In this specific example, we present an extract from a questionnaire that was associated with the codes: (a) simulation of real application, and (b) use of prototypes.

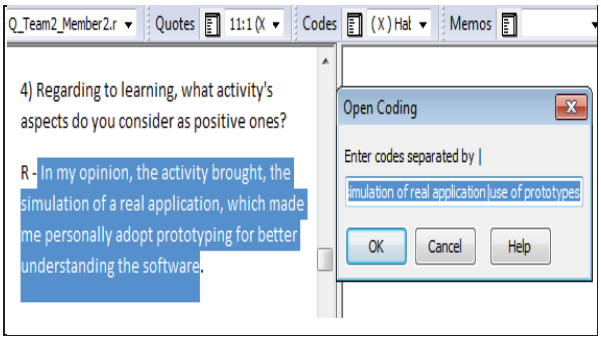


Figure 3. Open coding process

Figure 4 shows an example of the axial coding. In this figure we can observe some of the identified relationships among different codes. Furthermore, we highlighted the codes regarding communication and collaboration skills.

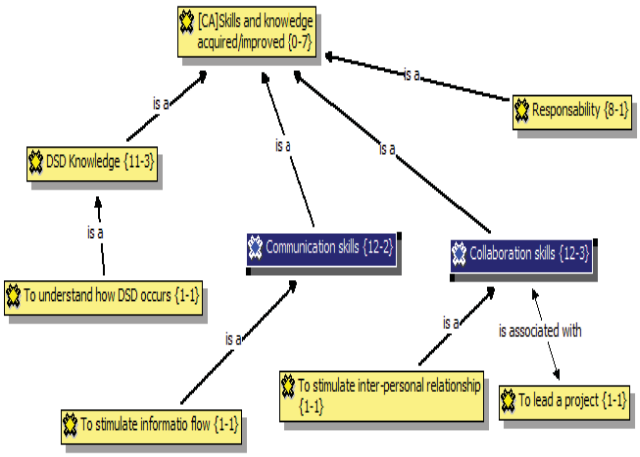


Figure 4. Representation of the relationship among some codes

IV. RESULTS

In the post-experiment questionnaire the subjects were asked to choose, from an items list, what skills they perceived had improved through the performed activities. Among the selectable items, there was a "collaboration and communication skills" item. The subjects had to answer an open question in which they had to describe how they perceived the improvement.

From the total number of students who answered the post-experiment questionnaire, about 85% stated that they had noticed an improvement in their communication and collaboration skills. The subjects also perceived a positive impact in their: technical knowledge skills, ability to abstract skills, among others. Furthermore, they also demonstrated satisfaction with the developed activity. We illustrate these results with the following quotations:

> "*The leadership of a real project allowed me to work my abstraction and collaboration skills*" – Member 1 from Team 3.

"*The activity helped me understand how does the software development works*" – Member 1 from Team 1.

"*In my opinion, the activity brought the simulation of a real application, which made me to adopt prototyping for better understanding the software*" – Member 2 from Team 2.

"*I could better understand the software analyst's work and the importance of the communication in a software development company*" – Member 1 from Team 5.

"*The activity allowed me to see, in a practical way, how does the modeling occurs in a real development process*" – Member 3 from Team 2.

This last quotation also demonstrates a high degree of immersion from the subjects in the simulation, because despite not being in a real process, they realized it as such. We can also infer this from the following quotations:

"*I could learn with people that have more experience than me*" – Member 1 from Team 4.

"*A positive aspect of the activity was the interaction with experienced people*" – Member 2 from Team 1.

"*I could perceived how really DSD occurs in the real life during the system analysis*" – Member 2 from Team 3.

"*A positive aspect was the interaction with a real client*" – Member 1 from Team 2.

As explained in Section III, the subjects did not have any knowledge about the true identity of the operator of the avatars. Moreover, they assumed that they were more experienced people or even real customers. This is an indicator that these students were able to imagine themselves in a real development scenario, which may explain why they believed their skills improved.

The teacher who oversaw the class also perceived the students satisfaction with the activity. During our interview, he said:

"*In the beginning of the experiment, they were very motivated and felt in a real environment*" – Teacher.

Since approximately 85% of the students perceived the improvement of their skills, we can infer that, in this specific context, the use of DSD simulation is a viable alternative to promote the development of skills for future software engineers in academic environments. We base this assumption on previous work that used a real distribution and obtained similar results [6] and in the fact that DSD simulation costs less than the use of real distribution.

Figure 5 shows a graphical representation of the answers from the students regarding the improvement of their skills and knowledge. We have highlighted the communication and collaboration skills since these skills were considered by the subjects as the skills with the highest degree of improvement. Figure 5 also shows the percentage of subjects who reported an improvement on a specific skill or knowledge. According to Kirkpatrick [20] the students' perception is an indicative of learning.

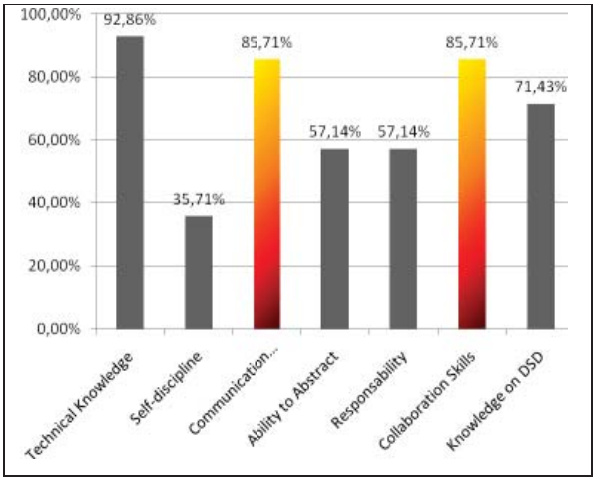


Figure 5. Graph with quantitative results

Our goal was to focus on communication and collaboration skills, but as we can see in Figure 5, the subjects also indicated an improvement in other skills and knowledge, such as the ability to abstract. This reinforces the importance of using distributed development in undergraduate courses to immerse the students in an environment similar to the industrial environment to improve their skills and knowledge.

Another point worth mentioning is the fact that 92.86% of the students noticed an improvement in their technical knowledge. Therefore, we can assume that our simulation-based approach allowed us to lead the students to improve their communication and collaboration skills, and in the same time we preserved the importance of technical knowledge.

V. FINAL REMARKS

In the context of this experiment, the use of DSD simulation in an academic environment was efficient for the improvement of communication and collaboration skills from software engineers in the context of this experiment. This improvement was observed by the class teacher and also through the data collected from logs of interaction and questionnaires with semi-open questions. Thus, we conducted a data crossing in order to provide greater reliability to results. However new studies are needed to obtain more conclusive findings.

The implications brought by the obtained results are important because they confirm that the use of simulation can be as beneficial as the real scenario of DSD. The use of this alternative is likely to be more interesting in some cases because it presents a lower cost than the use of real distribution since it requires less effort. The benefits related to the communication and collaboration skills are reported in other paper [10].

Moreover, our results lead us to plan further empirical studies. Some possibilities of further work are listed below:

- To replicate the experiment in other institutions in different regions;
- To use other software development activities, such as software testing;
- To investigate a different set of skills;
- To consider the psychological profile of students as one of the criteria for team formation;
- To use programmed virtual agents to express reactions, thus increasing the simulation level. Furthermore, it can result in even better evidence.

A limitation of our study is related to the sample size that does not allow the generalization of the results. However, in qualitative researches the main goal is to understand a phenomenon in depth. Each qualitative study aims to increase the body of knowledge about a particular phenomenon. By replicating a qualitative study it is possible that one day we will achieve the theoretical saturation, which will make it possible to create an explanatory theory. Therefore, the qualitative research process is iterative, and our research contributes to achieving a goal in the long term.

Despite the limitation of the sample size, our students demonstrated satisfaction with the developed activity and they also demonstrated a high level of immersion in the simulation. Thus, we can assume that is possible to use DSD simulation to improve the skills and knowledge of software engineers with a lower cost than using real distribution. Hence, we expect to contribute to stimulate the use of DSD simulation as a way to prepare software engineers for the expectations of the software industry.

ACKNOWLEDGMENT

This study is partially funded by the FTS-Brasil Project (CNPq 483125/2010-5). Rafael Prikladnicki is a CNPq researcher (CNPq 309000/2012-2). We also thank to CNPq and Fapesp for the support granted by INCT-SEC through processes 573963/2008-8 and 08/57870-9. And we would like to acknowledge the support granted by FAPEAM process PAPE 055/2013

REFERENCES

[1] H. Ye, "An academia-Industry collaborative teaching and learning model for software engineering education," in *Proceedings of the 21st International Conference on Software Engineering & Knowledge Engineering (SEKE'2009)*, 2009, pp. 301-305.

[2] R. Dawson, "Twenty dirty tricks to train software engineers," in *Proceedings of the 22nd International Conference on Software Engineering (ICSE)*, 2000, pp. 209-218.

[3] J. D. Herbsleb and A. Mockus, "An empirical study of speed and communication in globally distributed software development," *IEEE Transactions on Software Engineering*, vol. 29, no. 6, pp. 481-494, 2003.

[4] M. Humayun and C. Gang, "An empirical study on improving trust among GSD teams using KMR," in *Proceedings of the 24th International Conference on Software Engineering & Knowledge Engineering (SEKE'2012)*, 2012, pp. 131-134.

[5] A. Begel and N. Nagappan, "Global software development: who does it?," in *Proceedings of the 3rd International Conference on Global Software Engineering (ICGSE)*, 2008, pp. 195-199.

[6] R. Prikladnicki, "Can distributed software development help the practitioners to become better software engineers?: insights from academia," in *ICSE 2011. In: Proceedings of the 1st Collaborative Teaching of Globally Distributed Software Development*, 2011, pp. 16-19.

[7] J. Favela and F. Peña-Mora, "An experience in collaborative software engineering education," *IEEE Software*, vol. 18, no. 2, pp. 47-53, 2001.

[8] O. Gotel, V. Kulkarni, C. Scharff, and L. Neak, "Working across borders: overcoming culturally-based technology challenges in student global software development," in *Proceedings of the 21st Conference on Software Engineering Education and Training (CSEE&T)*, 2008, pp. 33-40.

[9] A. C. Strauss and J. M. Corbin, *Basics of qualitative research: techniques and procedures for developing Grounded Theory*, 3rd ed. SAGE, 1998.

[10] L. L. Fortaleza, S. R. C. Vieira, O. O. Matos Júnior, R. Prikladnicki, and T. Conte, "Using distributed software development in improvement of communication and collaboration skills in SE courses: an observational study," in *Proceedings of the 26th Conference on Software Engineering Education and Training (CSEE&T)*, 2013, pp.139-148.

[11] S. B. Lee and K. Steward, "A process-based approach to improving knowledge sharing in software engineering," in *Proceedings of the 24th International Conference on Software Engineering & Knowledge Engineering (SEKE'2012)*, 2012, pp. 700-705.

[12] L. L. Fortaleza, T. Conte, S. Marczak, and R. Prikladnicki, "Towards a GSE international teaching network: mapping global software engineering courses," in *ICSE 2012. In: Proceedings of the 2nd Collaborative Teaching of Globally Distributed Software Development*, 2012, pp. 1-5.

[13] M. Romero, A. Vizcaíno, and M. Piattini, "Teaching requirements elicitation within the context of global software development," in *Mexican International Conference on Computer Science*, 2009, pp. 232-239.

[14] E. Keenan, A. Steele, and X. Jia, "Simulating global software development in a course environment," in *Proceedings of the 5th International Conference on Global Software Engineering (ICGSE)*, 2010, pp. 201-205.

[15] D. Tappan, "Pedagogy-oriented software modeling and simulation of component-based physical systems," in *Proceedings of the 21st International Conference on Software Engineering & Knowledge Engineering (SEKE'2009)*, 2009, pp. 295-300.

[16] J. Kroll et al., "Follow-the-sun software development: a controlled experiment to evaluate the benefits of adaptive and prescriptive approaches," in *Proceedings of the 24th International Conference on Software Engineering & Knowledge Engineering (SEKE'2012)*, 2012, pp. 551-556.

[17] V. R. Basili and H. D. Rombach, "The TAME project: towards improvement-oriented software environments," *IEEE Transactions on Software Engineering*, vol. 14, no. 6, pp. 758-773, 1988.

[18] D. Viana, T. Conte, D. Vilela, C. R. B. de Souza, G. Santos, and R. Prikladnicki, "The influence of human aspects on software process improvement: qualitative research findings and comparison to previous studies" in *Proceedings of the 16th International Conference on Evaluation & Assessment in Software Engineering (EASE 2012)*, 2012, pp. 121-125.

[19] P. Fernandes, T. Conte, and B. Bonifácio, "Improving a web usability inspection technique through an observational study," in *Proceedings of the 24th International Conference on Software Engineering & Knowledge Engineering (SEKE'2012)*, 2012, pp. 588-593.

[20] D. L. Kirkpatrick. *Evaluating training programs – the four levels*, Berrett- Koehler Publishers, 1994.

A Feasibility Study of Follow-the-Sun Software Development for GSD Projects

Josiane Kroll, Rafael Prikladnicki, Jorge L. N. Audy
Computer Science School, PUCRS
Porto Alegre, Brazil
josi.unc@gmail.com, rafaelp@pucrs.br, audy@pucrs.br

Erran Carmel
Kogod School of Business, AU
Washington DC, USA
carmel@american.edu

Jude Fernandez
Infosys Technologies Ltd
Bangalore, India
JUDEF@infosys.com

Abstract— **Follow-the-sun (FTS) is a strategy for Global Software Development (GSD) where you hand off work at the end of every day from one site to the next, many time zones away, in order to speed up product development. Companies have tried to implement FTS, but have abandoned it after some point because of the difficulty to put it into practice. Consequently, there are few documented industry successes. The lack of FTS experience in the software industry is observed as the main barrier for its adoption. For this reason, we performed a study applying FTS to develop a software project. Our goal was to examine the feasibility and outcomes of FTS. In this paper, we present the experience report describing best practices and solutions performed to overcome the challenges we found.**

***Index Terms*—Follow-the-sun, software engineering, global software development, coordination across time zones.**

I. INTRODUCTION

Follow-the-sun (FTS) is a software development strategy for Global Software Development (GSD) projects used to take advantage of temporal distances between several sites located in different time zones [1]. Its main goal is to reduce software development cycle duration [2]. However, FTS implementation requires great coordination, collaboration and communication with all team members involved [1].

While FTS concept looks promising in theory, it appears to be difficult into practice [3]. Many software companies have attempted to implement FTS, but have abandoned it after some point because of the difficulty of putting it into practice [1].

For this reason, this study aims to examine the FTS feasibility and understand the challenges and possible solutions for its development.

Our study was performed at Infosys Technologies in Bangalore, India in the second quarter of 2012. Over the duration of one month, working teams distributed in Mexico, India, and Australia developed a software application using FTS concept. In this study, we present our results, details of software practices and solutions performed to overcome the challenges found to develop a software application in the FTS mode. We also discuss feasibility issues, lessons learned and the next steps planned for this study.

II. BACKGROUND FOLLOW-THE-SUN

In FTS scenarios, team members are distributed across different time zones and sites [2]. When team members from one site finishes its own regular working hours, other team members located in another site and time zone start its working day. Tasks are handed off from one site to another at the end of each working day [4]. The tasks transition between team members is called a handoff.

Handoffs are performed daily by teams following to the next site. At each site, handoffs are conducted on a daily basis at the end of each site shift [5].

In the literature, FTS is also referenced as round-the-clock. Although these terms are used in a similar way in the literature, the definitions are different. FTS is about speed, cutting project duration, while round-the-clock is about twenty-four hour coverage, running an operation in all shifts. Both of these concepts use time zone differences to design shifts, but for different purposes and kinds of tasks [2].

III. STUDY SETTINGS AND METHODS

Our research was developed at Infosys Technologies in Bangalore, India. Infosys is a global leader in consulting, technology and outsourcing with revenues of US$ 7075 million (FY12). Infosys provides business consulting, technology, engineering and outsourcing services to help clients in over 32 countries.

A. Case Setting

Our study focused in the development phase of a software application. This software application was developed by three distributed teams located in different time zones: Mexico, India and Australia.

In Mexico, there were two developers who were available full-time, in India two developers available for half-time and one project manager and in Australia we had two developers full-time and one developer half-time.

The sites had different experience levels. In Mexico, developers were trainees. In India, the project manager had working experience of approximately 10 years, did not have prior experience as a project manager. Developers from India, had two years and one year's experience respectively. In Australia, developers had between eight and fourteen years' experience.

B. Project Planning

1) Estimation

The application development was estimated to be completed in 4 weeks duration, and it was divided into two sprints. Since we did not have any estimation techniques or variations of standard techniques suited for FTS, we went more based on standard approaches and the experience of the project

manager. We also estimated the time for this project considering a typical two-location mode, which was estimated to be 6 weeks (this was done by an external experienced project manager not connected with this study).

2) Task Allocation

We followed the CPro concept introduced by [6]. Based on CPro formation concept, we formed 2 CPs (Composite Personas) with each CP comprising at least one team member from each location.

In each sprint, tasks were allocated to the CP, rather than to an individual. However, we found that this method caused some amount of confusion amongst CP members and resulted in some lack of direction and progress of the work in the sprint 1. Thus, this process was slightly modified in the sprint 2.

In the sprint 2, the project manager carried out twenty-four hour allocation of tasks to each CP. Following this each CP member would take on the tasks in the appropriate logical manner based on the tasks allocated for the twenty-four hours window.

3) Team training

We prepared guidelines for FTS teams based on the literature [1] [2]. These guidelines were reviewed by FTS experts before the project start. These guidelines were used to plan the project and conduct training sessions to clarify FTS.

Only the team from Australia had experience in agile methods. To mitigate this issue, we also conducted training sessions about the Scrum method. Additionally, a scrum master was allocated to the project and his role in the project was to coach and ensure that the teams followed Scrum.

C. FTS Methods and Guidelines

1) Team setup

Based on time zones where the teams were located and their available working hours, the daily working hours were arranged such that there was an overlap of 30 minutes between the locations for communication to enable the daily handover of tasks. Initially, the handover call was planned for 30 minutes in duration each day, subsequently it was found that it spilled over 45 minutes and 1 hour based on the need.

2) Daily handover across locations

Handoffs were performed over phone calls or communication tool. For sending tasks to another team, each developer should use an Excel template. This template was available on TFS system and it was called Task Handover. Individual developers were asked to add information for each handoff.

3) Communication between team members across locations

Communication between team members across locations was synchronous only during overlap times between the locations. A team member cannot talk with teammates from another site outside their time hours. Other forms of asynchronous communication were via email.

Time windows for interaction were available only one hour (maximum) per day. The first time window for interaction was available in the first 30 minutes (maximum) of a working day. The second time window for interaction is available on the last 30 minutes (maximum) of a working day.

The working day started in Australia following to India and after to Mexico. Handoffs using conference calls between developers and project manager should be done every day to discuss new and performed tasks.

4) Retrospectives at the end of each sprint

Following the Scrum framework, at the end of sprint 1, a detailed retrospective was conducted giving opportunity for all members to voice out what they felt went right and what did not. This helped considerably to identify issues and also identify potential solutions and improvements to the overall process.

IV. RESULTS

A. Performance

1) Performance in the sprint 1: The tasks in the sprint 1 were estimated for an effort of 368 hours. At the end of sprint 1, it was found that effort expended was 432.5 hours, which was 65.5 hours more than planned. It was also found that only 65.5% of the planned tasks were completed with the rest incomplete. These tasks were moved to sprint 2.

One of the main hurdles encountered by the team was certain delays from the internal stakeholders which necessitated rework due to new templates introduced. This was estimated to have caused approximately 50% extra work. Similarly, the setup of the project took up more time than estimated. Finally, the daily FTS handover process also took more time than estimated, because planning meeting were executed during handoffs.

2) *Performance in the sprint 2:* Considering the existing tasks and the carryover tasks from sprint 1, the effort estimated for sprint 2 was 464 hours. In the sprint 2, we observed that teams were more comfortable and productive having getting experience in the FTS approach from sprint 1. Several of the problems faced in the sprint 1 were minimized in the sprint 2. The effort expended in this sprint was only 350 hours (see Table I) which was due to team members attending trainings and two holidays in one particular location. Consequently, the task completion was approximately 62% of the planned tasks.

TABLE I. PERFORMANCE IN THE SPRINT 1 AND SPRINT 2.

	Sprint 1	**Sprint 2**
Estimated hours	368	464
Actual hours	465.5	350
Extra hours	65.5	0
Task completion %	68%	62%

3) *Completion of the project:* the remaining tasks of the project were completed in a subsequent phase in non-FTS mode, because some team members were committed to other client projects and had to be released.

B. Feasibility of the FTS model

The core question this study sought to answer was: *Is FTS feasible in a live practical scenario in a large organization?*

From our experience in this project, our conclusion was that the FTS model is feasible. We present the main reasons next.

1) Work distribution and execution in round-the-clock manner

This model was a major departure for all members of the team. The team went through some learning and ultimately settled down into a practical mode. At the overall level, the sprint user stories were divided amongst the two CPs. Further, within a CP, in the sprint 1, tasks were not specifically allocated to begin with. After facing some issues, this model was modified to some extent by adopting a 24 hours task allocation model, according to this, the project manager allocated tasks to each CP for each 24 hours. This practice brought in greater clarity in the team helping in better execution in the sprint 2.

2) Ownership of the CPro

In the sprint 1, due to the lack of defined CP ownership, this ownership was taken by the project manager which was not very effective. Post the retrospective, this model was modified and specific owners for each CP was defined, thus ensuring more commitment and execution to plan.

3) Daily Handover Process

Task handover from one time zone location to the CP member in the next time zone location was a concept tried out for the first time. As described above, and learning from previous research, a simple process was adopted. The team members reported a high level of comfort and satisfaction in the handover process and it was seen to be a good enabler in the FTS model.

C. Inherent Issues found in the Project

1) Project-specific challenges

A business application was chosen for this study with a defined end customer and stakeholders. While the end customer was supportive of the project, due to certain organizational constraints, the project faced considerable delays on account of other stakeholders. This contributed a certain amount of delay especially in the sprint 1.

The total lack of experience in one location and relative lesser experience in the second location, coupled with the higher experience in the third location, while, not entirely unusual, had some impact on the team working and productivity. Further, only one location members had adequate experience in agile software development, which meant the others had a learning curve on agile to be tackled in the project.

2) FTS methods and practices

The biggest limitation experienced in this study was the lack of good estimation techniques for the project. The team used conventional estimation techniques to arrive at a target of 4 weeks for the project to be completed, whilst a neutral estimate of the same project for two location model pegged the estimate at 6 weeks. However, the project finally took 5.5 weeks to complete. As a consequence, the study did not show a significant cycle time reduction as it could, although FTS shows to be faster than a software project develop in non-FTS mode.

3) Communication and Coordination issues

The main problem faced related to communication was language. While English was the mode of communication, due to differing accents spoken, team members across locations had some trouble understanding each other during the handoff meetings. Extra emails were requested by teams at the end of each handoff.

Coordination problems observed related to weekend handoffs task allocation and office timing management. Task allocation was problematic mainly at the beginning of the project, because tasks were not allocated properly. In addition, tasks not completed at the end of the day are handed to the next production site. Relate to office timing management, India team worked haft time and Australia team had a developer working half time.

D. Lessons Learned

With the information collected during the project, we highlight some lessons learned.

- *Templates and standard document*: at the beginning of the project, teams faced problems to identify standards utilized in the project. Teams must know templates and standard documents that will be used during the software development before the project start.
- *Coding standards*: to avoid re-work a standard to comment code must be defined before the coding starts. When the FTS project started, teams were spending a lot of time trying to understand the code and identifying the last changes made in the code.
- *Screen sharing*: transferring or explaining a task using screen sharing becomes easy when teams can see the information talked about. During the handoff process we observed that teams opted by using of the screen sharing to explain codes and design documents.
- *Communication protocol*: we observed that phone calls, emails and communicators, such as Microsoft office communicator, are useful to provide communication between teams, but they must be used together. We also observed during the calls meeting that some rules following by teams can improve the quality communication, such as, speak slowly to reduce accents and summarize the tasks talked about by CP giver.
- *Tasks for the day*: a daily email allocating tasks individually for members contributed to define priority tasks and reduce problems faced by teams to categorize a task in the sprint backlog.
- *Handover template*: we used an excel spreadsheet to manage tasks exchanged between teams. It worked very well, but it could be automatized.
- *Weekend handoff*: in the weekends is very difficult to manage the handoff processes. In this project, we have used communication via email. However, many problems were identified mainly in the first weekend. The receiver team faced difficulties to understand the new tasks and how to continue the work. On the second weekend was better, but the tasks were discussed before starting the weekend.

- *CP owner*: some tasks were assigned to a CP owner during the sprint 1. We observed that is a good way to ensure complete tasks. Tasks can be assigned by email to CP owners per location. Each CP owner will check if the task has been completed.

V. DISCUSSION

We consider some practices from literature designed for around-the-clock environments and it was adapted for FTS model. The experience of Infosys' experts allowed to improve practices and to create a software process for FTS. The adoption of Scrum practices was considered innovative in this context.

At the end of sprint 1, some changes were made in the process for the sprint 2. These changes are present as lessons learned in the section D.

We observed that the imbalance experience level had a negative effect on the project. The lack of experience affected from the project level to estimate the hours to complete tasks as well as the execution of tasks. Other challenges were identified like task allocation and lacking of standards and templates at the beginning of the project. However, these challenges were minimized for the next sprint.

A. Constraints & Limitations

This study has some limitations that must consider:

- *Imbalance team's experience*: employees with different experience levels were allocated for this study. To minimize possible threats, we conducted training sessions with entire team before study start. In addition, we created guidelines giving instructions about FTS approach and scrum methodology.
- *FTS experience*: the software process followed by FTS teams was created by Infosys experts based on own methodologies. Experts in FTS and agile methods reviewed guides, practices and processes.
- *Agile experience*: the lack of the team's experience in agile was minimized allocating one scrum master for the project. His role in the project was to make the FTS team follow scrum method.
- *Single application*: one of the major limitations of this study is that they have examined only a single system developed by a single organization.
- *Team availability*: in the middle of sprint 2, team members from two locations were allocated to other client projects. For this reason, the project was completed using a non-FTS mode.

VI. CONCLUSIONS AND FUTURE WORK

In this study, we reported the experience acquired with FTS applied to develop a software project. We used best practices from the literature and experts from the Infosys to create a process for FTS. Many software practices performed shown to be effective for FTS. In other hand, others shown to be ineffective resulting rework hours.

The main contribution of this study relates to the feasibility of FTS. Our findings show that FTS works for GSD projects with some evidence that FTS can be used to compress duration. However, many untypical issues had occurred during the project. Team members attending trainings and developers without experience allocated to the project, are some examples. In the end, the take away from this study at Infosys is that FTS is feasible.

A. Future Work

First, FTS needs to be experimented with more projects. Our study has shown good results using Scrum practices, but there is a need to gain more experience and understanding of when it works well, and how making it work better.

Another future opportunity is to study the impact of FTS in software projects in terms of cycle time reduction, as this is the main benefit expected from the implementation of a FTS project.

Our findings show that FTS is feasible, but it is hard to execute. We observed that few studies in the literature report solutions regarding to team coordination, task allocation and the process for daily handoffs.

Finally, the experience at Infosys show that FTS is an alternative to develop global software projects spread out in different time zones. Learning how to take the advantages for applying FTS successfully its part of the next steps at Infosys.

ACKNOWLEDGMENT

This project was financially supported by the Infosys Technologies Company-India. The authors are grateful for the access and support provided by the Infosys Technologies Company involved in this study. We also thank the PDTI program, financed by Dell Computers of Brazil Ltd. (Law 8.248/91) and Lero - The Irish Software Engineering Research Centre (www.lero.ie). Rafael Prikladnicki is a CNPq researcher (309000/2012-2).

REFERENCES

[1] E. Carmel, J. Espinosa and Y. Dubinsky, "Follow the Sun Workflow in Global Software Development," Journal of Management Information Systems Vol. 27 No. 1, 2010, 17 – 38.

[2] E. Carmel and J. A. Espinosa, I'm working While They're Sleeping: Time Zone Separation Challenges and Solutions, Kindle Edition, 2011, 188 p.

[3] C. Visser and R. V. Solingen, "Selecting Locations for Follow-the Sun Software Development: Towards A Routing Model," Fourth IEEE International Conference on Global Software Engineering, 2009.

[4] E. Carmel, A. Espinosa and Y. Dubinsky, "Follow the Sun Software Development: New Perspectives, Conceptual Foundation, and Exploratory Field Study," 42nd Hawaii International Conference on System Sciences, 2009.

[5] E. Hess and J. L. N. Audy, "FTSProc: a Process to Alleviate the Challenges of Projects that Use the Follow-the-Sun Strategy," In: 7th International Conference on Global Software Engineering (ICGSE), 2012, Porto Alegre, Brazil.

[6] A. Gupta, R. Bondade and N. Denny, "Software Development Using the 24-Hour Knowledge Factory Paradigm," (April 29, 2008). Available at SSRN: http://ssrn.com/abstract=1130062.

Structural Testing of Autonomous Vehicles

Vânia de Oliveira Neves, Márcio Eduardo Delamaro,
Paulo Cesar Masiero, Caio César Teodoro Mendes, Denis Fernando Wolf
Depto de Sistemas de Computação - ICMC
Universidade de São Paulo - São Carlos, SP - Brasil
{vaniaon,delamaro,masiero,caiom,denis}@icmc.usp.br

***Abstract*—Software to control autonomous vehicles is a type of embedded system that need to undergo severe testing before deployment. After unit testing and simulations, the actual vehicle needs to pass through field testing, which are mainly functional and based on scenarios. There are two difficulties for this type of testing: input data is not discrete, they occur in large quantity, change very quickly and depend on the environment; and it is difficult to repeat the tests with the same input data, unless by simulation. In this context, a testing model and a tool to support structural testing of field testing of autonomous vehicles is proposed. The model and the tool are based on an assumption that logs of field testing related to specific versions of the program and scenarios are captured and can eventually be visualized, analyzed and compared according to control flow criteria. An example is presented.**

I. Introduction

Embedded systems are present in people's lives in a way that is difficult to imagine everyday life without them. They represent a wide range of systems from personal and domestic applications such as cell phones and microwave ovens, to large and complex applications, such as automotive and avionics, as landing systems and automatic control of fuel injection. Generally, an embedded system may be considered as a computer system for specific purposes and must function autonomously (1).

As other embedded systems, a Mobile Robotic System (MRS) is a combination of various physical (hardware) and computational (software) components that has as its main feature the ability to move and operate partially or fully autonomously. A robot reaches a higher level of autonomy when it starts to integrate capacities such as perception (sensors able to "read" the environment in which it operates); action (actuators and engines capable of producing movements of the robot in the environment); robustness, and intelligence (1).

Autonomous vehicles are a particular class of MRSs. They are critical embedded systems and, as such, face high demand for quality. Thus, it is critical that they meet their requirements. If the embedded software is not good enough, it can cause serious problems such as injury, death or even severe disasters (2). Thus, these systems must be carefully and extensively tested and this is our motivation for investigating structural testing in embedded systems, particularly control software for autonomous vehicles.

In this context, the test of autonomous vehicles (AV) is critical to the success of constructing this type of device, as Urmson et al. (3) wrote about the car that won the DARPA challenge in 2007: "Testing was the central theme of the research program that developed Boss". The test of an autonomous vehicle control unit usually starts by testing the units' smaller components (eg. classes, in object-oriented programming) and is followed by off-line integration testing simulation tools, which can use both data from the simulation environment or real data from test logs collected previously (4; 5). Field testing is usually functional and based on scenarios. For example, the scenario or testing objective of an autonomous vehicle could be "pass a slow moving vehicle" or "reach an end point from a starting point deviating from obstacles". Additionally, it could be a more general scenario as formulated by Urmson et al. (6): blind path tracking and perception assisted path tracking.

Field testings are the final and decisive tests to assure the autonomous vehicles show de intended behavior but they usually lack any information about the code structure. It could be that the vehicle (hardware and software) passed in the testing but important parts of the code could have never been executed. The functional and structural testing techniques are complementary (7) and, therefore, it is important to apply them to test software that is part of embedded systems as it ensures that critical sections of code have been exercised. Therefore, the main objective of this work is to propose a testing model and a tool to support structural testing in the context of field testing of autonomous vehicles and a support mechanism for automated validation and use of this model.

This paper is organized as follows: Section 2 discusses the state of the art of testing of autonomous vehicles; Section 3 presents the model we propose to support structural testing of AVs; Section 4 presents a supporting tool that supports the proposal; Section 5 presents an example of usage of the tool and, finally, in Section 6 a few closing remarks are presented.

II. Testing of autonomous vehicles

The structure of an intelligent autonomous vehicle powered by the drive-by-wire technology consists of a control unit complex and structured in several hierarchical levels of control; embedded mechatronic systems that rely on internal and external information obtained by the sensors of the vehicle, on a communication system that integrates all the control structures, enabling remote commands and exchange of information between the vehicle and a control station for monitoring, as well as the communication between vehicles, and internal and external sensors (1).

A useful practice during the test of autonomous vehicles is to store input data from a field test for supporting further off-line analysis, as shown in Figure 1. The development of software (and systems) usually follows numerous interactions in which field testing based on scenarios, called "regressive

tests" by Urmson et al (3), are recorded and then rerun off-line with the help of debugging tools. The tools provide different ways of viewing the program and allow inspections of data and variables values (8).

There are two important characteristics of field testing input data of autonomous vehicles that hinder testing. The first is that the data is not discrete, that is, they occur in large quantity and change very quickly. For instance, thousands of data readings are carried out with high frequency during the movement of a vehicle. The second is the difficulty of guaranteeing that the input data are repeated, even when the test is redone carefully reusing a scenario.

The authors of this article have realized, from the literature of the area and contacts with a few groups of researchers of autonomous vehicles, that normally structural testing and coverage analysis are neglected in the development processes used. One of the reasons that justify the lack of emphasis given to structural testing of software for autonomous vehicles is that it relies heavily on software tools available in the development environment and in the dynamicity of the test.

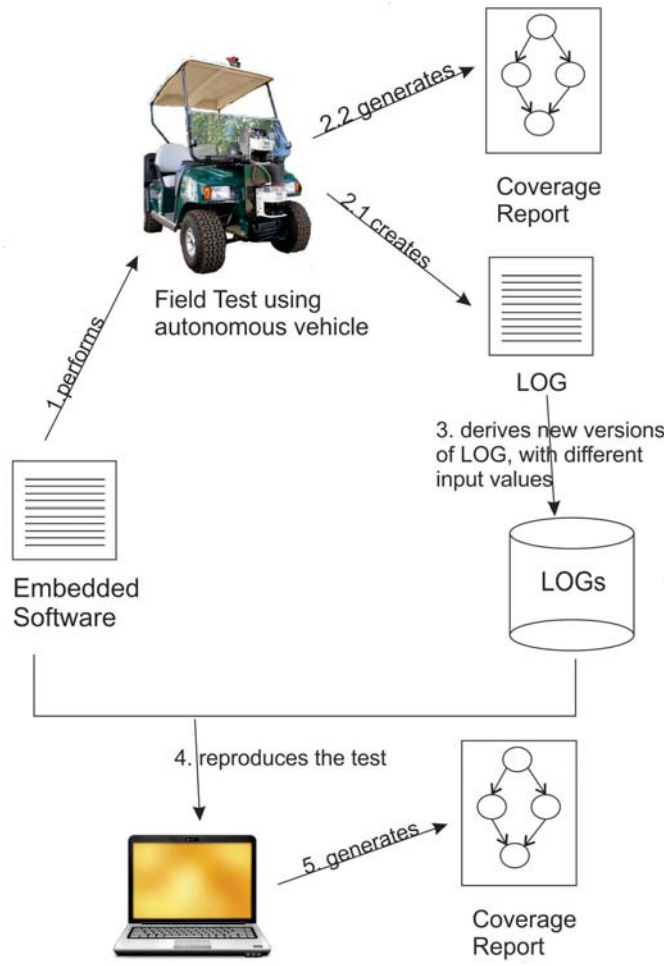


Fig. 1. Schema to Generate Test Data

Understanding and analyzing the instructions or other structural elements of the program that have not been exercised (covered) in a test scenario may be useful to assess the quality of the test. The research hypothesis of this study is that structural test can also contribute to the planning and analysis of test scenarios. For example, when planning a field testing, the tester may think: what kind of obstacle or obstacle positioning should be included in the scenario so that a method not exercised in previous testing sections can be exercised now? Or, when analyzing the coverage of a testing being able to relate the test requirements that were not covered (instructions, nodes, edges, etc.) with features of the system that have not been tested yet.

III. Proposal of a Model to Support Structural Testing of Autonomous Vehicles

Programs for autonomous vehicles go through successive refinements, many times parts of the code change or even the whole code of a unit, thus generating a large number of versions of the program and logs of testing scenarios. To manage development in this software domain it is very important to use not only testing tools but several other tools like version control, configuration management and static analysis. Moreover, it is also important to relate versions and logs of programs with testing scenarios run with each version and also with the coverage obtained. These features are not usually offered by the typical tools mentioned above. To support structural testing in this domain, the authors of this paper created a meta-model formalizing the schema of Figure 1 and representing the main concepts and artifacts involved, which is presented in Figure 2.

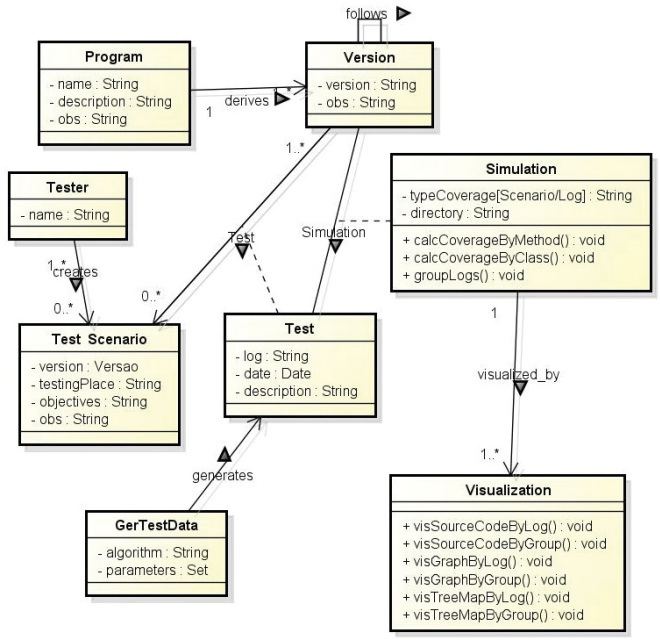


Fig. 2. A meta-model for part of a structural testing strategy

Versions of programs are defined as specific implementations of a program (or specification) that differ in some part of the code, but keep the same architectural elements. Thus, for example, a change of parameter does not create a new version but installing a new camera to help controlling a vehicle creates

a new version of the program. The test scenarios are test plans with the specific objective of testing subsets of program features under test subject to certain conditions. Logs are records of actual test data captured during a field testing of a program version for a particular scenario. The captured test data are the inputs received by the vehicle, eg, images from cameras and GPS position. To capture this data, we used the ROS system and, thus, no instrumentation is performed.

Each pair combining a version of the program and a log entry can be executed in a simulation environment and the results can be visualized using several types of criteria, graphs and granularity levels. Additionally, logs created from field testing can be used to generate other input test data by algorithms that have as objective function to improve test coverage according to certain criteria. The generated test cases can also be simulated and visualized. The logs are defined as:

- $log_{v,s}$ is a file containing data entries for the test of version v of a program for the scenario s. Notice that s can also be a log generated automatically by an algorithm for generating test data.

Based on the meta-model, it is possible to store information about the test that can be handled by a tool that supports the developer and the tester regarding the analysis of the results of a field testing from the structural and coverage points of view and use these insights to debug and refine the program (version) under test. The coverage obtained in a test is defined as:

- $cov_{v,s,c}$ coverage of the version v of the of a program executed or simulated for the scenario s and testing criterion c.

Setting a coverage criterion c, which may be the execution of all statements of a program or all the nodes or edges of the control graph of a program, the tool may then, for example, support the following comparative analysis for any coverage criterion c:

- $cov_{v_1,s,c}; cov_{v_2,s,c}; ...; cov_{v_n,s,c}$: coverage analysis of executions of different versions of a program v under the same scenario.
- $cov_{v_1,s_1,c}; cov_{v_1,s_2,c}; ...; cov_{v_1,s_m,c}$: coverage analysis of executions of the same version of a program v under different scenarios.

Figure 3 shows a graph with a generic object model for the meta-model described above, for a program P. Given this graph, pairs $< COV_{L_1}, COV_{L_2} >$ and $< COV_{L_3}, COV_{L_4} >$, for example, represent a sequence of structural coverage (by different criteria) of two tests of the same version and same scenario; the pair $< COV_{L_1}, COV_{L_6} >$ represents a sequence of the coverage of two tests of the same version for different scenarios; and the pair $< COV_{L_5}, COV_{L_1} >$ represents a comparison of the coverage of two tests of the same version, but one generated by a field testing and the other automatically generated by an algorithm that used V_1 as initial seed. It would also be possible, for example, to analyze the coverage only of executions of automatically generated logs or logs of different versions under different scenarios (if it can be useful). Analysis can also be made for pairs of coverage or for multiples coverages, depending on the tool's implementation.

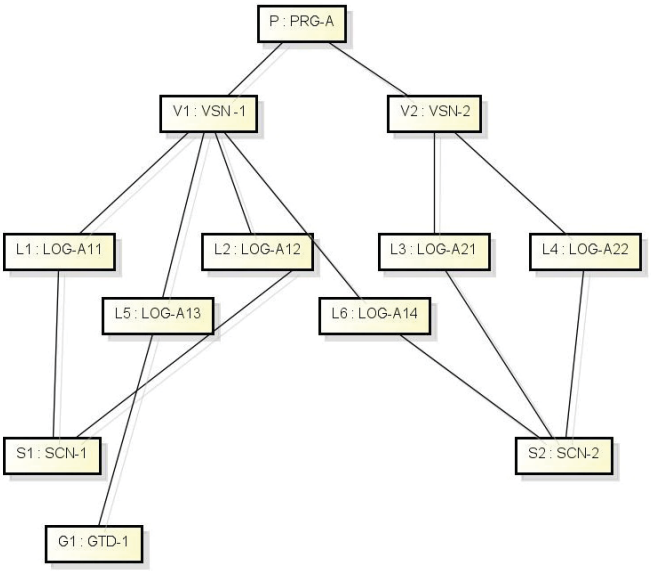


Fig. 3. An object model derived from the meta-model presented in Figure 2.

IV. TOOL SUPPORT

The model shown in Figure 2 was implemented as a tool to support field testing of autonomous vehicles. It runs in the linux operating systems and the target language is C++. The framework ROS (9) is also supported; it manages the creation of testing logs that can be imported to the tool. Within the tool environment these logs obtained from field testings can be used as inputs for simulated executions of different program versions and coverage can be visualized and analyzed. The context of each testing section can be registered indicanting the program and version identifications, the tester and the scenario used.

A simple architecture of the tool is shown in Figure 4. It uses a SBBD and the operating system infrastructure to store logs, information about field testing and other data. This module also provides several reports that support the tester to manage its testing activity. Then we have a simulation module that controls the execution of versions using logs as input and provides information for two other modules that do the actual simulation: ControlFlowGraphVis and TreemapVis. The first used an adapted version of the tool Trucov to build and show graphs (10) and the second uses a tool called Prefuse to show hierarchy of classes and methods (11)

Visualization of how the program version was exercised during a field testing can be done in several levels of abstraction and see the coverage obtained: by instructions, by nodes and branches of a control graph of each method, and by the whole class and the whole project. It is also possible to split the classes in modules or packages and visually through the architectural layers of the software. In this diagram, the size of classes or methods are proportional to their size in terms of number of lines. Moreover, the coverage of several executions of program version, using the same scenario or different scenarios can be combined to show an aggregated coverage. It also shows side by side two graphs allowing to

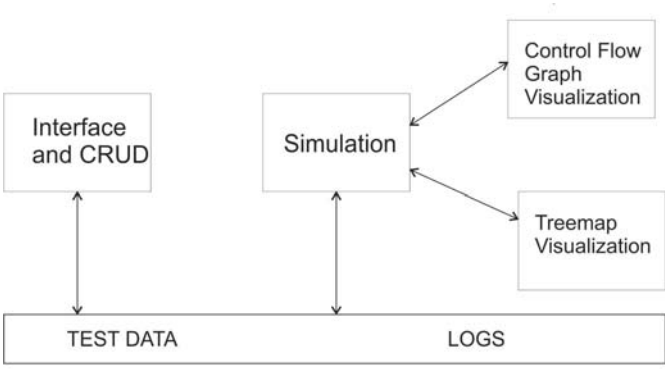


Fig. 4. Architecture of the tool

compare them using different aggregation levels and log and scenarios combination, as discussed in section III.

In the current implementation we keep track of two structural criteria: all nodes and all-edges. A design decision made to represent in colors the coverage was to group the intensity of executions of program instructions into five levels of green in which the highest coverage group is darker. The number of intensity levels can be increased or decreased as needed and work style of the tester. The nodes of the graph colored by yellow represent nodes that have been exercised, but have at least an edge not exercised by the test. The red nodes represent nodes and edges not exercised by the test.

The results of simulations are recorded by the tool. In the actual implementation, duplicates simulations for the same version of a program, scenario and log creates identical simulations but for their timestamps. Simulations chosen by scenarios or by logs. Visualizations are not recorded and are generated on line to produce coverage and all other analysis allowed by the tool.

Logs automatically generated from other logs can be stored and managed by the tool and its coverage analyzed according to the tester needs. The only overhead of our approach is the use of the ROS system to log the inputs. Code instrumentation is done off-line. Koong et al. (12) describe an approach to structural testing of embedded system in wich the code is fully instrumented to allow for structural testing. This can be a problem for some types of embedded systems.

V. Use and validation

To illustrate the proposed approach, we conducted field testings involving a program to control the navigation of an autonomous vehicle using stereo vision. This program uses two images of stereo cameras and sends these images to an algorithm that returns a 3D image (point cloud) (1; 13). Considering the information of the coordinates x, y and z of such images, the program calculates whether there are obstacles and, if so, performs the deviation. The algorithm uses GPS information to know if it reached the target destination. No sensor is used and the actuators are the wheel and the engine.

The overall goal of the field testing was to get to a predetermined point deviating from obstacles, if any. The field testing was performed considering two scenarios: in a narrow and in a wide terrain. In the narrow terrain the vehicle was expect to follow a straight trajectory and obstacles (traffic cones) have been put on the way during the testing according to the tester intuition as the vehicle moved (S1 - see Figure 6). In the wide terrain, the vehicle should perform a curve and two cones have been previously placed in strategic places (S2) as shown in Figure 5.

Fig. 5. Path traveled by the autonomous vehicle during the field test

In this context, the program was called ObstacleAvoidance and it has several versions but we will use just one of them in this paper. The two test scenarios have been conducted within the Campus of a university. A log was generated from each of these field testings. Figure 6 presents object model of the example described, which is an instance of the meta-model presented in Figure 2.

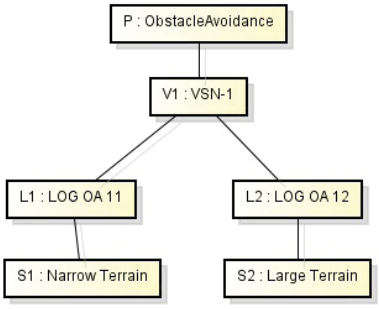


Fig. 6. An object model derived from the meta-model presented in Figure 2

Figure 7 shows an abstract vision of the coverage obtained after the field testing using scenarios S1 (top left) and S2 (top right). In the top part of the Figure 7 we can see eight classes comprising the whole project and inside them their methods. Two methods present a lower score of coverage, as can be seen by the lighter green. We choose to analyze deeply the one at the top rigth corner of the figure. The two diagrams at the bottom show a closer look at them.

Figure 8 shows partially the coverage of the control graph for both scenarios. We can see that an important part of the right graph has not been exercized by scenario S2, which is marked by the red color. Going back to the code we can see that this part of the code is related to the decision of

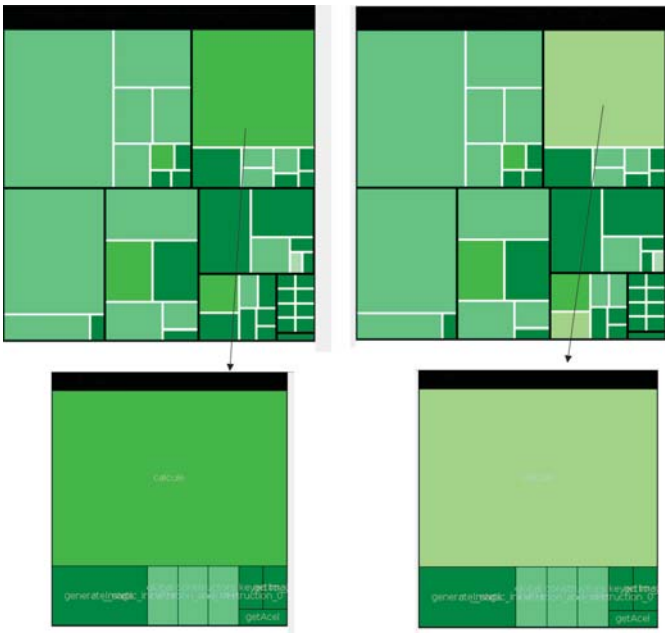


Fig. 7. Visualization of project and class coverage

deviating from an obstacle. It has not been exercized because the GPS was deactivated in this scenario, what changed the input data. Off course, we could create and visualize an aggregated version of all field testings of this version and them this part of the program would appear as green and the tester could be satisfied with this, as at least one testing covered that part of the code.

We can also see right bellow this part of the graph three decision nodes (in yellow) that have not exercized both its branches. We can conclude that these two scenarios have not been enough to reach a satisfactory coverage and new scenarios need to be created or the one already created need to be modified. In an aggregated graph these nodes would still appear as yellow indicating that these two field testing were not enough to exercized those nodes.

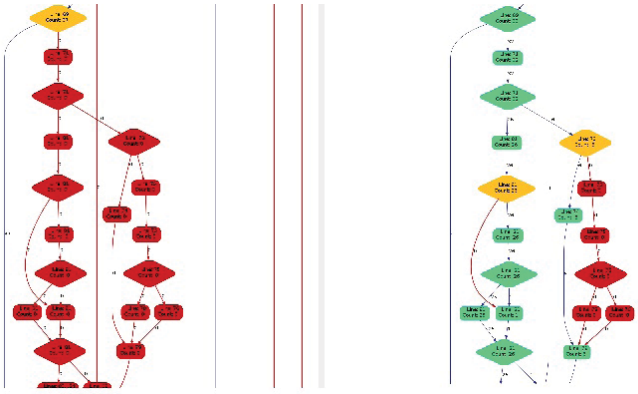


Fig. 8. Visualization of the coverage of a program for two different versions of the same program and two different scenarios.

A visual analysis as the one presented above in Figure 8, examining side by side the coverage of testings according to their scenarios or versions can also be done at the code level.

VI. Conclusions and Future Work

This paper proposed a model to support structural field testing of autonomous vehicles as well as a tool that implements this model. The model stores information of test scenarios which, in turn, can be performed by different versions of different programs. Each run of a test scenario for a version of a program can generate one or more logs. When performing the simulation of a scenario, we obtain the coverage obtained, the graph and the source code of the program colored according to the coverage achieved and a software module scheme that enables a more global view of the coverage of the program. Using the tool, the tester can analyze this information and think of new test scenario to cover parts of the test that have not been performed initially.

This tools is being used and validated by researchers and graduate students of our group of Mobile Robotic Systems and new reports and analysis are being developed. We devised two main important improvements to this environment in the near future. One of them is to be able to allow for dynamic, on-line acquisition and analysis. That is, the tester can go along with the autonomous vehicle while it is running and collect information on the command exercized and show coverage information that can be used to change the program or the scenario dynamically to better reach the tests objective.

The other improvement will be off-line. Using algorithms (e.g. genetic) for data set generation (14) we would like to use as input a log (or part of it) of a field testing and create a derived log satisfying a fitness function, like reaching a specific node or branch, or executing a particular command, in order to increase the coverage. This new log could then be used to simulate the behavior of the autonomous vehicle. Koong et al. (12) also explores data set generation in the proposal but use random testing.

Ackowledgments

The authors would like to thank the Brazilian funding agencies FAPESP, CAPES and CNPq and the INCT on Safety Embedded Systems for their support.

References

[1] L. C. Fernandes *et al.*, "Intelligent robotic car for autonomous navigation: Platform and system architecture," in *CBSEC 2012 - II Brazilian Conference on Critical Embedded Systems*, 2012, pp. 12 –17.

[2] C. Ebert and J. Salecker, "Guest editors' introduction: Embedded software technologies and trends," *Software, IEEE*, vol. 26, no. 3, pp. 14 –18, may. 2009.

[3] C. Urmson *et al.*, "Autonomous driving in urban environments: Boss and the urban challenge," *J. Field Robot.*, vol. 25, pp. 425–466, August 2008.

[4] F. W. Rauskolb *et al.*, "Caroline: An autonomously driving vehicle for urban environments," *J. Field Robot.*, vol. 25, no. 9, pp. 674–724, Sep. 2008.

[5] P. G. Trepagnier *et al.*, "Kat-5: Robust systems for autonomous vehicle navigation in challenging and unknown

terrain." *J. Field Robotics*, vol. 23, no. 8, pp. 509–526, 2006.

[6] C. Urmson *et al.*, "A robust approach to high-speed navigation for unrehearsed desert terrain," *Journal of Field Robotics*, vol. 23, no. 8, pp. 467–508, August 2006.

[7] G. Myers, *The Art of Software Testing*. John Wiley and Sons, 2004.

[8] S. Thrun *et al.*, "Stanley: The robot that won the darpa grand challenge: Research articles," *J. Robot. Syst.*, vol. 23, pp. 661–692, September 2006.

[9] ROS.org, "Documentation - ros," Online, 2013, available: http://www.ros.org - last access in em 27/02/2013.

[10] N. Terry, "trucov - the true c and c++ test coverage analysis tool," Online, 2011, disponível em http://code.google.com/p/trucov/ - Último acesso em 30/03/2012.

[11] Prefuse, "The prefuse visualization toolkit," Online, 2013, available: http://prefuse.org/ - last access in em 27/02/2013.

[12] C.-S. Koong *et al.*, "Supporting tool for embedded software testing," in *Proceedings of the 2010 10th International Conference on Quality Software*, ser. QSIC '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 481–487.

[13] C. C. T. Mendes, "Navigation of mobile robots using stereo vision," Master's thesis, ICMC/USP, São Carlos/SP - Brasil, 2012 (in Portuguese) http://www.teses.usp.br/teses/disponiveis/55/55134/tde-18062012-162436/en.php.

[14] P. McMinn, "Search-based software test data generation: a survey: Research articles," *Softw. Test. Verif. Reliab.*, vol. 14, pp. 105–156, June 2004.

Structural Testing of Exceptions Handling

Luciano Augusto Fernandes Carvalho, Vânia de Oliveira Neves, Paulo Cesar Masiero
Instituto de Ciências Matemáticas e de Computação - ICMC
University of Sao Paulo - USP
{luciano,masiero}@icmc.usp.br

***Abstract*—Exception handling mechanisms available in object-oriented languages are being increasingly used and account for about 8% of the lines of code in modern programs according to some surveys. Yet, this mechanism is among the least understood by programmers and the less tested. The difficulty of testing exceptions increases in the phase of integration testing (e.g. inter-class testing) because exceptions raised at a certain level and untreated can rise through the call hierarchy and also because aspects can insert exceptions not foreseen by the base program. This paper presents a proposal for structural integration testing of exception flows of Java and AspectJ programs. This is done by proposing changes to the classic control flow graph and by proposing new criteria based on the exception flow. Moreover, these proposals were implemented in a tool named JaBUTi/AJ.**

***Keywords*—exception handling, structural testing, tool, aspect-oriented.**

I. Introduction

Exception handling mechanisms have been incorporated into modern programming languages to raise and handle exceptional situations in the flow of program execution and to facilitate marking protected code regions. Sinha and Harrold (1) cite two studies to determine the frequency with which Java programs use exception handling constructs and thus assess the importance of these constructs in program testing and other activities. The conclusion of the first study was that 8.1% of the methods contains some form of exception handling and the second was that 23.3% and 24.5% of classes have the commands try and throw, respectively.

[1]

Despite the high frequency of occurrence, the exception handling mechanisms are generally poorly understood and are the lesser tested parts of the programs (2). This motivated research into these mechanisms and several authors proposed structural testing criteria for exceptions handling (1; 2), as well as algorithms for static analysis of OO programs. Integration testing is very important for exception handling mechanisms because an exception raised on a certain level of the execution flow and not treated, rises in the hierarchy of calls and may even reach the original caller unit. Motivated by the emergence of aspect-oriented programming and the observation that aspects can introduce unexpected exceptions in programs Rashid et al. (3) and Coelho et al. (4) investigated this problem. Exceptions introduced by aspects have greater chances of not being treated by the methods in each they are raised.

The objective of this paper is to present a proposal for structural integration testing of exceptions for Java and AspectJ programs. This is done by proposing changes to the integrated control flow graph and by means of testing criteria based on the exception flow. Moreover, these proposals have been implemented in an extension of the JaBUTi/AJ tool wich was proposed by Vincenzi et al. (5) to support structural testing of Java Program. We alo present a case study to validate the tool and the feasibility of using the proposed criteria. The results are discussed.

This paper is organized as follows. Section II presents a discussion of several papers related to this work. Section III presents our approach to structural integration testing of exception flows in OO and OA programs. Specifically, we show the graph created, the criteria defined and how they have been implemented in a tool. Section IV presents a case study conducted to test the proposal and the implementation. Section V presents our conclusions.

II. Related work

One of the first and most important proposals for structural testing of exception handling is that of Sinha and Harrold (2), which proposed six criteria for testing exception handling for Java programs. These criteria support integration test, whereas four are for control flow and two for data flow. They discuss the hierarchy of criteria and their relationships and also present a method to use in unit testing and integration testing. A continuation of this work suggests OO representations for programs in Java containing exception handling mechanisms by means of an integration graph, as well as algorithms to construct this graph (1) and produce several types of analysis.

Following this same line of work, Mao and Lu (6) proposed a new approach for integration testing of exception flow for C++ programs. An important contribution of that paper is to present a method to handle implicit exceptions, a subject treated by few authors. Mao and Lu (7) presented a prototype tool to support their approach. The authors of this article proposed ways of identifying implicit exception handlings but none can guarantee to identify all implicit exceptions, which remain a challenge for the test of exception flows.

Some testing criteria for Java programs have been proposed by Vincenzi et al. (5) for unit testing. The authors propose eight criteria for unit structural testing split evenly into exception dependent (ED), which represents the exception flow and exception independent (EI), which represents the regular flow. To support the use of these criteria, they have been implemented in the tool JaBUTi, which makes the generation of a control flow and data flow graph from the bytecode called Def-Use Graph (DUG) created from the program bytecode, generates testing requirements and calculates the testing coverage automatically. Subsequently, this tool has been extended to support testing

[1]The authors would like to thank the Brazilian funding agencies (FAPESP, CNPq) and INCT-SEC for their support.

aspect-oriented programs in AspectJ and a unity can now be a method or an advice (8). This work resulted in the initial version of the tool JaBUTi/AJ. Then, continuing this line of work Cafeo and Masiero (9) proposed and implemented integration testing in the tool JaBUTi/AJ, but did not address the mechanisms of exception handling, which is now proposed in this article.

III. Structural Integration Graph with Representation of Exception Flow

Our approach creates a representation of the exception flow using a conservative heuristic trying to represent virtually all possibilities of the exception flow in the integrated control/data flow graph, i.e., both explicit and implicit exceptions. Exception handling is not a local control structure. It consists of two basic and two optional elements: the element `try` block defines a scope that holds the code for which exception handling is being provided to; the element `catch` is responsible for capturing an exception and for its handling; the element `finally` is optional and defines a block of code that will be executed after the `try` block whether an exception is thrown or not and whether or not it is captured; and the fourth element is the command `throw`, which is responsible for raising explicit exceptions in the code.

Figure 1 shows an example of a code with exception handling implemented. The integrated control graph shown in Figure 2 is designed in the style of Sinha and Harrold. The graph shows the intra-method and inter-method normal and exception normal flows. The edge going from node (2).1.85 to node (1).2.7 represents an explicit exception flow for the command `catch` of line 24. Node (1).2.35 is isolated because it corresponds to the command `catch` of line 31 and there is not an explicit exception in the code that requires diverting the flow of execution to it. In our approach, the graph is actually integrated creating just one graph and, therefore, the flows of normal and exception intra-method and the regular inter-method flow of execution are represented as solid line arrows and only inter-method exception flows are represented by dashed arrows. When this flow crosses more than one hierarchical level, the destination node is the node where the exception is actually handled. The graphs shown in figures 3 and 2 are two variations of the ICFG of the method `main()` of the code shown in Figure 1 and will be better explained in the remainder of this Section.

```
1 public void  function1( )  {
2  int i=0,j=0;float k=0, sum=0;
3  float[] p = new float[20];
4  exception1 e1 = new exception1();
5  exception2 e2 = new exception2();
6  do{ k=cin[j++];
7      p[i]=(float) (1.6/k);
8      sum+=p[i];
9      i++;
10 }while(k <= 9999);
11 try{ if(sum>=0)
12         throw e1;
13     else
14         throw e2;
15 }catch(exception1 e){
16   System.out.println("Exception1");
17   throw e;}
18 catch(exception2 e){
19   System.out.println("Exception2");
20   } }
```

```
21 public void function2(){
22 try{
23     function1();
24    }catch(exception1 ep ){
25     if(ep.v > 0)
26      System.out.println
27      (ep.getMessage());
28     else
29      ep.printStackTrace();
30    }
31    catch(Throwable e){
32     System.out.println
33     ("Runtime Error");
34    }
35  }
```

```
36 public static void main
37 (String Args[]){
38  pex P = new pex();
39  P.function2(); }
```

Fig. 1. Sample Java code

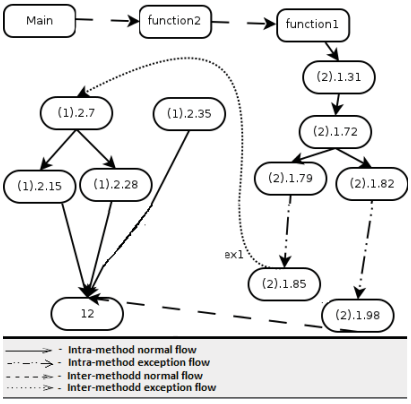


Fig. 2. Integrated graph

Notice in figure 3 that nodes with prefix (1) correspond to the method `function2` which is at level 1 in the hierarchy of call, nodes prefixed with (2) correspond to the method `function1` (level 2) and methods without prefix belong to the method `main` (level 0). Note also that there are two types of edges: one with continuous lines representing regular execution flow and other with a dashed line representing the exception execution flow. Some nodes contain the prefix (E), which indicates that they raise an explicit exception. These nodes are called `throw` nodes in this paper. It was also necessary to define how to identify and represent the elements of the exception flow. For the nodes `throw` of Java and AspectJ programs, where there is a command `throw` in the bytecode it is generated a command `athrow` corresponding to it. Thus, whenever a node contains `athrow`, it will be the last statement contained in that node, which implies that a node `throw` will never exit an output edge of the normal flow but only exception edges. Examples of `throw` nodes in figure 3 are (E).(2).179, (E).(2).182, and (E).(2).185.

The exception edges are generated in two different ways. The first case occurs when there is a block `try` and exception edges are generated from each node that is in the scope of the block `try` with destinations for all blocks `catch` and `finally` that refers to the block `try` to which they belong. In this case, all exception edges connect nodes at the same level as for example edges ((1).2.0, (1).2.7), ((1).2.0, (1).2.35), ((2).1.72, E.(2).1.85) and ((2).1.72, (2).1.98) of Figure 3*(a)*. It should be noted that none of these edges has as an output a throw node, which shows that they represent the exception flow and are generated implicitly. The second case occurs when there is a `throw` node, from which there are output exception edges to all nodes that can handle the exception raised, whether at the same level in which it has been raised or at any other level where there is a treatment for the exception raised. As an example of this case in Figure 3*(a)* we have all the edges that leave the nodes (E).(2).1.79, (E).(2).1.82 and E.(2).1.85. If it is not possible to handle the resulting exception the node is identified as an end node, thereby indicating that the flow of execution is interrupted at this point by an uncaught exception.

Five new criteria have been defined, related to the exception flow. For its definition, shown below, consider: T is a set of test cases for a program P whose graph is the integrated control flow/data graph integrated generated for the units of P and H is the set of paths exercised by T. A node x is included in H if H contains a path $(y_1, ..., y_n)$ such that $x = y_j$ for some j,

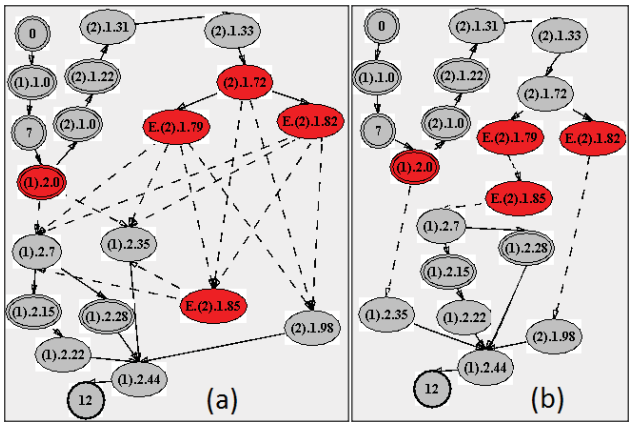


Fig. 3. Control flow graph of the method `main()`

Criteria	Requirements		
all-nodes-throw	(2).1.79; (2).1.82; (2).1.85		
all-edges-throw	((2).1.79,(1).2.35);	((2).1.79,(1).2.7);	((2).1.79,(2).1.85);
	((2).1.79,(2).1.98);	((2).1.82,(1).2.35);	((2).1.82,(1).2.7);
	((2).1.82,(2).1.85);	((2).1.82,(2).1.98);	((2).1.85,(1).2.35);
	((2).1.85,(1).2.7)		
all-nodes-ed	(1).2.35; (1).2.7; (2).1.85; (2).1.98		
all-edges-ed	((2).1.79,(1).2.35);	((2).1.79,(1).2.7);	((2).1.79,(2).1.85);
	((2).1.79,(2).1.98);	((2).1.82,(1).2.35);	((2).1.82,(1).2.7);
	((2).1.82,(2).1.85);	((2).1.82,(2).1.98);	((2).1.85,(1).2.35);
	((2).1.85,(1).2.7)		
all-nodes-ex	(1).2.35; (1).2.7; (2).1.79; (2).1.82; (2).1.85; (2).1.98		

TABLE I. TESTING REQUIREMENTS OF THE METHOD `MAIN()`

$1 \leq j \leq n$. Similarly, an edge (x_1, x_2) is included in *II* if and only if *II* contains an edge such that $x_1 = y_j$ and $x_2 = y_{j+1}$, for some j, $1 \leq j \leq n-1$. Consider also that whenever a node or edge is mentioned, it is a node or edge integrated, that is, it is a node that refers to a unit which is at a level j, for some j, $j \geq 1$, and if it is an edge it comes out of an integrated node . Accordingly, the criteria are defined as follows: **all-nodes-throw**, *II* satisfies this criteria if all nodes `throw` of an integrated graph is included in *II*; **all-edges-throw**, *II* satisfies this criteria if all edges (x_1, x_2), where x_1 is a node `throw` of an integrated graph, is included in *II*; **all-nodes-ed**, *II* satisfies this criteria if any input node in a block `catch` or `finally` of an integrated graph is included in *II*; **all-edges-ed:**, *II* satisfies this criteria if all integrated exception edge is included in *II*;**all-nodes-ex:**, *II* satisfies this criteria if every node `throw` and every input node in a block `catch` or `finally` of an integrated graph is included in *II*.

The proposed criteria are similar to those proposed by Sinha and Harrold (1). Some criteria are similar to the all-nodes-throw and all-throw criteria as both are designed to test the raising of explicit exceptions and the criteria all-nodes-ed and all-catch that are designed to test exception handling nodes. Besides that, in this work we have a criterion designed to test both raising and handling exceptions, the criterion all-nodes-ex. The proposal of Sinha and Harrold (1) also presents criteria for coverage of triples activation-deactivation, but the criteria proposed in this paper use this as exception edges instead as a requirement. Sinha and Harrold (1) also proposed criteria that address the use of exception variables, but in this work it was not created any such criterion, considering that these criteria are most relevant for exercising the data flow of the program than the exception flow. Other difference is that this work also allows the possibility of dealing with implicit exceptions, which they do not considered, such as the all-edges, which has as requirement besides the exception edges that leave throw nodes, exception edges leaving nodes that do not raise explicit exceptions.

We took a more conservative approach in the JaBUTi/AJ implementation, representing a greater number of possibilities for exception flows. This decision leads to a side effect that is the large number of exception edges that cam be created. Graphs with integrated representation of the exception flow are already naturally complex. Other issue is that in most cases, when there is a large number of exception edges leaving the same node, usually most of these edges may belong to infeasible paths.

Knowing that it can be difficult to interpret the integrated graph, we implemented in JaBUTi/AJ a few viewing options to facilitate the tester activity. One of these viewing options allows the display of only the edges required by the selected criteria. In figure 3*(a)*, for example, if the selected criterion is all-throw-edges, the edges ((1).2.0 (1).2.7), ((1).2.0 (1).2.35), ((2).1.72, (E).(2).1.85) and ((2).1.72. (2).1.98) will be invisible. If the criterion all-edges-ed is selected, all exception edges would be visible, and if a criterion that only has regular flow requirements, as all-edges-integrated-independent-of-exception, all exception edges will be invisible.

The tool JaBUTi/AJ allows the tester to indicate which test requirements were identified as infeasible. To help testers an option to automatically attempt to identify some edges exception as unfeasible was also implemented. We created an option of visualization to hide these requirements when viewing the graph. Figure 3*(b)* shows an example of the graph display with this option and it can be noted that the number of edges shown is smaller than the original graph 2 and diminishes from 14 to just 4 exception edges. With this option the display of exception flows in the graph of Figure 3*(b)* is very similar to the graph of Figure 2, with only one more exception edge, which is the ((1).2.0, (1).2.35) and refers to raising an implicit exception.

Aspects can generate changes in the exception flow, and when these changes are not done properly they can introduce faults (4). Aspects can interfere with the exception flow in different ways. The most obvious is that any advice may raise an exception. Moreover, advices of type `after` can disable an exception by means of the activation of a new exception. Exceptions can also be softened by aspects using the clause `declare soft`. Furthermore, advices of type `around` can introduce exception handlings. For example, exception handling can be introduced by an advice `around` to implement a try block, which uses a clause `proceed`.

In figure 4 we show an ICFG of the method `main()` for the code shown in Figure 1, which is an advice of type `after throwing` crosscutting function1. The change generated by the introduction of this advice modifies the exception flow and we can see the differences comparing figures 4 and 3. The latter has edges from nodes (E).(2).1.79, E.(2)1.82 and E.(2)1.85) to nodes (1).2.7 and (1).2.35, but with the addition of advice in figure 4 this six edges disappear and are created edges from nodes (E).(2).1.79, E.(2)1.82 and E.(2)1.85) to node (2).1.109, which represents the beginning of an advice, and edges from

node E.(2).1.119 (represents the end of an advice) to nodes (1).2.7 and (1).2.35. It mean that all exceptions raised by function1 that leave it to be redirected to the advice, which causes the execution of its code and finally the exception be raised again.

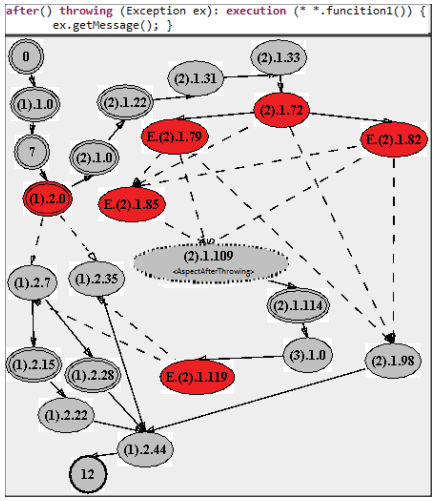


Fig. 4. Example of interactions between aspects and exception flows

IV. Case study

We conducted a case study involving the testing criteria implemented in JaBUTi/AJ and also some of the criteria already implemented in the tool to assess a testing strategy for structural testing of OO and AO programs in Java and AspectJ. The aim of this study was also to investigate regular and exception flows and the effort required to test a program, besides testing our implementation of the exception criteria and how they behave in relation to aspects. This study was conducted considering the unit under test and the integrated graphics of this unit and all the units called to the deepest level and for each unit were applied all the criteria proposed in this work and also the criteria proposed by (9).

For each application the following information was collected : number of classes (`#Cs`), number of aspects (`#As`), number of lines of code excluding blank lines and comments (`#Loc`), total number of program units (`#U`), number of methods (`#M`), number of advices (`#Ad`), total number of test cases designed to reach the 100% coverage (`#Tc`), number of test cases designed to cover integration testing up to the highest level of depth found to reach the 100% coverage (`#Tcd`) and maximum depth (`#Md`). Such information is shown in Table II.

Program	#Cs	#As	#Loc	#U	#M	#Ad	#Tc	#Tcd	#Md
Music	10	2	269	42	40	2	9	3	3
Telecon	6	3	229	33	26	7	11	1	3
VM-OO	12	0	247	14	14	0	29	9	3
VM-OA	12	3	417	33	17	16	37	12	7

TABLE II. Data collected in the case study

An analysis of the entire process of the case study, carefully evaluating each graph along with its bytecode and along with the subgraphs from which it was combined, allowed us to validate the implementation and verify the integrity of the tool JaBUTi/AJ after the changes made in this work. This study allowed us to make a preliminary analysis of the cost of applying the strategy for structural integration testing of OO and AO programs to Java and AspectJ. The average number of test cases necessary to cover each Class or Aspect was 1.75 and the average number of test cases to cover each unit was 0,69, this results showed that the number of test cases required for our approach is usually not high.The case study also demonstrated the applicability of the approach for small programs, but it cannot be generalized to medium and large programs.

V. Conclusions and future work

In this paper we proposed a control graph representation for integration testing of OO and OA programs aiming at testing exception mechanisms existing in this type of language. Five exception criteria for integration testing have also been proposed and the JaBUTi/AJ was extended to support these criteria. We are planning to conduct further experiments with other programs of different sizes and complexities to further assess the proposal.

References

[1] S. Sinha and M. Harrold, "Analysis and testing of programs with exception handling constructs," *IEEE Transactions on Software Engineering*, vol. 26, no. 9, pp. 849–71, sep 2000.

[2] ——, "Criteria for testing exception-handling constructs in java programs," *IEEE Transactions on Software Engineering*, pp. 265–74, 1999.

[3] A. Rashid, T. Cottenier, P. Greenwood, R. Chitchyan, R. Meunier, R. Coelho, M. Sudholt, and W. Joosen, "Aspect-oriented software development in practice: Tales from aosd-europe," *Computer*, vol. 43, no. 2, pp. 19–26, feb. 2010.

[4] R. Coelho, A. von Staa, U. Kulesza, A. Rashid, and C. Lucena, "Unveiling and taming liabilities of aspects in the presence of exceptions: A static analysis based approach," *Information Sciences*, vol. 181, no. 13, pp. 2700–20, 2011.

[5] A. M. R. Vincenzi, M. E. Delamaro, J. C. Maldonado, and W. E. Wong, "Establishing structural testing criteria for java bytecode," *Software Practice and Experience*, vol. 36, pp. 1513–41, Nov. 2006.

[6] C.-Y. Mao and Y.-S. Lu, "Improving the robustness and reliability of object-oriented programs through exception analysis and testing," in *10th IEEE International Conference on Engineering of Complex Computer Systems*, june 2005, pp. 432–39.

[7] C. Mao and Y. Lu, "Cpptest: A prototype tool for testing c/c++ programs," in *Availability, Reliability and Security, 2007. ARES 2007. The Second International Conference on*, april 2007, pp. 1066 –1073.

[8] O. A. L. Lemos, A. M. R. Vincenzi, J. C. Maldonado, and P. C. Masiero, "Control and data flow structural testing criteria for aspect-oriented programs," *Journal of Systems and Software*, vol. 80, no. 6, pp. 862 – 882, 2007.

[9] B. Cafeo and P. C. Masiero, "Contextual integration testing of object-oriented and aspect-oriented programs: A structural approach for java and aspectj," in *25th Brazilian Symposium on Software Engineering.*, sept. 2011, pp. 214–23. DOI: 10.1109/SBES.2011.12.

A Hybrid Coverage Criterion for Dynamic Web Testing

Yunxiao Zou[1], Chunrong Fang[1], Zhenyu Chen[1*], Xiaofang Zhang[2] and Zhihong Zhao[1]
[1]State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China
[2]School of Computer Science and Technology, Soochow University, Suzhou, China
[*]Email: zychen@software.nju.edu.cn

Abstract

Testing criterion is a fundamental topic of software testing. A criterion is important to evaluate and drive a testing method. Code coverage is widely used in software testing, due to its simple implementation and effectiveness. Dynamic web techniques have been used to improve the usability and user experience of applications. However, it brings some new challenges for testing. Dynamic web applications have richer iterations between client-side and server-side, such that code coverage is difficult to capture these complex iterations for sufficient testing.

In this paper, we present a novel coverage criterion - hybrid coverage. A hybrid coverage criterion which combines statement coverage and HTML element coverage, covers both client-side and server-side features. The experimental result shows that the hybrid coverage can detect 22.2%-48.1% more bugs than statement coverage, 7.9%-57.1% more bugs than element coverage.

Keyword: Dynamic Web, Coverage Criteria, Web Crawler, Test Case Prioritization

1 Introduction

Web applications are one of the most widely used types of software. And a new technology, acting as the core of the web 2.0, is AJAX (Asynchronous JavaScript and XML) which are changing the way of developing web applications significantly.

Testing is a major way to ensure the quality of software. In order to evaluate and drive testing methods, some coverage criteria are introduced to calculate the percentage of program elements that the test sets exercise. Statement coverage is widely used because of its ease of implementation and its low overhead on the execution of the program under test. However, statement coverage is not sufficiently capable to cover rich features of web 2.0 enabled web applications. First, a large number of client-side functions and contents in web applications are generated dynamically by the server-side code (e.g., PHP). The same code can produce many different web contents via combining different server states, session variables and other data. Test cases with the same statement coverage information may generate different front-end contents. Second, the new AJAX technology, which is the combination of JavaScript and DOM tree manipulation with asynchronous server communications enables powerful client-side execution capabilities. Server-side statement coverage is not appropriate to cover these rich features on client-side and interactions between server-side and client-side.

For web applications, the basic unit of displaying contents and receiving user input events is standard HTML element (in this paper, HTML element may be abbreviated as element). A majority of user interactions with the web applications are interactions with the event-able HTML elements. These front-end elements can be suitable indicators for testing coverage. The structure of the web application user interface (UI) which contains HTML elements can be created by web crawlers. However, coverage criterion merely based on HTML elements has its own shortages. It lacks execution information of server-side scripts, which is an important runtime feature of dynamic web applications.

Intuitively, each of the mentioned coverage measures one dimension of the web application. A hybrid coverage based on both of them can measure multiple dimensions of web application features. In this paper, we present a novel hybrid coverage for dynamic web applications. It combines statement coverage and element coverage to represent both server-side and client-side runtime conditions of the test sets. The experiment result shows that hybrid coverage can drive a more effective testing.

The main contributions of this paper are as follows.

1. A novel coverage criterion: hybrid coverage which counts both the code executed and the HTML elements accessed by the test sets which represents both server-side and client-side conditions.

2. An empirical study is conducted to evaluate the effectiveness of hybrid coverage. Two web applications are

used to compare hybrid coverage with coverage criteria such as statement coverage and element coverage. The result shows that our approach can drive a more effective testing for dynamic web applications.

The rest of paper is organized as follows. Section 2 defines hybrid coverage and element coverage, and proposes the detail of our approach, including methods of calculating the element coverage and hybrid coverage. The experiment design and the result analysis are presented in Section 3. We discuss the related work in Section 4. Section 5 are the conclusion and future work.

2 Approach

The element coverage and hybrid coverage definitions are presented as follows:

Definition 1 (element coverage) *A set T of test cases satisfies the element coverage criterion if and only if for each element $e \in E$, there is at least one test case $t \in T$ such that e is accessed by t, where E denotes the element set of the application under test.*

In the definition, an element is accessed by a test case, which means that the element is displayed or receives events on the browser during the execution of the test case. The element set is obtained from the web UI model which is created by the web crawler [3].

Definition 2 (hybrid coverage) *A set T of test cases satisfies the hybrid coverage criterion if and only if for each element $e \in E$, there is at least one test case $t \in T$ such that e is accessed by t, and for each statement $s \in S$, there is at least one test case $t \in T$ such that s is executed by t, where E denotes the element set, and S denotes the statement set of the application under test.*

Our hybrid coverage is built on statement coverage and element coverage. In the current state, there are mature tools to calculate statement coverage for web applications. However, to our knowledge, there are no ready-to-use tools to obtain element coverage data for the web application under test.

We implement a framework to obtain the element coverage of test cases. In the framework, a web crawler dynamically navigates on the web application under test to generate a web UI model to represent the front-end structure of the web application. On the other hand, test cases are executed and specific test data are collected at runtime. And the element coverage information are calculated from the web UI model and the collected data.

We utilize the Crawljax[1] to generate the front-end model. Because to the best of our knowledge, it is the only currently available tool to support AJAX-based web sites. The data (e.g., user name and password) needed to access hidden web content are provided to the crawler. The output of the crawling process is a state graph with runtime DOM tree contents contained in state vertices.

Having got the web front-end model in the previous step, the framework performs the coverage calculation of the test suite using the web UI model and the test data collected in the test automation runtime environment. For collecting test data, the framework runs the test cases in the test suite on the web application and collects data from the DOM tree in the browser after executing each test command in the test case. Our technique records three attributes (id, class and tag name) and XPATH value of the HTML elements which receive events from each test command, and it also records the same kinds of attributes and XPATH for the all the HTML elements displayed in the DOM tree after executing each test command. The data collected are used for calculating the coverage information for each test case.

The data collected in the previous step are used for identifying and matching the accessed HTML elements in the web UI model and computing the element coverage. The tracked attributes and XPATH are used for matching. Each of above properties has to be exactly matched. If there are multiple matches in the model, the first match is selected. After matching the HTML elements between the elements in the collected data and in the web UI model, the set of accessed HTML elements of test cases can be obtained.

For statement coverage, we utilize a combination of the Xdebug[2] and PHPUnit[3]. The tool chain can generate test report which contains statement coverage information. Hybrid coverage can be easily computed with element and statement coverage information.

3 Experiment

To explore the effectiveness of different coverage criteria, a test case prioritization experiment is conducted. We evaluate the usefulness of different coverage criteria by comparing the fault-detect capabilities of the test case sets which are prioritized via different coverage-based strategies.

3.1 Experiment Subjects

Due to the considerable manual cost to create web automation test scripts, we chose two PHP web applications, schoolmate[4] and timeclock[5]. Table 1 shows some information of these subjects.

[1] http://crawljax.com

[2] http://www.xdebug.org

[3] http://www.phpunit.de

[4] http://sourceforge.net/projects/schoolmate/

[5] http://timeclock.sourceforge.net

We recruited a team consisting of both senior undergraduate and graduate students to create test cases for subject programs. They created test automation scripts using Selenium[6] according to the specification requirements. Each member of this team submitted test cases and a document of detected-bug description, along with a test report. Table 1 lists some information of the test cases, and the number of detected bugs. The bugs (defects) are real bugs in the programs and include both functional and representation bugs.

Table 1. Subject Programs

program	bugs	tests		version	LOC	files
		fail	pass			
schoolmate	23	44	151	1.5.4	8181	63
timeclock	13	13	142	1.0.2	20789	62

3.2 Experimental Procedure

In this sub-section we present a description in detail of the experiment procedure. We first initialize the test pool, which contains all the test cases shown in Table 1. The initial set of selected test cases is empty, and at every round of the procedure, one test case is selected from the test pool. For each selection, the procedure traverses the unselected test cases in the test pool, and calculates the fitness value as the selection metric for each candidate test case. The candidate test case with maximum fitness is selected. If there are multiple qualified candidate tests, we pick up one test case randomly among them. If the coverage of the selected tests reaches the maximum coverage (That is, the coverage of the test pool), the coverage data are cleared and the procedure continues. The procedure will end if the end conditions are met.

One of the end conditions is that all bugs are detected by the selected test cases. However, in the industrial practices, Re-executing a large number of test cases is too time-consuming in regression scenarios. Therefore, we set the maximum number of selected test cases. For our experiment, the maximum number of selected test cases is one third the size of the test pool. The procedure should be sampled for many times for there exists randomness in the selection phase. The guidance in [1] suggests the sampling frequency should be more than 30. Following this suggestion, the procedure will be performed 100 times for each combination of different subject programs and selection strategies.

3.3 Selection Strategies

Test case with maximum fitness value is chosen from the test pool at each round of the procedure. Like a multi-objective function in [5]. The basic idea is that test case with high contribution of coverage points and low executing cost is selected at each round. The coverage contribution of candidate test case to the selected test case set can be measured by *Additional Coverage Set* (ACS). The additional coverage set can be computed by the formula $ACS(t, Cover\_Info(T)) = CS(t) - CS(Cover\_Info(T))$. We substitute $CS(Cover\_Info(T))$ for $CS(T)$ because the coverage information may be cleared in the experiment procedure. The executing cost of test case is measured by the cost of initializing the test case plus the cost of executing test commands in the test cases. In our measures, the cost of test case initializing is a constant value, we set it with 10 according to experience; the command-executing cost is approximated by the number of test commands in the test case. The general fitness formula is defined as follows: $Fitness(t, T) = \frac{|ACS(t, Cover\_Info(T))|}{Cost(t)}$

We define the fitness formulas for the statement coverage, element coverage, and hybrid coverage, respectively. The formulas are defined according to the general formula, with the substitution of the information and calculation of each specific coverage for the information and calculation of the general coverage.

3.4 Result And Analysis

An effective strategy can detect a large number of faults with a small number of test cases. That means an effective strategy has high fault-detect capability with low cost. We measure this feature with fault-detect rate for each combination of subject programs and selection strategies. For each sample of test prioritization, we record the fault-detect rate, which means the number of detected bugs at any given number of selected test cases. Then we calculate the average rates of all 100 samples to get the final results. Figure 1 and 2 illustrate the fault-detect rates of three strategies on two subject programs, schoolmate and timeclock.

From the figures, We can see that the hybrid-coverage-based strategy can detect more bugs than the statement-coverage-based and element-coverage-based strategies. And whether element coverage performs better than statement coverage is dependent on the subject programs. On schoolmate, the hybrid coverage can detect 22.2% more bugs than statement coverage, 57.1% more bugs than element coverage. On timeclock, the hybrid coverage can detect 48.1% more bugs than statement coverage, and 7.9% more bugs than element coverage.

4 Related Work

There are some additional coverage criteria for web applications. Most of them are page-based coverage cri-

[6] http://seleniumhq.org

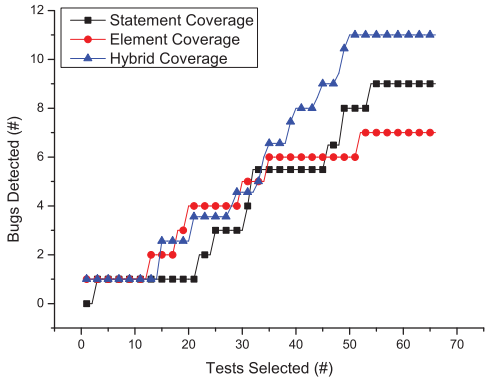


Figure 1. Fault-detect Rates on Schoolmate

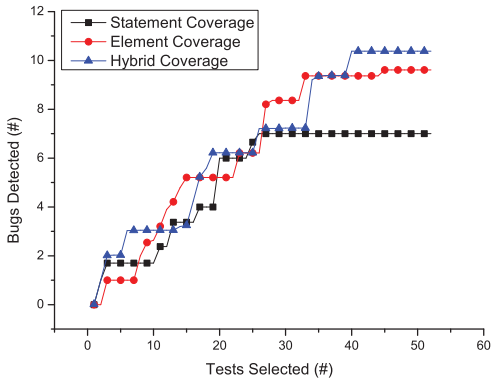


Figure 2. Fault-detect Rates on Timeclock

teria. Sampath et al. [4] presented a dynamic-generated page-level coverage criteria for web applications. In their method, the coverage criteria are based on URLs (uniform resource locator). But AJAX-based web sites are not based on multi-page paradigm in which every page has one unique URL [3]. The presented coverage criteria in [4] may not be effective on these AJAX-based web applications. In [6] Tonella et al. developed an UML-based model for web applications and extended the model to include server pages. In their method, multiple entities are grouped for one server-side page outcome using static analysis. Generating each possible outcome for every server page can quickly lead to a model of impractical size. Di Lucca et al. [2] also developed a UML-based test model and tools for the evaluation and automation of testing web applications. They consider that pages of the web application as components should be tested at the unit level, but the method to generate the test model seems not to be well presented.

5 Conclusion and Future Work

In this paper, we present a novel coverage criterion for web application testing. Element coverage is used for measuring the set of HTML elements accessed by the test cases. Based on this element coverage and the traditional statement coverage, we present a hybrid coverage to cover both client-side and server-side features. An experiment evaluates the usefulness of the hybrid coverage criterion.

Our future work is to build a web UI model with high completeness of elements. Element coverage is an useful complement to the traditional code coverage. But the possible incompleteness of the underlying model may adversely affect the performance of element coverage. Static analysis seems another way to build the underlying model.

6 Acknowledgement

The work described in this article was partially supported by the National Natural Science Foundation of China (61103045, 61003024, 61170067).

References

[1] A. Arcuri and L. Briand. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In *ICSE'11*, pages 1–10, 2011.

[2] G. Lucca, A. Fasolino, F. Faralli, and U. Carlini. Testing web applications. In *ICSM'2002*, pages 310–319, 2002.

[3] A. Mesbah, A. van Deursen, and S. Lenselink. Crawling ajax-based web applications through dynamic analysis of user interface state changes. *ACM Transactions on the Web*, 6(1):3, 2012.

[4] S. Sampath, E. Gibson, S. Sprenkle, and L. Pollock. Coverage criteria for testing web applications. Technical report, University of Delaware, 2005.

[5] W. Sun, Z. Gao, W. Yang, C. Fang, and Z. Chen. Multi-objective test case prioritization for gui applications. In *SAC'13, to appear*, 2013.

[6] P. Tonella, F. Ricca, E. Pianta, and G. C. Evaluation methods for web application clustering. In *WSE'03*, pages 33–40, 2003.

Towards the Effectiveness of a Variability Management Approach at Use Case Level

Anderson Marcolino, Edson Oliveira Junior, Itana Gimenes
State University of Maringá - Maringá-PR, Brazil
Email: andersonmarcolino@gmail.com, {edson, itana}@din.uem.br

José Maldonado
University of São Paulo - São Carlos-SP, Brazil
Email: jcmaldon@icmc.usp.br

***Abstract*—Software product line (PL) is an approach focused on a systematic software reuse that has been successfully applied to specific domains. One of its essential activities is the variability management to which there are several existing approaches, including the UML-based SMarty approach. Although there are several variability management approaches for PL, there is a need to demonstrate the effectiveness of such approaches for industry adoption. Therefore, this paper presents an experimental study that aims to gathering evidence of the SMarty approach effectiveness at use case level taking into consideration a consolidated and well-known UML-based variability management method, named PLUS. The experimental study provided evidence that SMarty is an effective approach for managing variability at use case level.**

I. Introduction

The software product line (PL) approach has gained increasing attention in recent years due to the competition in the software development segment [1]. Its main objective is the derivation of products for a specific domain. Such an approach comprises a set of essential activities, such as variability management, which is a key issue for the success of PLs. Several approaches for variability management have been proposed in the literature, as pointed out by Chen et al. [2].

Amongst existing variability management approaches are SMarty [3] and the PLUS method [4]. SMarty aims to manage variabilities in UML models supported by a profile and a set of guidelines for applying such a profile to use cases, classes, components, activities, and sequence models, as well as to packages. PLUS is a well-known method that allows explicit modeling of common and variable features supported by UML extensions for use case and class models.

These approaches are promising taking into consideration the variability management research field. However, their effectiveness was not experimentally analyzed, which can make it feasible for technology transfer to industry. Therefore, this paper presents an experimental study to gathering initial evidence with regard to the effectiveness of the SMarty approach by targeting UML use case models for a given PL. Use case models play an essential role in PL by linking features to lower-level models of a PL archicteture, such as classes and components, taking into account important traceability issues. The following use case relationships are taking into consideration: communication, include, extend, and dependency [3].

The remainder of this paper is organized as follows: Section II presents essential concepts with regard to variability management, the SMarty approach and the PLUS method; Section III presents the planning, execution and analysis and interpretation of this experimental study; and Section IV presents conclusion and directions for future work.

II. Background

A. Variability Management

Variability management is an essential PL activity for the derivation of specific products for a given domain. It brings out important benefits, such as, increases the reusability of the PL core assets, while decreases the time to market and justify the return on investment (ROI).

There are four main concepts taking into consideration for variability management, which are:

- **Variability**, according to Pohl et al. [1], is "the ability of a software or artifact to be changed, customized or configured for use in a particular context."
- **Variation Point** is the resolution of variabilities in generic artifacts of a PL. According to Jacobson et al. [5], "a variation point identifies one or more locations at which the variation will occur." Basically, a variation point answers the question: What varies in a PL? [1].
- **Variant** represents the possible elements through which a variation point may be resolved. Basically, a variant answers the question: How does a variability or a variation point vary in a PL? [1].
- **Variant Constraints** state the relationships between two or more variants to resolve a variation point or a variability. For instance, a PL manager decides not to offer certain combinations of mutually exclusive variants for a set of products. Thus, a mutually exclusive constraint needs to be defined for these variants.

The relevance of the variability management activity for PLs has been gained attention of many researches, as we can see in several existing studies in the literature [2], [4], [6]–[8].

Several existing variability management approaches do not make it clear how to identify, represent and trace variabilities in different artifacts [2], specially those based on UML models. This kind of approach most takes into account stereotypes and tagged values for representing PL variabilities. However, they fail on presenting the rationale on how to apply such

stereotypes and their relationships. Industry needs evidence on the effectiveness of these approaches to make their adoption feasible.

In order to provide a more precise UML-based approach for variability management, we have been developed the SMarty approach [3], [9], which is supported by a profile and a set of guidelines for applying its stereotypes and relationships. However, we need to gathering initial evidence with regard to its effectiveness by means of an experimental study. Therefore, the Gomaa's widely-known PLUS method [4] was chosen to perform such an experimental study. Thus, next sections present the PLUS method and the SMarty approach essential concepts.

B. The PLUS Method

The *Product Line UML-based Software Engineering* (PLUS) method, proposed by Gomaa [4], allows its integration with other software process models, such as, the unified process (UP) development.

Gomaa proposes several PL activities for requirement, analysis and design. The requirement activity encompasses PL scope definition, use case modeling and feature modeling. The analysis activity is composed by: static modeling, object construction, dynamic modeling, finite state machine and feature/class dependency modeling.

The PLUS use case modeling activity aims to explicitly model commonalities and variabilities. PLUS provides a set of concepts and techniques to extend UML-based design methods and processes for single systems to handle PLs.

PLUS does not provide a definition of an UML profile, thus there is no explicit meta attributes and classes for the variability modeling activity. PLUS uses stereotypes to provide identification of variation points and variants, in which several of them are specific to certain UML models. The rationale with regard to the use of such stereotypes is twofold: forward evolutionary engineering and reverse evolutionary engineering.

The stereotypes proposed by the PLUS method to represent variabilities in use cases are as follows:

- ≪kernel≫ - used to represent a mandatory use case, which is always selected for PL specific products;
- ≪optional≫ - used to represent use cases that might be present in a PL specific product;
- ≪alternative≫ - used to represent a mutually exclusive relationship between use cases.

PLUS is a well-known method, thus this is the main reason for its selection for this study, as it can be observed in the studies of Bragança and Machado [6], Gomaa [4], Korherr and List [7], Ziadi et al. [8], and Chen et al. [2].

C. The SMarty Approach

SMarty [3] is an approach for UML **S**tereotype-based **M**anagement of V**ar**iabili**ty** in PL. It is composed of an UML 2 profile, the *SMartyProfile*, and a process, the *SMartyProcess*.

SMartyProfile contains a set of stereotypes and tagged values to represent variability in PL models. Basically, *SMartyProfile* uses a standard object-oriented notation and its profiling mechanism [10] both to provide an extension of UML and to allow graphical representation of variability concepts. Thus, there is no need to change the system design structure to comply with the PL approach.

SMartyProcess is a systematic process that guides the user through the identification, delimitation, representation, and tracing of variabilities in PL models. It is supported by a set of application guidelines as well as by the *SMartyProfile* to represent variabilities.

The **SMartyProfile** comprises the following stereotypes, which can be applied to UML use case, class, component, activity, and sequence models, as well as it supports the package merging UML mechanism:

- ≪**variability**≫ represents the concept of PL variability and is an extension of the metaclass `Comment`;
- ≪***variant***≫ represents the concept of PL variant and is an abstract extension of the metaclasses `Actor`, `UseCase`, `Interface`, and `Class`. This stereotype is specialized in four other non-abstract stereotypes which are: ≪mandatory≫, ≪optional≫, ≪alternative_OR≫, and ≪alternative_XOR≫.
- ≪**mandatory**≫ represents a compulsory variant that is part of every PL product;
- ≪**optional**≫ represents a variant that might be selected to resolve a variation point or a variability;
- ≪**alternative_OR**≫ represents a variant that is part of a group of alternative inclusive variants. Different combinations of this kind of variants may resolve variation points or variabilities in different ways;
- ≪**alternative_XOR**≫ represents a variant that is part of a group of alternative exclusive variants. This means that only one variant of the group can be selected to resolve a variation point or variability;
- ≪**mutex**≫ represents the concept of PL variant constraint and is a mutually exclusive relationship between two variants. This means that when a variant is selected another variant must not be selected; and
- ≪**requires**≫ represents the concept of PL variant and is a relationship between two variants in which the selected variant requires the presence of another specific variant.

III. The Experimental Study

This study is characterized as a quasi-experiment [11] that relaxes the conditions imposed by probability distributions and statistical inferences for the population. Therefore, we performed the non-equivalent grouping method, considering that the population distribution was not random (discussed in Section III-E).

A. Definition

The goal of the experiment was to **compare** the PLUS method and the *SMarty* approach, **for the purpose of** identify the most effectiveness, **with respect to** the capability of identification and representation of variabilities in Software Product Line use case models, **from the point of view of** software product line architects, **in the context of** graduate students and lecturers of the Software Engineering area from

the State University of Maringá (UEM), Federal Technological University of Paraná (UTFPR), and Federal University of Amazonas (UFAM).

B. Planning

1) **Local Context:** a PL for Electronic Commerce (e-commerce), proposed by Gomaa [4], was taken into consideration to apply the PLUS method and the SMarty approach aiming the representation of variabilities in use case models.
2) **Training:** subjects were trained with regard to essential concepts of PL and variability and use case model variability identification and representation using PLUS or SMarty.
3) **Pilot Project:** a pilot project was performed for evaluating the study instrumentation taking into account a small sample of graduate students and a lecturer of software engineering. Thus, corrections on the instrumentation were made based on the pilot project results. Note that the pilot data was not taken into consideration by the overall experimental study data analysis.
4) **Selection of Subjects:** the subjects must be graduate students, lecturers or practitioners of the software engineering area with at least minimal knowledge in modeling use cases. In addition, after the training sessions, each subject must be familiar with the essential variability management concepts (Section II-A).
5) **Instrumentation:** every subject was giving the following documents:
   - the consent form to the experimental study;
   - a characterization questionnaire, in which the subjects must indicate their academic background, area of expertise and experience, their level of experience with the UML notation and the PL approach; and
   - the description of the e-commerce PL and its use case model with no variabilities represented.

   Subjects were splitted into two groups. One group focused on the X approach (the PLUS method) and one group focused on the Y approach (the SMarty approach). One group was trained to identify and represent variabilities according to the X approach. The other group was trained to identify and represent variabilities according to the Y approach.
6) **Hypothesis Formulation:** the following hypothesis were tested in this study:
   - **Null Hypothesis** (H_0): both X and Y approaches are equally effective in terms of representing variabilities in use case models.
     H_0 : μ(*effectiveness(X)*) = μ(*effectiveness(Y)*);
   - **Alternative Hypothesis** (H_1): X approach is less effective than Y approach.
     H_1 : μ(*effectiveness(X)*) $<$ μ(*effectiveness(Y)*); and
   - **Alternative Hypothesis** (H_2): X approach is more effective than Y approach.
     H_2 : μ(*effectiveness(X)*) $>$ μ(*effectiveness(Y)*).
7) **Dependent Variables:** the effectiveness calculated for each variability management approach (X and Y) as follows:

$$\textit{effectiveness(z)} = \begin{cases} nVarC, & \text{if } nVarI = 0 \\ nVarC - nVarI, & \text{if } nVarI > 0 \end{cases}$$

   where:
   - z is the variability management approach
   - *nVarC* is the number of correct identified variabilities according to the z approach
   - *nVarI* is the number of incorrect identified variabilities according to the z approach
8) **Independent Variables:** the variability management approach, which is a factor with two treatments (X and Y) and the e-commerce PL, which is a variable with a prefixed value.
9) **Qualitative Analysis:** aims to evaluate the results obtained in this study with respect to the results obtained by means of descriptive statistical analysis, based on the effectiveness obtained from the resolution of the use case variability model by each subject, according to the X and Y approaches.
10) **Random Capacity:** the selection of the subjects was not random within the universe of the volunteers as this was quite restricted. The random capacity took place at the assignment of the variability management approach (X or Y) to each subject.
11) **Block Classification:** because the application of two different approaches to represent variability in use case models, it was performed the random sampling, where the population was divided into two blocks, one for the X approach and one for the Y approach.
12) **Balancing:** tasks were assigned in equal numbers to a similar number of subjects.
13) **Review Mechanism:** for reviewing the study analysis it was used the calculation of the effectiveness for each treatment.

C. Execution

1) **Selection of Subjects:** it was selected for this study 21 graduate students and 3 lecturers of the Software Engineering area.
2) **Instrumentation:** the main assessment tool was the e-commerce use case model with variabilities represented according to the X and Y approaches. The main task for each subject was reading and understanding the e-commerce PL overview. Then, the subjects annotated variabilities in the e-commerce use case model.
3) **Participation procedure:** standard procedures were adopted for each subject participation, which are:
   a) the subject attends the place where the study was conducted;
   b) the experimenter gives the subject a set of documents:
      - the experimental study consent form;
      - the characterization questionnaire;

- essential concepts on variability management in PL; and
- the description of the e-commerce PL.

c) the subject reads each given document;
d) the experimenter explains the given documents;
e) the experimenter randomly associates each subject to the X or Y approach;
f) the experimenter trains the subjects on the respective approach;
g) the subject reads and clarifies possible doubts about his/her assigned approach; and
h) the subject identifies and represents variabilities in the e-commerce use case model according to his/her given approach.

4) **Execution:** collected data is presented in Table I and analyzed using appropriate statistical methods, which are properly discussed in Section III-D. For each subject ("`Subject #`" column), it was collected the following data for his/her given approach: the number of correct and incorrect identified and represented variabilities; and the effectiveness calculation.

D. Analysis and Interpretation

Based on the results obtained by analyzing the application of the PLUS and SMarty to the e-commerce PL, the following steps were taken:

- analyze and interpret the X and Y collected data (sample) by means of the Shapiro-Wilk normality test and the T-test; and
- analyze and interpret the correlation between the effectiveness of the approaches and the subjects characterization questionnaire by means of Shapiro-Wilk normality tests and the Spearman's ranking correlation technique.

1) Effectiveness of the Approaches:

- **Collected Data Normality Tests:** the Shapiro-Wilk [12] normality test was applied to the e-commerce sample (Table I) providing the following results:
  - for the X approach with sample (SampleX) size (N) 12, mean value (μ) 1.6667, standard deviation value (σ) 4.1096, it was obtained p = 0.0827, which means that with α = 0.05, the sample is normal;
  - for the Y approach with sample (SampleY) size (N) 12, mean value (μ) 4.500, standard deviation value (σ) 5.5453, it was obtained p = 0.1378, which means that with α = 0.05, the sample is normal.
- **T-Test for SampleX and SampleY:** this kind of test can be applied for both independent and paired samples. In the case of this study, SampleX and SampleY are independent. As each sample size is less than 30 and both samples are normal, it was defined the following hypothesis:
  - **Null Hypothesis** (H_0): approach X has the same effectiveness of approach Y.
    H_0 : μ(*effectiveness(X)*) - μ(*effectiveness(Y)*) = 0;
  - **Alternative Hypothesis** (H_1): approach Y is more effective than approach X.
    H_1 : μ(*effectiveness(Y)*) - μ(*effectiveness(X)*) > 0.

First we obtained the value of *T*, which allows the identification of the range entered in the statistical table t (*student*). This value is calculated using the average of SampleY (μ1 = 4.5000) and SampleX (μ2 = 1.6667), standard deviation value of both (σ1 = 5.5453 and σ2 = 4.1096), and the sample sizes (N = 12). It was obtained the value t = 3,9699.
By taking the sample size (N = 12), we obteined the degree of freedom (*df*), which combined to the t value indicates which value of p in the t table must be selected. The p value is used to accept or reject the T-test null hypothesis (H_0).
By searching the index *df* and defining the value t at the t table (*student*), we found a value which is greater than 0.001, with a significance level (α) of 0.05. The relation between α and p produces p = 0.001, which is less than α = 0.05. Therefore, the null hypothesis H_0 must be rejected and (H_1) must be accepted. It means that there is evidence that the Y approach (SMarty) is more effective in identifying and representing variability in use case models than the X approach (PLUS). This result also corroborates to reject the null hypothesis (H_0) of this experimental study (Section III-B) and accept the alternative hypothesis (H_1).

2) Correlation between the Approaches Effectiveness and the Subjects Variability Characterization:

- **Subjects Variability Characterization Normality Test:** Shapiro-Wilk was applied to the data extracted from the subjects characterization questionnaire of each approach:
  - for the X approach with sample size(N) 12, mean value of (μ) 2.2500, standard deviation value of (σ) 0.9242, the calculated variability knowledge level was p = 0.0002. This value with α = 0.05, indicates that the characterization data is normal;
  - for the Y approach with sample size(N) 12, mean value of (μ) 2.9167, standard deviation value of (σ) 1.1149, the calculated variability knowledge level was p = 0.4333. This value with α = 0.05, indicates that the characterization data is non-normal;
- **Spearman's Correlation:** this technique was applied to verify whether there is a correlation between the effectiveness of each approach (X and Y) and the level of knowledge of the subjects. Equation 1 shows the formula to calculate the Spearman ρ correlation, where n is the sample size:

$$\rho = \{1 - \frac{6}{n(n^2-1)} \sum_{i=1}^{n} d_i^2 \quad (1)$$

TABLE I
E-COMMERCE PL COLLECTED DATA AND DESCRIPTIVE STATISTICS: X (PLUS) AND Y (SMARTY) APPROACHES.

The X Approach (PLUS)			
Subject #	Correct Identified Variabilities	Incorrect Identified Variabilities	Effectiveness Calculation
1	5	6	-1.0
2	8	3	5.00
3	9	2	7.00
4	6	5	1.00
5	8	3	5.00
6	10	1	9.00
7	5	6	-1.00
8	4	7	-3.00
9	4	7	-3.00
10	8	3	5.00
11	4	7	-3.00
12	5	6	-1.00
Mean	6.3333	4.6667	1.6667
Std. Dev.	2.0548	2.0548	4.1096
Median	5.5000	5.5000	0.0000

The Y Approach (SMarty)			
Subject #	Correct Identified Variabilities	Incorrect Identified Variabilities	Effectiveness Calculation
1	5	6	-1.00
2	10	1	11.00
3	5	6	-1.00
4	8	3	5.00
5	11	0	11.00
6	11	0	11.00
7	9	2	7.00
8	9	2	7.00
9	5	6	-1.00
10	9	2	7.00
11	2	9	-7.00
12	8	3	5.00
Mean	7.6667	3.3333	4.3333
Std. Dev.	2.6874	2.6874	5.3748
Median	8.5000	2.5000	6.000

Table II presents the data needed to calculated the Spearman correlation for X and Y effectiveness and the subjects level of variability knowledge.
Equations 2 and 3 present the calculation of the Spearman correlation for the X (Corr.1) and Y (Corr.2) approaches, respectively.

$$\left.\begin{array}{l}\rho(Corr.1) = 1 - \frac{6}{12(12^2-1)} * 112 = 1 - 0.39 = \\ 0.61\end{array}\right\} \quad (2)$$

$$\left.\begin{array}{l}\rho(Corr.2) = 1 - \frac{6}{12(12^2-1)} * 300 = 1 - 1.04 = \\ -0.04\end{array}\right\} \quad (3)$$

Corr.1 for the X approach shown that there was a positive strong correlation ($\rho = 0.61$). This means that the subjects knowledge level on variability is important to correctly apply the PLUS method stereotypes for identifying and representing variability in use case models. On the other hand, Corr.2 shown that there was a negative weak correlation ($\rho = -0.04$). This means that the subjects knowledge level on variability is not important to correctly apply the SMarty approach stereotypes for identifying and representing variability in use case models.
An important evidence of this analysis is that the PLUS method does not provide guidelines to identify and represent variabilities in use case models. Thus, there is a need for previous variability knowledge to properly apply the PLUS stereotypes. In addition, the SMarty approach provides a set of guidelines, which may improve the activity of identification and representation of variabilities in use case models even for those subjects with lower variability knowledge level.

E. Validity Evaluation

1) **Threats to Conclusion Validity:** the major concern is the sample size. Although obtaining well-qualified subjects is not an easy task in software engineering experiments, we tried to minimize this threat by selecting the subjects by convenience. However, it is clear that such a sample must be increased in prospective studies to allow generalizing the conclusions.
2) **Threats to Validity Construction:** effectiveness is calculated based on the ability ot the subjects in modeling variability by taking into consideration the X and Y approaches and the e-commerce PL. The independent variable `variability modeling approach` is guaranteed by the pilot project undertaken.
3) **Threats to Internal Validity:** we dealt with the following issues:
   - **Diferences among subjects:** as we took into consideration a small sample, variations in the subject skills were reduced by performing the tasks in the same order. The subjects experience had approximately the same level for UML modeling and variability concepts;
   - **Fatigue effeects:** on average, the experiment lasted for 100 minutes, thus fatigue was considered not relevant; and
   - **Influence among subjects** it could not be really controlled. Subjects took the experiment under supervision of a human observer. We believe that this issue did not affect the internal validity.
4) **Threats to External Validity:** two threats were detected:
   - **Instrumentation:** failing to use real use case models, as the e-commerce PL is not commercial. More experimental studies must be conducted using real PLs, developed by industry; and
   - **Subjects:** lecturers and graduate students of Software Engineering were selected. However, more experiments taking into account industry practitioners must be conducted, allowing to generalizing the study results.

TABLE II
SPERMAN'S CORRELATION BETWEEN X AND Y EFFECTIVENESS AND THE SUBJECTS KNOWLEDGE LEVEL.

The X Approach (PLUS)						
Subject #	Effectiveness	r_{a1}	Knowledge Level	r_{b3}	r_{a1} - r_{b1}	$d1^2$
1	1,00	4	5	6	-2	4
2	9,00	6	3	1	5	25
3	7,00	3	2	2	1	1
4	5,00	2	2	3	-1	1
5	5,00	5	2	4	1	1
6	5,00	10	2	5	5	25
7	-1,00	7	2	8	-1	1
8	-1,00	12	2	9	3	9
9	-3,00	8	2	10	-2	4
10	-3,00	9	2	11	-2	4
11	-3,00	11	2	12	-1	1
12	-1,00	1	1	7	-6	36

The Y Approach (SMarty)						
Subject #	Effectiveness	r_{a2}	Knowledge Level	r_{b2}	r_{a2} - r_{b2}	$d2^2$
1	-7,00	10	5	12	-2	4
2	7,00	7	4	5	2	4
3	5,00	3	4	7	-4	16
4	5,00	11	4	8	3	9
5	11,00	5	3	1	4	16
6	7,00	6	3	4	2	4
7	-1,00	1	3	9	-8	64
8	11,00	12	2	2	10	100
9	9,00	4	2	3	1	1
10	-1,00	2	2	10	-8	64
11	-1,00	8	2	11	-3	9
12	7,00	9	1	6	3	9

IV. CONCLUSION

New theories and technologies must be experimented before they can be transferred to industry and effectively be adopted by software engineering practitioners. In this paper, it is shown how the effectiveness of a variability management approach (SMarty) can be analyzed to facilitate and improve variability activities in a PL perspective.

This experimental study is important to provide a means to demonstrate the ability to use the variability management approaches (PLUS and SMarty). An effectiveness calculation was done for each approach based on the application of the approaches theory for variability representation in use case models. This calculation allowed us to identify the more effective approach for use case models taking into account the e-commerce PL proposed by Gomaa [4].

Shapiro-Wilk normality test was applied to the data collected from the subjects, which demonstrated the normality of such a collected data. Therefore, the parametric T-test technique was applied providing evidence that SMarty is more effective than PLUS. As a last step of this study, it was performed a correlation between the variability knowledge level of the subjects and the effectiveness of each approach. Spearman ranking correlation technique provided evidence that, for PLUS, the previous knowledge on variability was important to guide the subjects on correctly identify variabilities in use case models. On the other hand, for SMarty, such a previous knowledge was not important for one to identify and represent variabilities in use case models. One of the main reasons might be the fact that SMarty provides a set of guidelines to identify and represent variabilities, making such an activity easier.

New experimental studies and replications must be planned and conducted to make it possible to reduce the threats, increasing the effectiveness of SMarty and generalizing the results. As new experiments, we are: (i) planning an effectiveness analysis of SMarty for class models taking into consideration PLUS; (ii) planning a replication of this study to corroborating the obtained results; and (iii) planning an experiment for effectiveness analysis of SMarty for sequence models.

ACKNOWLEDGEMENTS

The authors would like to thank the National Science and Technology Institute for Critical Embedded Systems (INCT-SEC, Brazil) for funding this work by means of the following agencies: CAPES, CNPq (grant # 573963/2008-8) and FAPESP (grant # 2008/57870-9).

REFERENCES

[1] K. Pohl, G. Böckle, and F. J. v. d. Linden, *Software Product Line Engineering: Foundations, Principles, and Techniques*. Secaucus, NJ, USA: Springer-Verlag, 2005.

[2] L. Chen, M. Ali Babar, and N. Ali, "Variability Management in Software Product Lines: a Systematic Review," in *Proc. Int. Software Product Line Conference*. Pittsburgh, PA, USA: Carnegie Mellon University, 2009, pp. 81–90.

[3] E. A. Oliveira Junior, I. M. S. Gimenes, and J. C. Maldonado, "Systematic Management of Variability in UML-based Software Product Lines," *J. Universal Computer Science*, vol. 16, no. 17, pp. 2374–2393, 2010.

[4] H. Gomaa, *Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures*. Redwood City, CA, USA: Addison Wesley, 2004.

[5] I. Jacobson, M. Griss, and P. Jonsson, *Software Reuse: Architecture, Process and Organization for Business Success*, 1997.

[6] A. Braganca and R. J. Machado, "Extending UML 2.0 Metamodel for Complementary Usages of the ≪extend≫ Relationship within Use Case Variability Specification," in *Proc. Int. Software Product Line Conference*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 123–130.

[7] B. Korherr and B. List, "A UML 2 Profile for Variability Models and their Dependency to Business Processes," in *Proc. Int. Conference on Database and Expert Systems Applications*. Washington, DC, USA: IEEE, 2007, pp. 829–834.

[8] T. Ziadi, L. Helouet, and J. marc Jezequel, "Towards a UML Profile for Software Product Lines," in *Product Family Engineering Conference*. Springer, 2003, pp. 129–139.

[9] D. R. Fiori, I. M. S. Gimenes, J. C. Maldonado, and E. A. Oliveira Junior, "Variability Management in Software Product Line Activity Diagrams," in *Proc. Int. Conf. on Distributed Multimedia Systems*, 2012, pp. 89–94.

[10] OMG. (2011) Unified Modeling Language, Superstructure Version 2.4.1. [Online]. Available: http://www.omg.org/spec/UML/2.4.1/Superstructure

[11] 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.

[12] S. S. Shapiro and M. B. Wilk, "An Analysis of Variance Test for Normality (Complete Samples)," *Biometrika*, vol. 52, no. 3/4, pp. 591–611, 1965.

Selecting Agile Practices for Developing Software Product Lines

Diego Spillere de Souza
Departament of Informatics and Statistics (INE)
Federal University of Santa Catarina
Florianópolis, Brazil
diego.ss@terra.com.br

Patrícia Vilain
Departament of Informatics and Statistics (INE)
Federal University of Santa Catarina
Florianópolis, Brazil
vilain@inf.ufsc.br

***Abstract*—Although traditional development practices are often utilized in the development of software product lines, new approaches have been proposed recently as an alternative to turn it into a more agile process. In this paper, we discuss how agile practices can be included in the development of software product lines in order to achieve that goal. The Framework of Agile Practices is utilized as a basis for our work. Each of its practices is included in the process as either optional or mandatory. We present an example that demonstrates the application of our proposal.**

Keywords-Agile Development; Agile Practices; Software Product Lines

I. INTRODUCTION

Agile methods have attracted great interest from developers for being fast, efficient and meeting the needs of customers. As we work directly with customers in initial development phases, it is common that they have difficulties in defining their real needs and, therefore, software requirements. That usually affects the development process when using a traditional method [1].

Agile methods had their origin in the 90's and appeared in opposition to traditional software development methods that were being practiced at the time. The software community started to question the effectiveness of traditional methods and their difficulty of being utilized. In 2001, a group of 17 software developers decided to spread their ideas and created the agile manifesto with values and principles that must be followed by agile methods [2].

Some important differences between agile and traditional methods of software development can be mentioned: agile methods are adaptive rather than predictive, and agile methods are people-oriented rather than process-oriented [3, 4]. However, software development methods, agile or traditional, are mainly used in the development of a single product. Although the developing of a product always considers the reusability of components and frameworks, the reusability of an architecture that is common for all products in the same domain is not yet widely adopted. This is where the development of software product lines plays an important role.

According to [5], a software product line (SPL) consists of a set of software systems that share common features and meet the needs of a particular market segment. Products of a SPL are developed from a common set of core artifacts in a prescribed manner. Although the definition and use of SPL began before the year 2000, the idea of incorporating agile practices in the development of SPL is somewhat recent [6, 7, 8].

This work proposes a set of agile practices to be utilized in the development of SPLs. It was inspired by a general framework for selecting agile practices [9]. We show how that framework can be extended to support the development of SPLs.

The paper is organized as follows. Section 2 revisits the Framework of Agile Practices as originally defined. Section 3 discusses the development of software product lines. In section 4 we propose an extension to the Framework of Agile Practices considering aspects relevant to the development of software product lines. Our proposal is demonstrated with an example in section 5. Section 6 presents our conclusions.

II. FRAMEWORK OF AGILE PRACTICES

According to the agile manifesto, every agile development must consider the following values: individuals and interactions among them worth more than processes and tools, working software worth more than comprehensive documentation, collaboration with the client worth more than negotiating contracts, and responses to changes worth more than to follow a plan [2]. Besides the values, the agile manifesto also presents 12 principles that must be followed by methods to be considered agile.

Several agile methods have been created since the 90s, and although they share common features and follow the same values and principles, they often have different agile practices [9].

The work proposed in [10] and extended in [9] consists of a comparative analysis of different agile methods and definition of a framework that combines agile practices from such methods. The agile methods included in this analysis are Extreme Programming (XP), Scrum, Feature Driven Development (FDD), Adaptive Software Development (ASD), Dynamic System Development Method (DSDM), Crystal Clear, Lean Software Development (LSD), and Agile Modeling (AM). This framework, also called Framework of Agile Practices (FAP), highlights the common features and the best

practices from different methods, and allows the definition of a specific agile process through the selection of a subset of agile practices.

Fig. 1 presents an overview of the framework, showing its activities flow.

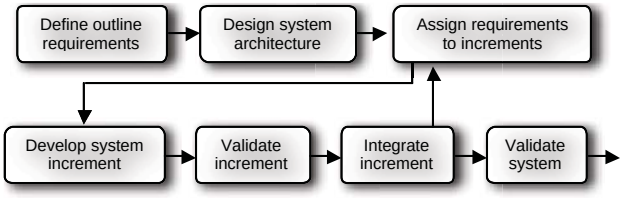


Figure 1. Activities of the Framework of Agile Practices

The agile practices pertaining to the activities of the current version of FAP are grouped as follows:

- Define outline requirements: list of requirements (mandatory), overall model, initial documentation.
- Design system architecture: system architecture design.
- Assign requirements to increments: iteration planning (mandatory).
- Develop system increment: daily meetings, user stories, use cases, iteration design, unit tests writing, acceptance tests writing, development (mandatory), pair programming, side-by-side programming, refactoring, collective development, continuous integration.
- Validate increment: code inspection.
- Integrate increment: iteration review meeting.
- Validate system: brief documentation, system delivery.

III. SOFTWARE PRODUCT LINES

Software Product Line (SPL) is a software engineering technique for creating a portfolio of similar software systems from a base of shared components and a similar production way [11]. The idea is to use common artifacts that can be applied to different products from a specific domain.

SPL is a paradigm that grows constantly in companies looking for reducing development time and costs, and increasing productivity and quality of their products. It allows entering into the market in a faster way [5].

According to [12], a key characteristic in the development of a SPL is the development of a component core and the development of products from this core. Both activities may occur in different orders or even in parallel. Moreover, the entire process should be managed. Fig. 2 characterizes these key activities.

It is possible to observe that these activities are iterative and are interrelated. It is also important to emphasize that there is no order of execution in these activities. From existing products, information about generic requirements and architecture can be extracted to the core of a SPL.

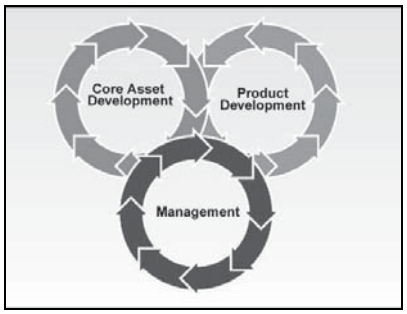


Figure 2. SPL key activities [12]

Next, the activities Core Asset Development and Product Development, also known as Domain Engineering and Application Engineering, are presented. They are the basis for extending the FAP. The Management activity will not be mentioned in this work.

A. *Domain Engineering*

Domain engineering is the activity responsible for establishing the capacity that is necessary to develop different products [13]. It requires building artifacts that can be reused.

According to [14], domain engineering is composed by three phases:

1) Domain Analysis: Definition of the reusable requirements for the systems belonging to the domain.

2) Domain Design: Development of a common architecture for the systems and a product plan.

3) Domain Implementation: Implementation of the core to be reused.

B. *Application Engineering*

Application engineering is the process of developing a specific application reusing the knowledge obtained from domain engineering [15].

There are several techniques for identification and representation of common features and variabilities among applications in the same domain [16]. In this work, we utilize the Feature-Oriented Reuse Method (FORM) [15].

The FORM proposes two phases for application engineering, as follows.

1) Requirements Analysis and Feature Selection: the application engineering starts by selecting the features defined in domain engineering that will be part of the product. The selection of features is done from the domain model, which presents the features that can be selected and their relationships.

2) Selection of the architecture and development of the application: a reference architecture model is generated automatically from the selection of resources done previously. From the architecture, reusable components can be easily found. Then, specific modules of each product must be implemented to complement the reused architecture.

IV. EXTENDED FRAMEWORK

This work aims at applying agile practices to the development of SPLs. However, as we analyze some definitions and principles of these paradigms, we can identify conflicting ideas between agile and SPL development processes. Having agility in the development process demands the reduction of diagrams and documentation, emphasizing immediate requirements. In contrast, to create a SPL with high quality and efficiency, the practices include the definition of diagrams and documentation right from the beginning.

Our extension to the Framework of Agile Practices (FAP) was defined as follows. We started with two development cycles[1]: the Domain Engineering cycle and the Application Engineering cycle, as shown in Fig. 3. Each cycle is similar to the FAP process and it is also divided into activities[2]. First, we included agile practices from the FAP. Then, from each cycle, we excluded agile practices that were not relevant to that cycle. We tried to keep all agile practices in the FAP that are related to iteration planning and programming and that can be applied to SPL development, such as daily meetings and writing tests. Finally, in each cycle, we included traditional practices we found essential to the development of SPLs, such as practices related to domain modeling, assuming there is no sense in modeling a SPL without modeling the domain. We also made mandatory some practices related to documentation that were optional in the original FAP because they are also essential for SPL development.

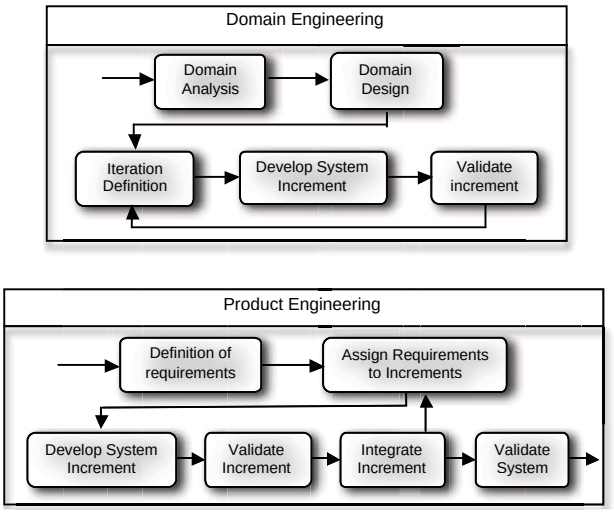


Figure 3. Cycles and activities of the Extended FAP

In domain engineering, initially an analysis of the domain is done, a feature model is defined and the architecture of the SPL core is designed. Next, the development of each component is made in an iterative manner.

In application engineering, the process is quite similar: for each product, an analysis of its requirements is initially done, and then followed by a specialization of the feature model and the architecture. The development is also iterative. Later the product is validated and then delivered to the customer.

It is important to point out that both domain engineering and application engineering can be carried out simultaneously.

A. Domain Engineering

As mentioned above, domain engineering is the cycle that gathers information and features belonging to a specific domain and describes them as reusable units. Domain engineering is divided in five activities, which are described next.

1) Domain Analysis: Domain analysis is the activity where an overview of the products belonging to the SPL and their characteristics is done. It consists of the following practices:

a) List of requirements of applications (mandatory): A list of requirements is built for each application belonging to the SPL. In order to do so, we analyze applications that have already been developed, are being developed or will be developed later.

b) Feature model (mandatory): It is the model that describes the characteristics of the SPL. This is usually a model that highlights the features identified in the products and describes those features by hierarchies of composition. Such a model identifies features that are common for all products and features that may vary among different products.

c) Documentation: Documentation should contain a brief description of the features. Moreover, this documentation may include an overview of the SPL with its cost estimation, risks, and expectations for each product.

2) Domain Design: This is the activity where a reusable architecture is specified according to the following practices:

a) Component design (mandatory): A component is defined for each feature. If necessary, a feature may be broken into components or a group of features may be combined as a single component. The design should be reviewed to verify that it meets the initial specification of the features in the feature model. Interfaces and classes that will be part of the core components must be specified for each module.

b) Domain architecture design (mandatory): The domain architecture has to be represented according to the design of its components. Different representations (e.g. package diagrams) may be utilized to describe that architecture.

3) Iteration Definition: Each iteration must be defined based on the following practice:

a) Iteration planning (mandatory): Components and their requirements are assigned to the iteration. The time required for development of the iteration is estimated. According to the FAP, this time should not exceed eight weeks. The iteration is broken in tasks, which are distributed to developers.

4) Develop System Increment: The SPL can always be revisited and new features added to it. The development of each increment should follow these practices:

a) Daily meeting: These are short meetings that occur every day among team members who report their progress or difficulties.

[1] Here, the term "cycle" corresponds to the term "activity" used in [12].
[2] The term "activity" utilized here corresponds to "phase" adopted in [14].

b) Iteration design: It consists of a design to describe the new functionality that will be implemented in the iteration. It is possible to enhance the existing documentation or develop a new simple diagram.

c) Unit test writing: It is recommended to write unit tests before coding components, based on pre-defined interfaces.

d) Acceptance test writing: When appropriated, acceptance tests are written for features that represent testable functionalities.

e) Iteration development (mandatory): It is the codification of the components belonging to the iteration.

f) Collective code development: Everyone who is part of the team becomes responsible for the code. Therefore, it is necessary that everyone knows and understands the code.

g) Pair programming: Coding is written by a pair of programmers. While one of them implements the code, the other one analyzes and suggests simplifications about what is being done.

h) Refactoring: Refactoring may happen anytime the team thinks it is important to improve the code.

i) Side-by-side programming: The work environment may be divided in such way that allows programmers to be side by side in order to improve the relationship among the team members.

j) Continuous integration: Components must be integrated into the code as soon as the corresponding components are implemented.

5) Validate Increment: Whenever a considerable change is done, there is need to validate the code. It may include the following practice:

a) Code inspection: Developers look for defects that may have been introduced in the generated code.

B. Application Engineering

Application Engineering refers to the development of a particular product utilizing the core artifacts obtained from domain engineering. Application engineering is divided in six activities, as follows.

1) Definition of requirements: Requirements are defined according to these practices:

a) List of requirements (mandatory): It is the definition of requirements for the product. The list of requirements may describe and assign a priority to each requirement.

b) Selection of components (mandatory): Considering the list of product requirements, components are selected from the core artifacts to be part of the product. The feature model resulting from domain engineering is utilized in this practice.

c) Overall model: It is defined as an instantiation of the feature model, where the selected features are included. When a new requirement is found not to be present in the feature model, it can be included in the overall application model.

d) Initial documentation: It is a document that provides an overview of the product considering its costs, benefits and risks. It can also contain information such as customer contact and meaningful technologies or tools.

2) Assign Requirements to Increments: Each iteration is defined as in the following practice:

a) Iteration planning (mandatory): It is necessary to define what will be developed in each iteration, taking into account core artifacts to be reused, requirements with higher priorities, risks and dependencies. As in the domain engineering cycle, iterations have a time length set between one and eight weeks. Requirements are distributed among developers who are part of the team.

3) Develop System Increment: The development of each iteration is carried out according to the following practices:

a) Daily meeting: These are short meetings where team members discuss their current progress and difficulties.

b) User stories: Users do a brief description of how to use the product functionalities.

c) Use cases: Each product requirement may have a corresponding use case.

d) Iteration design: The design is usually done using UML diagrams according to the requirements assigned to the iteration.

e) Unit test writing: It is recommended to write unit tests before coding the requirements.

f) Acceptance test writing: Acceptance tests are also written before coding.

g) Iteration development (mandatory): It consists of coding the product requirements assigned to the iteration, reusing the core artifacts developed in domain engineering.

h) Collective code development: The development of the code is responsibility of all members in the team.

i) Pair programming: Programmers are grouped in pairs. In each pair, one writes the code while the other one looks at how it is being done and suggests simplifications.

j) Refactoring: Code refactoring can be performed at all times during development.

k) Side-by-side programming: Programmers are arranged side by side in their work environment.

l) Continuous integration: Whenever a task is complete, its code is integrated into the next system build.

4) Validate Increment: Whenever after the increment development, the code is validated.

a) Code inspection: The code developed in the iteration is reviewed in order to identify potential problems. Developers inspect the code of each other.

5) Integrate Increment

a) Iteration Review Meeting: This is a meeting that occurs at the end of the iteration to discuss whether the code that was implemented satisfies the requirements set for that iteration.

6) Validate System

a) Generation of a Brief Documentation: If the product documentation has not been generated during its development, a brief document with a description of the product may be written and delivered to the customer.

b) Delivery of the System to Customer: The product is delivered to the customer as soon as all product requirements are properly implemented. A meeting to recognize the end of the project is recommended.

V. EXAMPLE

We now present an example that demonstrates the FAP extension proposed in this work. Our SPL included, initially, three products: Document Management System (DMS), an online tool that allows managing documents through a web browser; File Manager (FM), a web application for organizing files in a directory structure; Datebook System (DS), an online tool that offers an agenda for user appointments. As none of these products was already implemented, the domain engineering cycle was carried out prior to application engineering.

The following technologies were utilized in the development of our example SPL: Java 1.6, JSF 2.0, Tomcat 6.0, Richfaces 4.0, and the Eclipse Indigo tool. Next, we briefly talk about its development.

A. Domain Engineering

Table I presents the activities proposed by the extended FAP, highlighting the practices that we applied to this example.

TABLE I. DOMAIN ENGINEERING CYCLE

	Extended FAP	
Domain Engineering	**Activity: Domain Analysis**	
	List of requirements of applications (mandatory)	√
	Feature model (mandatory)	√
	Activity: Domain Design	
	Component design (mandatory)	√
	Domain architecture design (mandatory)	√
	Activity: Iteration Definition	
	Iteration planning (mandatory)	√
	Activity: Develop System Increment	
	Daily meeting	√
	Iteration development (mandatory)	√

After selecting practices in order to define an agile process, we began the SPL development. Initially, we defined the feature model containing the SPL features (Fig. 4). The components to support the features were implemented and included as core artifacts. Some of these features were declared as common and others as variable. After that, the SPL architecture was defined (Fig. 5).

The development of core artifacts in domain engineering was done during three iterations. Iterations lasted a week and included the following practices: iteration planning, distribution of modules to responsible, daily meeting, and the actual development.

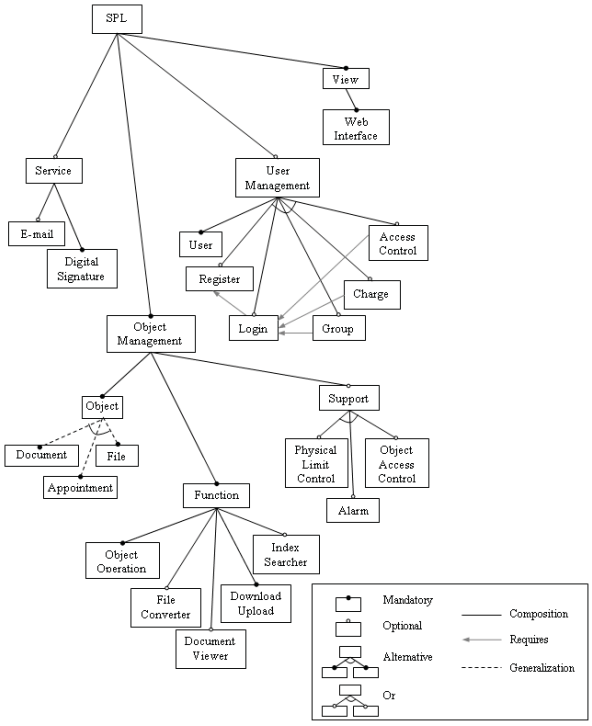


Figure 4. SPL Feature model

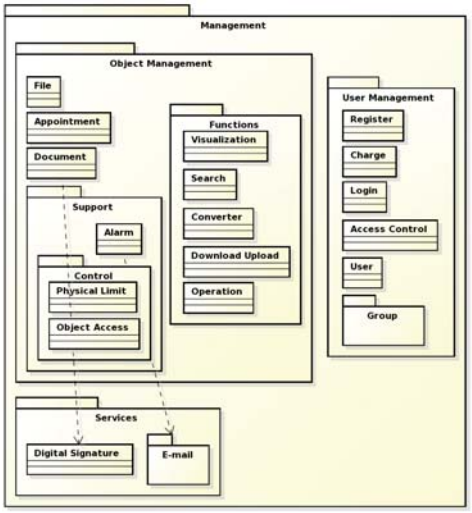


Figure 5. SPL Architecture

B. Application Engineering

Table 2 presents only the practices of the FAP application engineering cycle used in this example.

We developed three simple applications: DMS, FM, and DS. Next, we talk a little about the DMS development. DMS was the first application selected to be developed. It performs the management, control, storage, sharing and viewing of document information. Moreover, DMS stores documents in an internal format that allows the searching indexed by name and content.

TABLE II. APPLICATION ENGINEERING CYCLE

Application Engineering	**Extended FAP**	
	Activity: Definition of requirements	
	List of requirements (mandatory)	√
	Selection of components (mandatory)	√
	Overall model	√
	Activity: Assign Requirements to Increments	
	Iteration planning (mandatory)	√
	Activity: Develop System Increment	
	Iteration development (mandatory)	√
	Activity: Integrate Increment	
	Iteration Review Meeting	√
	Activity: Validate System	
	Delivery of the System to Customer	√

Considering the requirements list of DMS, some core components were selected to be used in its development.

DMS was implemented through two iterations. In the first one, we developed the view layer with its web pages and ManagedBeans. In the second iteration, we developed the layer responsible for the interface between ManagedBeans and core components. Fig. 6 shows the screen to view documents.

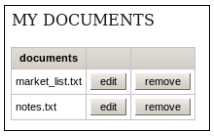


Figure 6. DMS View Screen

VI. CONCLUSIONS

In this work we analyzed the commonalities and conflicts between agile methods practices and SPLs practices. From this analysis, the Framework of Agile Practices (FAP) was extended to support the development of SPLs. This extension proposes the use of specific practices in the domain engineering and application engineering cycles.

We also developed an example SPL using this proposal. Spite of being simple, the SPL development was important to identify that the project was agile, except in relation to the documentation (feature model and architecture design). But as some documentation practices are mandatory for SPL development, we conclude that even the most agile SPL process will have to include this minimal documentation. As a continuation of our research work, we intend to develop other SPLs using the proposed extension by applying practices we did not use in this example.

Other authors also combine SPL and agile method approaches, but in different ways. [17] proposes an agile process for the SPL scope, called RiPLE-SC. Different from our proposal, RiPLE-SC is divided into four activities: pre-scope, domain scope, product scope, and components scope. The practices of pre-scope and components scope are similar to the practices included in the domain engineering of our proposal.

In [7], the authors propose an iterative model directed to organizations that already use agile methods but intend to build their SPL. Thus, a SPL is built from existing products. This model consider four topics: requirements analysis, reuse planing, artifacts core management, and flexible architecture. Comparing to our proposal, the practices of the first, second and fourth topics are similar to those included in our domain engineering cycle. The third topic has no equivalent in our proposal.

REFERENCES

[1] J. Highsmith, Agile software development ecosystems. Boston: Addison-Wesley, 2002.

[2] Manifesto Ágil, http://manifestoagil.com.br/, December 2011.

[3] M. Fowler, The new methodology, http://www.martinfowler.com/articles/newMethodology.html, November 2011.

[4] G. Chin, Agile project management: how to succeed in the face of changing project requirements, 2004.

[5] Software Engineering Institute, Software Product Lines – Overview, http://www.sei.cmu.edu/productlines/, December 2011.

[6] G.K. Hanssen, T.E. Faegri, "Process fusion: An industrial case study on agile software product line engineering", The Journal of Systems and Software,

[7] Y. Ghanam, F. Maurer, "Extreme Product Line Engineering - Refactoring for Variability: A Test-Driven Approach", 2010/

[8] K. Mohan, B. Ramesh, and V. Sugumaran, "Integrating Software Product Line Engineering and Agile Development", IEEE Software, May/June 2010.

[9] P. Vilain, P.B. Fagundes, T.L. Machado, "A Framework for Selecting Agile Practices and Defining Agile Software Processes", SEKE 2007, pp. 25-28, 2007.

[10] P. B. Fagundes. Framework for Comparing and Analyzing Agile Methods, Master Thesis, UFSC, Brazil, 2005 (in portuguese).

[11] C.W. Krueger, Introduction to Software Product Lines, http://www.softwareproductlines.com/, December 2011.

[12] L. Northrop, P. Clements, A Framework for Software Product Line Practice, Version 5.0, http://www.sei.cmu.edu/productlines/frame_report/index.html, December 2011.

[13] P. Clements, L. Northrop, Software Product Lines: Practices and Patterns. Addison-Wesley, 2002.

[14] E.S. Almeida, RiDE: The RiSE Process for Domain Engineering, PhD Thesis, UFPE, Brazil, 2007.

[15] K.C. Kang, J. Lee, and P. Donohoe, "Feature-oriented product line engineering", IEEE Software, vol. 19, pp. 58–65, July/August 2002.

[16] L. Chen, M.A. Babar, and N. Ali, "Variability management in software product lines: a systematic review", SPLC 2009, pp. 81-90, 2009.

[17] M. Balbino, E.S. Almeida, S. Meira, "An Agile Scoping Process for Software Product Lines", SEKE 2011, 2011.

Domain Analysis in Combination with Extreme Programming to Address Requirements Volatility Problems

Andrea Janes[1], Sarunas Marciuska[1], Alessandro Sarcia[2], Giancarlo Succi[1]
[1]Free University of Bozen-Bolzano, Piazza Domenicani 3, Bozen-Bolzano, Italy
[2]University of Rome "TorVergata", Via del Politecnico 1, Rome, Italy
andrea.janes@unibz.it, marciuska@inf.unibz.it,
sarcia@disp.uniroma2.it, giancarlo.succi@unibz.it

Abstract

It is of interest for practitioners as well as academics to apply Agile software development practices in high ceremony environments.

This paper presents the challenges a software development team faced when developing software for the Italian Army in a contracting environment.

We used action research as the research methodology to first identify the challenges and then make improvements. The identified challenges are volatile requirements, low domain knowledge, and unclear responsibilities. To address the challenges, the team applied Extreme Programming in combination with domain analysis techniques. Moreover, the team customized a requirement management system to increase visibility to show the causes of delays.

The overall feedback from the team was that the proposed practices improved managing the requirements.

1 Introduction

In a contracting environment, user requirements are typically predefined and deadlines already in place. To obtain the acceptance of the bid, a company has to make an upfront estimate of the costs to build it and offer a better bid than the competitors. The project finishes successfully if the company meets its estimates.

This paper reports about a team that was involved in the development of a customized governance software system for the Italian Army in a contracting environment, but adopting single practices recommended by Extreme Programming [1].

One of such a practice is "Reflection", i.e., to reflect regularly about past successes and failures to institutionalize the lessons learned from these experiences. During such a reflection meeting, 1 month after the start of the project, the development team identified the following major challenges during the development: 1) requirements volatility, 2) low domain knowledge, and 3) the delay in the predefined schedules.

According to the programmers, those problems arose from the fact that the estimate of the plan was based on a poor upfront requirements analysis. The description of some of the requirements was abstract and incomplete. Therefore, the team expected to have a high volatility in the requirements. In addition, the requirements document was written using domain specific terms that were not familiar to the developers. Therefore, the fear was that the predefined schedule would not be met.

To address these problems, the team investigated how to address these issues. This paper reports about the experiences made in this effort: to deal with the abstract and evolving requirements, the team adopted Extreme Programming practices; to understand the domain specific terms and to obtain other missing information of the domain, it performed a domain analysis; to address the problem with upcoming deadlines, it customized a requirement management system that keeps track who—the development team or the customer—is responsible for the delay of the project.

This paper is structured as follows: section 2 presents related works, section 3 the project environment and the decisions that the team made to solve the identified problems in the contracting environment, section 4 a brief description of the adopted research approach, section 5 the summary of the results.

2 Related work

It is hard to use Agile methods within a traditional software development environment. Three attempts to

do so are summarized below.

Alleman et al. [2] present how Agile can be used in conjunction with earned value management in a contracting environment. The authors show that earned value analysis provides means to overcome one shortcoming of Agile: the inability to forecast the future costs and schedule of a project beyond the usual short releases approach. Their work focuses more on the plan estimation in a contracting environment than on dealing with requirements volatility.

Fruhling et al. [3] present a case study about how Extreme Programming was applied in a US Government system development project. The authors present the practices that they applied in the project and the lessons learned while applying those practices. They report that it was difficult to follow the Extreme Programming practices during the all software development phase. In addition, the tasks were not reprioritized and the development plan was not modified when the scope of the project changed substantially. The authors suggest that the problems can be solved through a better management.

Domain analysis methods [4] help to collect and store the knowledge of a domain expert and identify commonality and variability within the domain. In a high ceremony software development process, the domain analysis is performed upfront and it requires a large amount of resources. In this paper we report how an iterative domain analysis helped to enhance the flexibility provided by an Agile development process.

3 The faced problems and the adopted solutions

To provide the context of the project we studied, we briefly summarize the requirements given by the Italian Army. The initial meetings revealed that the development team would have to design and implement an application for the Italian Army to manage, define, monitor, and execute tasks and their respective budgets. The system had to provide an efficient, traceable, and repeatable method to evaluate the performance of tasks. In addition, the system had to respect the hierarchical structure of the Italian Army. Moreover, the system had to allow to create and visualize a predictive analysis during the strategic planning and financial management. The first meetings showed that the Italian Army representatives had a high-level idea of the final system and that the requirements would refine and eventually change and evolve during the development.

The project was carried out by a team of eleven developers that had Java development skills with an average of 5 years of experience, a strong algorithmic background, and experienced to work in an Agile environment. During the development, since the priorities for the company changed, three members were replaced by other developers and were assigned to another project.

We now describe three problems identified during a reflecton session, describe the undertaken solutions, and present the effect of the solutions on the develoment project.

3.1 The volatiliy of requirements

The requirements documents and meetings with the customer revealed that there was a clear target for the final product, but some of the required features (as well as their prioritization) were unclear. Due to the imprecision found in the requirements and due to the complexity of the envisioned system, the team decided to adopt a customized version of Extreme Programming [5]. The team excluded "the planning game" practice from the development process, because of already predefined deadlines (see [6] for a comparison of project management in plan-based and agile companies). The developers were aware that "the planning game" is one of the core practices in the Extreme Programming software development, but they assumed that by applying the remaining practices they would reduce the complexity of the project and would be able to follow the predefined deadlines [7].

The Extreme Programming practices applied by the team are summarized in table 1, together with the expected benefits. In the last column we report the feedback (i.e., the opinion of the team) whether the majority of the team members considered a practice successful, i.e., having a positive impact on the outcome of their project (marking it with a +), or not (marking it with a –).

To evaluate the effect of the adopted practices, the team discussed about the impact of those Extreme Programming practices that were applied on the project, of those that were not applied on the proejct, and about the reasons why those practices were not applied. The discussion showed that the team observed the positive impact of the Extreme Programming practices when they started to develop the software. Small releases, continuous integration, pair programing, and unit tests helped the development team to identify and fix bugs efficiently. Collective ownership of the code allowed developers easily to modify code of their team members.

Predefined coding standards helped to train and integrate new developers into the project. The team observed that simple design and refactoring helped developers easier extend the system with the new features, or to update existing ones. However, the ap-

Table 1. Extreme Programming practices in the contracting environment

Practice	Expectations	Impression
Whole team	The fast feedback of the customer help to reach the predefined deadlines.	–
Acceptance tests	Acceptance tests help to make sure that developers implement the features as required. This practice helps to avoid the time spend to re-implement unclear requirements.	–
Small releases	Small releases expose the problems in the system early.	+
Continuous integration	A continuously built and tested system reveals existing problems sooner than one that is built and tested before shipment.	+
Collective ownership	Unexpected problems can be addressed more efficiently if several team members are familiar with the code and we practice common code ownership.	+
Coding standards	Standards help to modify the system faster.	+
Sustainable pace	A sustainable pace helps the quality of the system design and code high.	–
Pair programing	Pair programing helps the development team to keep the continuous attention on technical excellence and good design. Also, it supports the knowledge transfer.	+
Test driven development	Tests help the development team to be confident in their development effort by constantly signaling if newly implemented code breaks previously developed functionalities.	+
Refactoring	Refactoring helps the development team to decrease the complexity of the system.	+
Simple design	The design of the system has to be simple and understandable, otherwise it will become difficult to implement new requirements later.	+

plication of other Extreme Programming practices was more difficult than expected. First of all, the customer was not part of the team. He attended meetings using Skype[1], but there were several cases when the development team had to wait a few days for clarification of the requirements. Another issue was related to the fact that the team could not manage to convince the customer to write the acceptance tests in a formal way. This, because the writing of an acceptance test is a time consuming process or the customer had only an abstract understanding what the requirements suppose to do.

An additional difficulty was to keep a sustainable pace. This problem was particularly acute before the predefined deadlines. There was a big pressure from the managers to deliver as many features as possible to get close to the plan as close as possible. The direct outcome of such situation was that the developers had to work overtimes and the simple design and the refactoring was postponed. In total the development became more and more costly, because the quick fixes introduced before the deadlines had to be fixed afterwards.

3.2 Delays

While the side effects of the absence of the sustainable pace practice was visible only for the short periods of time (particularly before deadlines), the missing acceptance tests and the delayed responses from the customer were adding uncertainty to the requirements during the whole development process. Having uncertainty in the user requirements made it difficult to fulfill a release plan, especially if there were a high number of unclear requirements that had to be delivered for an upcoming deadline.

To have a global overview of the release plan, to alleviate the absense of the customer, and to force everybody to state clear requirements, the team introduced a ticketing system. For that purpose, the team used the Trac ticketing system[2], because the development team was familiar with it.

The team discussed about the impact of the ticketing system used in the project. The team reported that some of the expected benefits could not be achieved. The number of unclear tickets was increasing. In addition, a lot of times fixed tickets coming from that group were reopened by the customer, because developers misunderstood the requirement. This was more evident just before the deadline when developers tried to deliver as many requirements as possible (including the unclear ones), but the deliverables were returned back by the customer. Such situation increased a tension between the customer and the development team: developers were blaming the customer that he did not know what he wanted; the customer was blaming the developers that they were not able to deliver what he asked.

[1]Skype, http://www.skype.com

[2]The Trac Ticket System, http://trac.edgewall.org

Table 2. Requirements life cycle

Phase	Description	Responsible: Developer	Responsible: Customer
Unclear	The development team is missing some information from the customer to proceed with the requirement.		×
Analyze	The requirement is clear, but developers are discussing how to implement it.	×	
Implement	The requirement is clear, developers agreed how to implement it, and it is being implemented.	×	
Test	The development team is testing if the requirement was implemented correctly.	×	
Verify	The client is verifying if the requirement was implemented correctly.		×

Finally, developers delivered what was required using quick fixes and refactored large parts of the code after the deadline. It was a short time solution, because the resources planned for the next deadline were used.

To avoid such problems in the future the team modified the ticketing system. The main idea of the modification was to point out the people responsible for the delays of the implementation of the requirements. The increased visibility helps to coordinate the activities of different stakeholders [8]. We introduced 5 phases that a requirement typically traverses: unclear, analyze, implement, test, and verify. The description of each phase and who is reponsible if in that phase a delay occurs, is described in table 2.

The team defined rules how the requirement can move from one phase to another. A requirement leaves the unclear requirement phase only if both the customer and the development team agree on that. The phases "analyze", "implement", and "test" were introduced to monitor the internal requirement life cycle. Only if a requirement passed the last phase it was considered as implemented as required.

The team discussed about the impact of the modified ticketing system used in the project. The outcome was that the modified ticketing system added transparency on the people responsible for the project delays. The team observed that this modification added additional pressure on the customer and the developers. As a result of this the customer was clarifying the uncertainty in the user requirements faster, and the developers were more focused and efficient.

3.3 Missing domain knowledge

In parallel to the problems related with volatile requirements, the team had a lack of domain knowledge. The detailed analysis of the requirements and the development indicated that the development team as not familiar with some domain specific terms used in the requirements document. In some cases, to understand a requirement, the domain context was needed. It also happened that when a requirement changed, also the scope of the project changed, which also increased the necessary domain knowledge.

To address the uncertainty brought by the lack of domain knowledge the team used three practices suggested domain analysis [4]: the creation and maintenance of a domain dictionary, of a feature diagram, and to perform an iterative domain analysis.

The domain dictionary describes the terminology and the necessary background to understand requirements for a particular domain. Feature diagrams show dependencies between features, and document variabilities. They are needed during domain analysis, a practice that aims to improve reuse studing systems to identify their common and variable parts [4].

The practices applied by the team are summarized in table 3, together with the expected benefits. In the last column we report the feedback (i.e., the opinion of the team) whether the majority of the team members considered a practice successful, i.e., having a positive impact on the outcome of their project (marking it with a +), or not (marking it with a –).

The team discussed about the impact of the applied domain analysis practies on the project. The result was that the application of domain analysis confirmed the assumptions of the team. Since the system that the developers were developing was complex and had a specific vocabulary, the domain dictionary helped to solve communication problems between the customer and the software development team. Feature diagrams created of competitor products helped developers to understand the context of the domain and to recommend additional features that the customer needed, but he was not thinking about. Finally, the domain analysis, performed in iterations, was useful when the requirements exceed the scope of the initial domain and developers needed to gather the missing information about the extended domain. Iterative domain analysis

Table 3. Domain analysis practices in the contracting environment

Practice	Expectations	Impression
Domain dictionary	Helps developers to understand the terms used in the requirements document.	+
Feature diagram	Is useful to capture the general picture of the expected requirements and help to model the missing domain context.	+
Iterative domain analysis	Provides the missing information when new requirements exceed the scope of the considered domain.	+

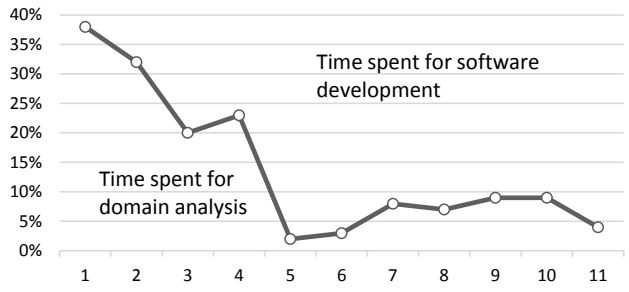


Figure 1. Time spent for the domain analysis

helped the team to understand the new requirements and suggest possible extensions of the product. Some of the suggestions led to solutions to choose COTS products. In such a way the overall resources needed for the development were reduced.

From an agile perspective, domain analysis is not recommended, since it requires effort and does not provide value to the customer. Nevertheless, in a context in which the domain was completely new, it proved to be of value to the customer since the team was able to implement the desired solution faster than without domain analysis.

During the first year of development, the team performed 11 domain analysis iterations. The iterations were performed once a month, because in this period a reasonable amount of missing domain information was accumulated. The biggest effort was invested at the beginning of the project, because all of the domain analysis deliverables had to be obtained from scratch. Figure. 1 shows the effort dedicated to domain analysis. In the middle of the year the team had a deadline, so few resources where available for domain analysis.

The total effort spent for the domain analysis practices was ∼3% of the overall time spent for the project. The activities took ∼150 out of ∼5,000 working hours spent for the project in one year. The time spent on domain analysis activity was measured using automatic, non-invasive measurement [9, 10, 11].

The study of the effects of the domain analysis practices showed that domain analysis was helpful to integrate new team members into the project. The domain analysis deliverables helped the new team members to understand the context of the domain in a few days instead of a few weeks that were spent by the core developers at the beginning of the project when the domain analysis was not yet performed.

4 Discussion

The here described experience was gained using an action research approach, a research method in which the researcher acts to accomplish something and at the same time analyzes his and the actions of others to learn from it. It is "action disciplined by enquiry, a personal attempt at understanding while engaged in a process of improvement and reform [12]." In fact, two of the authors of this paper were part of the development team described here.

The findings we report in this paper are the findings and considerations that the development team made during the project. The main validity threat of our work is external validity, i.e., if our findings can be generalized beyond the reported study. In this context, "recoverability" plays an important role: one has to be able to recover the research content and "appraise the judgments being made by the researcher in the course of the work [13]."

Due to space constraints, to improve recoverability, for each of the introduced practices or modifications we reported the considerations of the team before and afterwards. In a detailed report, the reasons of different team members to consider a practice useful or not should also be described, the particularities of the contracting context, the exact way how the adopted techniques where used, etc.

5 Conclusion

This paper reports how a team combined Extreme Programming development practices, a clear assignment of responsibilities during the requirement life cy-

cle, and domain analysis to overcome the problem of unclear, volatile requirements within a contracting environment.

Both, the team and the Italian Army representatives reported that the requirements management system brought transparency to the project development process. It helped to point out for each requirement who is currently responsible for the progress of the project. The team reported that the additional transparency results in a more efficient work.

Whenever the project scope changed, the team had to cope with new domain knowledge. Sometimes the change required a large code refactoring, or brought a new technology into the system. To alleviate this problem, the team performed the iterative domain analysis. The team reported that the iterative domain analysis helped to stay up to date with the domain knowledge.

Initially, the team thought that domain analysis consumes a lot of resources and should not be applied in an Agile development process when the requirements are changing. However, the team learned that the biggest effort was put in the initial iteration and that the successive iterations—when the domain knowledge did not change considerably—required a relatively low effort.

One problem that could not be solved was over-optimistic planning: The team reported that before deadlines managers were changing the development plan to fulfill the predefined goals. This resulted in sloppy development practices aimed to achieve short term goals. The resulting system became complex and required a lot of resources to refactor it after the deadlines. It became clear to the development team that the adopted practices do not help to avoid over-optimistic planning and that accumulated technical debt caused further delays due to the necessary refactoring sessions after the deadline and delaying the next iteration.

References

[1] K. Beck and C. Andres, *Extreme Programming Explained: Embrace Change*, 2nd ed. Addison-Wesley Professional, 2004.

[2] G. Alleman, M. Henderson, and R. Seggelke, "Making agile development work in a government contracting environment-measuring velocity with earned value," in *Agile Development Conference, 2003. ADC 2003. Proceedings of the*, 2003, pp. 114–119.

[3] A. Fruhling, P. McDonald, and C. Dunbar, "A case study: Introducing extreme programming in a us government system development project," in *Hawaii International Conference on System Sciences, Proceedings of the 41st Annual*, 2008, pp. 464–464.

[4] P. Predonzani, G. Succi, and T. Vernazza, *Strategic Software Production with Domain-Oriented Reuse*, 1st ed. Norwood, MA, USA: Artech House, Inc., 2000.

[5] A. Sillitti and G. Succi, "Requirements engineering for agile methods," in *Engineering and Managing Software Requirements*, A. Aurum and C. Wohlin, Eds. Springer Berlin Heidelberg, 2005, pp. 309–326.

[6] M. Ceschi, A. Sillitti, G. Succi, and S. De Panfilis, "Project management in plan-based and agile companies," *Software, IEEE*, vol. 22, no. 3, pp. 21 – 27, May-June 2005.

[7] A. Sillitti, M. Ceschi, B. Russo, and G. Succi, "Managing uncertainty in requirements: A survey in documentation-driven and agile companies," in *Proceedings of the 11th IEEE International Software Metrics Symposium*, ser. METRICS '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 17–.

[8] A. Janes, A. Sillitti, and G. Succi, "Effective dashboard design," *Cutter IT Journal*, January 2013.

[9] A. Sillitti, A. Janes, G. Succi, and T. Vernazza, "Monitoring the Development Process with Eclipse," in *ITCC (2)*. IEEE Computer Society, 2004, pp. 133–134.

[10] ——, "Measuring the architecture design process," in *Software Engineering Research and Practice*, H. R. Arabnia and H. Reza, Eds. CSREA Press, 2004, pp. 80–82.

[11] ——, "Measures for mobile users: an architecture," *Journal of Systems Architecture*, vol. 50, no. 7, pp. 393–405, 2004.

[12] D. Hopkins and E. Ahtaridou, *A Teacher's Guide to Classroom Research*. McGraw-Hill, 2008.

[13] P. Checkland and S. Holwell, "Action research: its nature and validity," in *Information Systems Action Research: An Applied View of Emerging Concepts and Methods*, ser. Springer's integrated series in information systems, N. Kock, Ed. New York, New York, USA: Springer-Verlag New York Inc., November 2006, no. 13, pp. 9–21.

A Mutation Approach to Feature Testing of Software Product Lines

Johnny Maikeo Ferreira, Silvia Regina Vergilio
{jmferreira, silvia}@inf.ufpr.br
DInf-UFPR - Federal University of Parana
Brazil CP: 19081, CEP: 81531-970

Marcos Antonio Quinaia*
quinaia@unicentro.br
UNICENTRO- State University of Central West
Guarapuava, Brazil CEP: 85040-080

Abstract

Diverse development methodologies use the feature model (FM) to represent common and variable features of a software product line (SPL). This model has also been used to derive products for testing. However, the test of all combinations of features (products) is infeasible in practice, due to the growing complexity of the applications, and only a subset of products is usually selected. Existing selection methods do not consider faults in the FM. The application of a fault-based approach can increase the probability of finding faults and the confidence that the SPL products match the requirements. Considering that, this paper introduces a mutation based approach to help in the selection of products for feature testing of SPLs. Mutation operators are introduced and a testing process is also proposed. Results from a case study are reported, and a comparison with pair-wise testing shows that other kind of faults can be revealed by the introduced approach.

Keywords: SPL; feature model; testing criteria

1 Introduction

A Software Product Line (SPL) is defined as a set of software products that share common features. A product of the SPL is then given by a combination of its features. To represent such features and their relationships, the feature model (FM) has been widely used by different SPL development methodologies.

We can observe a growing SPL adoption in the industry, and as a consequence, an increasing demand and interest in SPL testing. This is corroborated by the great number of existing surveys on this subject [6, 8, 13, 14] mentioning publications regarding to general and specific aspects of SPL testing. All of them emphasize variability and feature testing as a fundamental key aspect. An important question to be addressed is how to ensure that the products generated from the SPL feature model match their requirements.

To answer this question, ideally, all products should be validated. But, unfortunately, the space of possible combinations in many cases is likely to be enormous and the exhaustive test of all combinations is not always possible [5]. The increasing size and complexity of SPL applications can make testing all functionality combinations almost infeasible in practice. Variability and testability are viewed as a trade-off [13]. This is similar to what happens in the test of programs; it is impracticable to test all the paths of a program, and to deal with this limitation, testing criteria are used to select the best ones. In the SPL context, the problem is to select a subset of products for testing. According to Oster et al [10], an adequacy SPL test criterion should cover as many interactions among different features as possible, increasing the probability of finding faults.

In the literature, we can find selection criteria based on combinatorial testing [5, 7, 9, 10, 11, 15]. Most of them are based on pairwise testing. The idea is to select products to cover all pairs of feature interactions. The work of Cabral et al [3] generates a set of independent paths (products) by applying the basis path testing for control flow graphs in a feature inclusion graph that represents the dependencies between features.

A limitation of these works is that they do not con-

*The authors would like to thank CNPq and Araucaria Foundation for financial support.

sider common faults that can be present in the FM to select the products. The proposed criteria are not fault-based. Such kind of criterion, like mutation analysis, was experimentally considered the most efficacious to reveal faults in the context of program testing [18]. This is a motivation to investigate the use of such criterion in the feature testing of SPL, and this is the goal of our work. To do this, this paper introduces a set of mutation operators derived from a FM meta model introduced in [2]. The operators describe mistakes associated to the features management and that can introduce faults in the FM. A mutation based testing process that uses such operators is also described. Results of using this process in a case study are also presented. They are compared with a pair-wise approach based on the AETG algorithm [7] and show that the mutation approach is capable to reveal faults not revealed by the combinatorial approach. It can be used in a complementary way to select and evaluate the feature combinations (products) for SPL testing.

The paper is organized as follows. Section 2 reviews FM concepts and adopted metamodel, as well as related work. Section 3 introduces the mutation approach, describing the mutation operators, mutation testing process, and examples. Section 4 describes the evaluation case study showing some preliminary application results. Finally, Section 5 concludes the paper and points out future research works.

2 Background and Related Work

The feature model is a compact representation of all the products of an SPL [12] and has been widely used by different SPL development methodologies. The model has been formalized to automatically perform SPL analysis. FAMA [1][1] is the framework used in our study. It adopts the metamodel of Figure 1, and an XML representation to the FM [2].

To illustrate the model, consider the FM for the car audio system (CAS) of Figure 2. CAS is the root feature that has a set of constraints (depends and excludes, respectively RX and RY in the figure). CAS is composed by an optional set of relations. Relations can be of two different types: i) binary relations which includes mandatory (e.g. R1), optional (e.g. R2) and cardinality-based; or ii) set relations (e.g alternative choice R10, and or-relation R11). A feature can be of

[1]http://wwwisa.us.es/fama. It allows automated FM analysis to determine whether a feature model is void (case in which it represents no products); whether it contains dead features (that are not part of any product); whether a product is valid for a FM (the product can be generated from the FM); the number of SPL valid products for a FM; and so on.

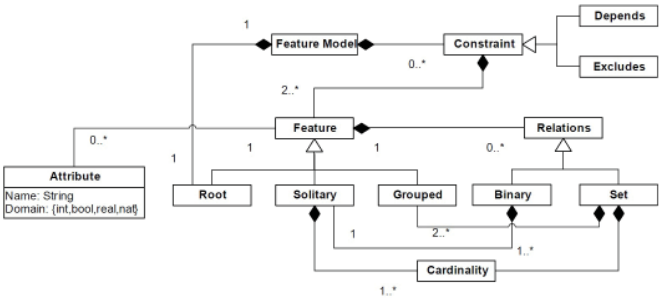


Figure 1. FM metamodel (extracted from [2])

three different types and is composed by one or more relations. A set relation is composed by at least two grouped features (e.g. CD, Cassette and DVD). A binary relation is composed by one and only one solitary feature. In addition to this, a solitary feature has a cardinality. Only cardinalities $n > 1$ are represented in the graph. The graph has no solitary feature that is cloned a number n of times. The cardinality of the solitary features Traffic Message Channel (not cloned mandatory feature) and Wheel Control (optional feature) are respectively [1..1] and [0..1], which are not usually represented in the graph [2].

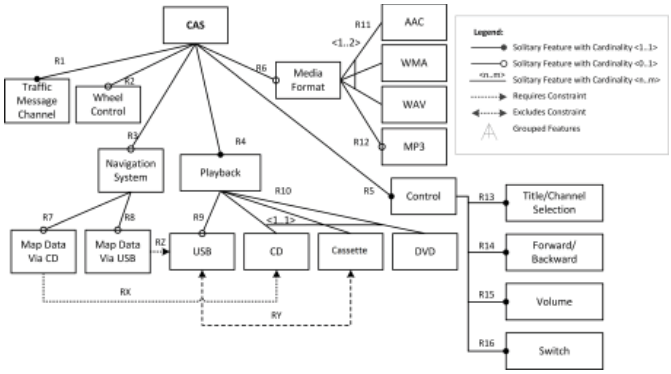


Figure 2. Feature diagram of the car audio system (adapted from [17])

Feature models have been used to instantiate products for testing [5, 11, 15]. The products are software systems built by composing the software assets that implement each feature, and are generally represented in terms of their variabilities. An example of test case for the FD of Figure 2 is (Navigation System, Map Data via CD, Map Data via USB, USB, CD, WMA, MP3). This test case is a valid product. But there may be invalid test cases, such as (Navigation System, Map Data via CD, Map Data via USB, USB, CD, Cassette, WMA, MP3). According to the FD, USB and Cassette can not be present at the same time in a product.

Ideally, all products should be validated. Unfortunately the space of possible combinations in many cases is likely to be enormous and exhaustive consideration of all combinations is infeasible [5]. The increasing size and complexity of applications can make testing of all functionality combinations almost impossible in practice. Variability and testability is a trade-off [13].

To solve this problem we can find in the literature many works, most of them based on combination testing. McGregor [9] considers combinatorial interactions methods (orthogonal arrays) to covering the space of SPL variabilities. The work of Cohen et al [5] is based on pairwise testing and covering arrays to select the products to be tested. It uses as basis the OVM (Orthogonal Variability Model). The work of Oster et al [10] combines combinatorial and model based testing for pairwise feature generation. Uzuncaova et al [15] transform the feature model into Alloy specifications and uses this to generate test cases. The use of Alloy formulae is also explored in [11]. This work investigates two divide-and-compose strategies do deal with scalability of t-wise testing.

The work of Lamancha and Usaola [7] proposes an extension to the AETG algorithm (algorithm introduced by Cohen et al [4] to perform t-wise testing) to consider depends and excludes relationships[2].

Differently, the approach of Cabral et al [3] is not based on combinatorial testing. The approach named FIG Basis Path, transforms the OVM in a feature inclusion graph that represents the dependencies between features. The goal is to generate a set of independent paths (or products) for the graph, analogous to the basis path testing in control flow graphs of programs.

All the mentioned works offer a test criterion to cover as many interactions among different features as possible, since the test of all combinations is not practicable. However, it is not possible to ensure that the subset of products selected is capable to reveal all the feature management faults. To increase the probability of finding faults and the confidence that the products represented by the FM match their requirements, we introduce in the next section a fault based approach based on mutation testing. The approach introduced can be viewed as complementary to the existing ones.

3 The Fault Based Approach

This section introduces the mutation operators based on classes of possible FM faults, some examples of mutants, and the mutation testing process applied to the products selection.

[2]This algorithm is available in a tool named Combinatorial Tool, http://161.67.140.42/CombTestWeb.

3.1 Mutation Operators

The operators are defined according to a formal model based on the metamodel of Figure 1. A feature model is denoted by $FM = (F, C, R)$ where:

• F is the set of features, which is composed by the subsets *Root* (root), S (solitary), and G (grouped). A feature $f \in F$ can be: (i) $r \in Root$; (ii) $s_{min,max} \in S$ where the cardinality associated to the solitaire feature is represented by min and max, lower and upper bounds respectively; and (iii) $g \in G$.

• C is a set of constraints composed by sets D (depends) and E (excludes), such as $d(f_j, f_k) \in D$ and $e(f_j, f_k) \in E$.

• R is a set of relations composed by the sets Bin (binary) and Set (set). A binary relation $b(s_{min,max}) \in Bin$ is composed by a single solitary feature. A set relation $st_{min,max}(g_1, ..., g_n) \in Set$ is composed by grouped features. Similarly to solitary features, this relation comprises one or more cardinalities, represented by min and max, for lower and upper bounds respectively, with $n > 1$, $min \leq max$, and $max \leq n$.

Mandatory and optional features are represented respectively as $b(s_{1,1})$ and $b(s_{0,1})$. A cardinality-based relation (cloned feature) is represented as $b(s_{n,m}) \mid m > 1,\ m \geq n$. The or-relation is represented by $st_{min,max}(g_1, ..., g_n) \mid 0 \leq min \leq max$, and $max \leq n$. An alternative choice is given by: $st_{1,1}(g_1, ..., g_n)$.

To propose the operators, possible faults in the FM were identified and grouped into the following classes:

1) *Incorrect Cardinality of a Solitary Feature*: a solitary feature is mistakenly defined. Once solitary feature cardinality accepts different value ranges, a possible fault occurs if a required mandatory feature receives cardinality values 0 and 1 for lower and upper bounds respectively. In this case the feature was incorrectly defined as optional;

• DFL (Decrease solitary feature lower bound, change mandatory feature to optional): $DFL(s_{min,max}) = s_{min-1,max}$, if $min > 0$;

• IFL (Increase solitary feature lower bound, change optional feature to mandatory): $IFL(s_{min,max}) = s_{min+1,max}$, if $min < max$;

• DFU (Decrease solitary feature upper bound): $DFU(s_{min,max}) = s_{min,max-1}$, if $min < max$;

• IFU (Increase solitary feature upper bound): $IFU(s_{min,max}) = s_{min,max+1}$.

2) *Incorrect Elements of a Grouped Relation*: one of the grouped features should not belong to the set relation, or a solitary feature should be included into a

set relation;

• AFS (Add feature to a set relation, solitary feature to grouped): $AFS(st_{min,max}(f_1, ..., f_{k-1}, f_{k+1}, ..., f_n), b(f_{k_{i,j}})) = st_{min,max}(f_1, ..., f_{k-1}, f_k, f_{k+1}, ..., f_n)$;

• RFS (Remove features from a set relation, grouped to solitary and set relation to binary): $RFS(st_{min,max}(f_1, ..., f_{k-1}, f_k, f_{k+1}, ..., f_n)) = st_{min,max}(f_1, ..., f_{k-1}, f_{k+1}, ..., f_n)$ and $b(f_{k_{0,1}})$;

3) *Existence of a Set Relation*: faults associated to a wrong set relation. Some child features belong to a set relation with an appropriate cardinality but they should be defined as solitary features;

• RSR (Remove a set relation, create solitary and new binary optional relations): $RSR(st_{min,max}(f_1, ..., f_n)) = b(f_{1_{1,0}}), ..., b(f_{n_{1,0}})$;

4) *Incorrect Cardinality of a Set Relation*: set relation allows that a certain number of grouped features is part of a created product. Once cardinality is responsible by defining this number, a mistake on setting the max or min values can result in a diagram that allows products with more or less features than required; and

• DRL (Decrease set relation lower bound): $DRL(st_{min,max}(f_1, ..., f_n)) = st_{min-1,max}(f_1, ..., f_n)$, if $min > 0$;

• IRL (Increase set relation lower bound): $IRL(st_{min,max}(f_1, ..., f_n)) = st_{min+1,max}(f_1, ..., f_n)$, if $min < max$;

• DRU (Decrease set relation upper bound): $DRU(st_{min,max}(f_1, ..., f_n)) = st_{min,max-1}(f_1, ..., f_n)$, if $min < max$;

• IRU (Increase set relation upper bound): $IRU(st_{min,max}(f_1, ..., f_n)) = st_{min,max+1}(f_1, ..., f_n)$, if $max < n$;

5) *Incorrect Constraint*: this class is associated to depends and excludes constraints, and includes the following cases: (i) the two features of the constraint were incorrectly selected; (ii) the constraint should not exist; (iii) a constraint is absent in the FM.

• FDC (Change depends constraint): $FDC(d(f_a, f_b), f_k \in F) = d(f_b, f_a) | k \neq a, k \neq b$;

• RDC (Remove a depends constraint): $RDC(d(f_a, f_b)) = D - \{d\}$;

• REC (Remove an excludes constraint): $REC(e(f_a, f_b)) = E - \{e\}$;

• CDC (Create a depends constraint): $CDC(f_1, ..., f_n \in F) | f_k = s_{0,max} \in S$ or $f_k = g \in G = d(f_i, f_j) | i, j \leq n, i \neq j$ and $c(f_i, f_j), c(f_j, f_i) \notin C$;

• CEC (Create a excludes constraint): $CEC(f_1, ..., f_n \in F) | f_k = s_{0,max} \in S$ or $f_k = g \in G = e(f_i, f_j) | i, j \in [1, n], i \neq j$ and $c(f_i, f_j), c(f_j, f_i) \notin C$

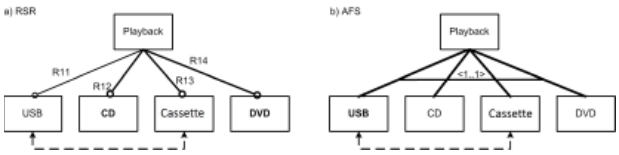


Figure 3. Examples of mutants for FD of Fig. 2

Figure 3 presents two mutants generated by operators RFS and RSR for the FM of Figure 2. Due to space restrictions the FMs are not complete.

3.2 Using the Operators

The operators can be used similarly to the mutation testing of programs, following a process with analogous steps: mutant generation, mutant execution with test cases, and mutation score production.

First of all, mutants are generated by selecting operators to be applied, as well as, a percentage of mutants to be generated by each operator. For example if DFL (decrease solitary feature lower bound) is selected with 10%, only 10% of the existing solitary features will be mutated. Observe that all mutants are valid FMs. If an inconsistent mutant diagram is obtained by applying an operator, it is considered anomalous and is discarded. An example of anomalous mutant in the program testing is a division by zero produced in execution time. An example of anomalous mutant in the feature testing is the production of a void FD, which that does not produce valid products.

The test cases (products) are "executed" with a FM analyser. A mutant is considered dead if the validation of a product by using the mutant produces a different result from the validation of the same product against the original FM. At the end, a mutation score is calculated, given by the number of generated and non-equivalent mutants divided by the number of dead ones. Equivalent and original diagrams generate the same set of products.

Consider the test case (Navigation System, Map Data via CD, Map Data via USB, USB, CD, WMA, MP3). It kills the mutant of Figure 3b, generated by operator AFS. The product is valid for the original FM, however it is not valid for the mutant. This does not happen for the mutant of Figure 3a, generated by operator RSR. The product is valid for both diagrams, hence, the test case is not capable to distinguish them. The analysis of the valid and invalid products can point out modeling faults. At the end a set of valid products to be tested is obtained.

Uses to the approach are testing criterion uses. The mutants can be used to 1) to guide the selection of products, and 2) to evaluate the quality of the test set

(set of products). In the first use, no set T of products is available, a task of test data generation is performed to find products to kill the mutants, until the desired score is obtained (ideally score = 1). In the second one, the mutation score is used as an adequacy measure. In such case the tester has a set T and wants to know how good it is. It is also possible to use the score to compare two test sets T_1 and T_2.

Notice that the uses are not exclusive. If a test set T is available, and its score is not adequate, this set can be improved with additional test cases. The use of such initial test set allows effort reduction in the mutation approach application. In this way, it can be considered complementary to existing ones, such as pair-wise testing. It improves efficacy in terms of revealed faults and offers a coverage measure to evaluate sets of products.

4 Case Study

The approach uses for testing the SPL CAS [17] were evaluated in a case study. We first analyse the applicability of the mutation operators for products selection according to the number of generated mutants and required test data. After this the approach is used to evaluate a set of products generated by combinatorial testing. The pair-wise testing was applied using the AETG algorithm and the framework Combinatorial Tool. The set produced is named here AETGSet.

To apply the fault based approach the mutants were generated by using all of the operators with percentage of 100%. The operators associated to the cloned features DFU and IFU were not used, since framework FAMA, used to execute the mutants, does not work with this kind of feature. The products to kill the mutants were generated manually. The number of mutants generated by each operator, as well as the number of required products are presented in Table 1.

It was generated 268 mutants. We can observe that the operators CDC, CEC, which create depends and excludes constraints, generated the greatest number of mutants, followed by IFL and DFL that change the bounds of solitary features. In another hand, the operators REC and RDC, which are also from the class related to constraint faults, generated lower number of mutants. The first ones, which deal with absent constraints, have a lot of possibilities. The last ones deal with only the constraints that are present in the diagram. Other operators that generated low number of mutants are: IRL and DRU related to the cardinality of a set relation. We find only 3 equivalent mutants, 2 generated by the operator CDC and 1 by IRL.

A total of 27 different products were generated. In many cases there is a correspondence between the number of mutants and necessary products. However, for the operators that generated the greatest numbers of mutants such relation does not exist. For example, the operators CDC and CEC required only few test cases. The number of test cases (products) does not grow proportionally to the number of mutants, what makes the application of the operators feasible in practice. However we intend to conduct other studies to better evaluate the cost according to the FM characteristics.

Table 1. Mutants and required products

Operator	generated mutants	required products	dead m. AETGSet	required p. AETGSet
DFL	10	10	0	0
IFL	12	8	4	3
AFS	2	1	1	1
RFS	6	4	5	4
RSR	2	2	0	0
DRL	2	1	0	0
IRL	1	1	0	0
DRU	1	1	0	0
IRU	2	1	0	0
FDC	2	1	2	2
RDC	2	2	2	2
REC	1	1	1	1
CDC	150	26	127	7
CEC	75	1	75	1
Total	268	27	217	11

In a second step, the mutants were used to evaluate AETGSet. The number of mutants (per operator) killed is presented in Table 1. The set killed 217 mutants, with a score of 0.819. This means that 18% of the mutants could not be killed by the set, particularly the mutants of nine operators. They are: DFL, IFL, RFS, RSR, DRL, IRL, DRU, IRU and CDC. Most of these operators are related to cardinality faults. This points out that the pair-wise testing may not reveal faults from such classes.

The number of AETGSet products used to kill the mutants from each operator is presented in the last column of Table 1. For example, 3 test cases from AETGSet were used to kill 4 mutants generated by the operator IFL. Any other product killed mutants from this operator. This result was obtained, with the evaluation order of products provided by Combinatorial tool. Product 1 was evaluated and the corresponding mutants killed. After this, product 2. If such product 2 did not contribute to increase the score it was not counted. It is noticeable that 11 (out 12) AETGSet products were necessary to kill mutants, 7 of them were used to kill mutants generated by operators CDC. Maybe faults revealed by pair-wise are described by these operators.

5 Conclusions

This paper presented a fault based approach for feature testing of SPL. The idea is to use a mutation based criterion to select products for testing, considering common faults in the FM related to feature management. These faults are described by mutant operators. To apply the operators, a mutation process, similar to the mutation testing of programs, is used. A FM mutant is considered dead if it validates a test case (product) in a different way that the original FM does. At the end a mutation score is obtained that can be used to guide the generation of products, or to evaluate the quality of an available set of products.

The operators were used in a case study that shows the approach applicability. The number of test cases does not grow proportionally to the number of generated mutants. In addition to this, we observe that 18% of mutants were not killed by the set of products generated with pair-wising testing. The pair-wising testing was not capable to reveal faults associated mainly to the cardinality of features. The main faults revealed by this kind of test are related to the use of mandatory and optional features. A conclusion of the study is that the approach should be used in a complementary way to the combination testing, revealing other kind of faults and increasing the confidence that the products of a FM are according to the specification.

The results obtained are limited to the LPS studied and can not be generalized. We are now implementing a tool to automatic generation of products. The tool will allow evaluation experiments with large FMs to refine the introduced operators and to evaluate scalability. We intend to analyse the operators considering the FM characteristics and factors such as cost, in terms of number of mutants generated, and efficacy in terms of revealed faults. Works on mutation test of programs investigate ways to reduce mutation cost without decreasing the score, based on a set of sufficient operators. Studies like this could be conducted in the FM context.

References

[1] D. Benavides, S. Segura, P. Trinidad, and A. Ruiz-Cortés. FAMA: Tooling a framework for the automated analysis of feature models. In *First International Workshop on Variability Modelling of Softwareintensive Systems*, pages 129–134, 2007.

[2] D. Benavides, S. Trujillo, and P. Trinidad. On the modularization of feature models. In *First European Workshop on Model Transformation*, 2005.

[3] I. Cabral, M. B. Cohen, and G. Rothermel. Improving the testing and testability of software product lines. In *International Conference on Software Product Lines: going beyond*, pages 241–255, 2010.

[4] D. M. Cohen, S. R. Dalal, J. Parelius, and G. C. Patton. The combinatorial design approach to automatic test generation. *IEEE Software*, 13(5):83–88, 1996.

[5] M. B. Cohen, M. B. Dwyer, and J. Shi. Coverage and adequacy in software product line testing. In *ISSTA 2006 Workshop on Role of Software Architecture for Testing and Analysis*, pages 53–63, 2006.

[6] E. Engström and P. Runeson. Software product line testing: a systematic mapping study. *Information and Software Technology*, 53(1):2 – 13, 2011.

[7] B. P. Lamancha and M.P. Usaola. Testing product generation in software product lines using pairwise for features coverage. In *International Conference on Testing Software and Systems*, ICTSS'10, pages 111–125, 2010.

[8] B.P. Lamancha, M.P. Usaola, and M.P. Velthius. Software product line testing, a systematic review. In *International Conference on Software Paradigm Trends*, volume 49, pages 78 – 81, 2009.

[9] J.D. McGregor. Testing a software product line. Technical report, Software Engineering Institute, Carnegie Mellon University, CMU/SEI-2001-TR-022, 2001.

[10] S. Oster, M. Zink, M. Lochau, and M. Grechanik. Pairwise feature-interaction testing for SPLs: potentials and limitations. In *15th International Software Product Line Conference*, pages 6:1–6:8, 2011.

[11] G. Perrouin, S. Sen, J. Klein, B. Baudry, and Y. le Traon. Automated and scalable t-wise test case generation strategies for software product lines. In *3rd ICST*, pages 459–468, 2010.

[12] S. Segura, R. M. Hierons, D. Benavides, and A. Ruiz-Cortés. Automated test data generation on the analyses of feature models: A metamorphic testing approach. In *3rd ICST*, pages 35–44, 2010.

[13] P.A.M. Silveira Neto, I.C. Machado, J. D. McGregor, E. S. Almeida, and S.R.L. Meira. A systematic mapping study of software product lines testing. *Information and Software Technology*, 53(5):407–423, 2011.

[14] A. Tevanlinna, J. Taina, and R. Kauppinen. Product family testing: a survey. *SIGSOFT Software Engineering Notes*, 29(2):12–12, 2004.

[15] E. Uzuncaova, S. Khurshid, and D.F. Batory. Incremental test generation for software product lines. *IEEE Trans. Softw. Engin.*, pages 309–322, 2010.

[16] F. van der Linden, K. Schimd, and E. Rommes. *Software Product Lines in Action - The Best Industrial Practice in Product Line Engineering*. Springer, 2007.

[17] S. Weissleder, D. Sokenou, and B.-H. Schlinglo. Reusing state machines for automatic test generation in product lines. In *1st Workshop on Model-based Testing in Practice (MoTiP 2008)*, June 2008.

[18] W.E. Wong, A. Mathur, and J. Maldonado. Mutation x all-uses: An empirical evaluation of cost, strength and effectiveness. In *Soft. Quality and Productivity Theory, Practice, Education and Training*, 1994.

Scrum-based Approach for Analyzing Commonalities and Variabilities in Software Product Lines

Ivonei F. da Silva
Informatic Center, Federal University of Pernambuco
and State University of Western Paraná
Cascavel, Brazil
ifs3@cin.ufpe.br

Tassio Vale
Software Engineering Lab, Federal University of Bahia
Salvador, Brazil
tassio.vale@dcc.ufba.br

Silvio R. L. Meira
Informatic Center, Federal University of Pernambuco,
Recife, Brazil
srlm@cin.ufpe.br

Eduardo S. de Almeida
Reuse in Software Engineering (RiSE), Federal University
of Bahia, and Fraunhofer Project Center (FPC) for
Software and Systems Engineering
Salvador, Brazil
esa@dcc.ufba.br

***Abstract*— Management issues (e.g. long iterations, poor feedback, and lack of adaptations) in the early stages of Software Product Line (SPL) engineering can foster demotivation and considerable effort, mainly in scenarios with volatile domain, technology, or requirements. On the other hand, Agile Software Development (ASD) fosters short iterations, rich feedbacks, and adaptations in single-system development through values, principles, and practices. We believe integrating Scrum practices into SPL early stages to perform commonality and variability analysis can bring a balance between agility and formalism when creating SPL for existing systems. In this paper, we present an outline of our approach to analyze commonality and variability among the existing products based on Scrum practices to deal with issues regarding iteration, feedback, and adaptation. A feasibility study is also discussed describing the application of the approach in the mobile domain.**

Keywords- Scrum, commonality and variability analysis, software product lines, agile software development

I. INTRODUCTION

Software Product Line (SPL) enables software development to manage a set of similar systems through the commonality and variability analysis [1] [2]. Traditionally, SPL engineering aims a considerable upfront planning and design with a heavyweight software process to achieve the organization business goals. Management issues can emerge when performing commonality and variability analysis for SPL scoping and requirements in scenarios with volatility in the domain, technology, or requirements. Examples of such issues are long iterations, poor feedback, and lack of process adaptations/improvements. In practice, there are not approaches that provide commonality and variability analysis with a systematic mechanism to foster iterations, feedbacks, and adaptations [9].

On the other hand, Agile Software Development (ASD) achieves the organization business goals through a set of practices, principles, and values. ASD focuses on people and interactions, working software, customer collaboration, responding to change, and continuous improvement [3] [4]. Unlike SPL engineering, ASD aims a lightweight process and low upfront planning and design. In addition, ASD deals with issues regarding iterations, feedbacks, and improvements through specific values, principles, and practices. These practices are wrapped in agile methods. Scrum is considered one of the most popular agile methods [5] and deals with those issues.

This study deals with management issues when analyzing the commonality and variability for the SPL early stages, scoping and requirements. Thus, this paper presents an approach that integrates Scrum practices into commonality and variability analysis for software product lines. To integrate Scrum on SPL early stages, we adopted a method engineering that defines several ways for combining both approaches [6] [7] [8].

The remainder of this paper is organized as follows: Section 2 describes related works. Section 3 presents the approach for commonality and variability analysis in the SPL early stages. Section 4 presents the Scrum practices applied on commonality and variability analysis. Section 5 discusses our feasibility study, and Section 6 presents a conclusion and outlook of future work.

II. RELATED WORK

Recently, there is a growing interest in investigating the possibilities of combining ASD with SPL engineering [9], [17], since both approaches aim to increase the productivity of teams, reduce the products time to market, reduce the development costs, and improve customer satisfaction.

In this research, we are interested in how the ASD (in particular, Scrum practices) can mitigate issues regarding the long iterations, few feedbacks, and adaptations during the commonality and variability analysis. Some previous studies have addressed related problems.

Noor et. al [18] proposed a collaborative approach to improve existing approaches for reengineering legacy products in the product line planning. The approach aims to build with

speed an initial product map, foster the flexibility and non-prescriptive nature of the process, promote the collaboration of business and technical people with face-to-face interaction, enable the ability to embrace changes during the process, and focus on the technical excellence. Collaborative practices from the area of collaboration engineering provide building blocks for the approach codifying agile principles such as stakeholder involvement and rapid feedback. Although the approach is well complete and deals with an interesting strategy to foster the collaboration among the stakeholders, it does not provide systematic mechanism to specify, inspect, and validate the features, as well as, analyzing the commonality and variability deeper among them.

Carbon et al [19] reported the experience of adopting the product line planning game to improve feedback from application to family engineering. They use the planning game agile practice as the main framework to handle the communication between domain and application engineering. They do not focus on commonality and variability analysis activity, although the approach can be adopted to improve the feedback between development teams.

Tourret [20] aimed to evaluate the feasibility of using an agile method to develop a software product line. In particular, he combines DSDM and XP into SPL engineering, where DSDM is used to wrap XP. The approach seems to be complete for two SPL life-cycles, however, there is not details about how to analyze the commonalities and variabilities during the SPL early phases.

Ghanam and Maurer [21] provided a framework that allows agile organizations to incrementally and reactively construct variability profiles for both existing and new systems. The framework leverages common agile practices such as iterative software development, refactoring, continuous integration and testing to introduce variability into systems only when it is needed. The approach is interesting for scenarios that use test-driven development or intend to use it. The scope of this approach begins when our scope end, i.e., the approaches could be complementary to each other.

These works have provided inspiration for our study, however, they do not address the issues mentioned in this paper.

III. The Proposed Approach

A. Development of the Approach

Some requirements were defined as input to build the process. These requirements were based on basic needs [26] to understand the SPL scoping and requirements through an iterative and incremental strategy [9], [10], [17], [27]. The requirements for the proposed approach are:

1. The approach must identify the features from legacy systems, as well as, the new ones.
2. The approach must analyze the variability and commonality for the identified features.
3. The approach must foster feedbacks among stakeholders.
4. The approach must foster iteration among activities.
5. The approach must be adaptative, when necessary.
6. The approach must foster effort reduction, when possible.

The two first requirements are investigated in the industrial case study previously performed [10], which identified much effort and demotivation with a traditional process for SPL early stages without agility. The requirements 3, 4 and 5 are investigated in several previous studies [9][17][27] that combine SPL and Agile methods emphasizing the importance of the frequent feedbacks, short-iterations, and adaptations in the process for agile SPL. The requirement 6 is highlighted in [26], which mapped sixteen approaches for SPL scoping and identified future work, including case studies and approaches that reduce the effort to perform the SPL early stages.

B. Approach Description

The proposed approach comprises roles, units of work (activities and tasks), work products (artifacts), and iterations (Scrum sprints). They are explained as follows:

1) Roles

All the tasks are performed through the collaboration among stakeholders. In this approach six roles were defined, representing the responsibilities of the stakeholders during the process: business expert, domain expert, product expert, scoping expert, SPL developer and inspector. The **Business expert** plays a key role in analyzing of the market and defining the marketing strategy. The **Domain expert** can be a user, client, sponsor, business analyst, or any group of them. They use a deep knowledge on the business to explain the system tasks to the developers in various levels of details.

The **Product expert** has deep knowledge on product business processes. This role is usually involved in training, deployment, and user manuals of the products. The **Scoping expert** aids in all the tasks of the approach. It has the vision of the scope of the legacy products and their features as a possible product line. The **SPL developer** develops reusable assets for the SPL. This role is involved in all SPL tasks such as *commonality and variability analysis* and *commonality and variability implementation*. Finally, the **Inspector** verifies whether the features, feature model and product map artifacts are developed appropriately according to the templates previously defined.

2) Units of work

Activities and tasks are units of work performed by the roles with specific goals and define inputs and outputs. The activity groups also the tasks with common goals. Each activity or task is associated to some traditional scoping activities [26]. The units of work are organized in a workflow to perform commonality and variability analysis on legacy systems. These activities are: pre-analysis, develop product map, prioritize major features, define sub-features and analyze commonalities/variabilities.

The **Pre-analysis** activity comprises the *analyze market* to evaluate the market regarding needs, characterization of the users, legal cultural constraints and business opportunities. Besides, *identify marketing strategy* task identifies the method of product delivery to the customers. In [32], we have the original proposal, which we based on for create this activity.

In the **Develop product map** activity, the *identify products* task defines which products legacy and no-legacy will compose the line. *Identify major features* task defines the main features for those products. *Group features* task organizes the major features to reveal sub-domains. Finally, the *Develop product map* task produces the product map as a matrix of the products and features that address them. In [33], we have an approach for SPL scoping, which has steps that we considered essential for this activity and the next one.

The **Prioritize major features** activity defines two tasks. First, the *Assess domain potential* task evaluates each sub-domain relevant for the business. This evaluation supports the decision-making when prioritizing them in the second task, *Prioritize major features.*

The **Define sub-features,** where the *identify sub-features* and *mine legacy assets* tasks elicit and specify the sub-features (legacy and no-legacy) of the product line. The *Inspect sub-features* task verifies whether the sub-features are following the templates previously agreed. The *Validate sub-features* task reviews the sub-features with the product owner role. If the product owner (for example, domain experts) specified the sub-features, this task is not necessary. This activity is wrapped by the Scrum practices (*sprint planning*, *review*, and *retrospective*). During this activity, the *daily meeting* is conducted as well. For this activity, we did not map it for any other approach, however we were inspired on [34] and [35] for defining features and analyzing their commonalities and variabilities (next activity).

The **Analyze commonalities and variabilities**, where the *update product map* task refines the product map with the sub-features, as well as, new information such as priority and binding time. The *Develop feature model* task produces the feature model artifact. The *Detect clone* task aids by providing details of similarities and variabilities between legacy products. The *Inspect models* task verifies whether each model (feature model and product map) is consistent, mainly after some sprints, where the models are updated and can become inconsistent. This activity is wrapped by the Scrum practices (*sprint planning*, *review*, and *retrospective)* as well. During this activity, the *daily meeting* is conducted.

3) Work products

The work products are artifacts produced, modified, or read by the stakeholders when performing tasks and activities. Some of the approach artifacts are: market analysis document, product list, product map and feature model. The **Market analysis document** that summarizes information about the customer needs, user characteristics, cultural and legal constraints, business opportunities, and the delivery method of products. Other relevant information can be included such as the business goals. If possible, this artifact should be the input for the approach. The **Product list** that contains the legacy products and new ones. Configurations of the products can be considered a product as well.

Furthermore, the **Product map** that is a matrix of features (major features and/or sub-features) and products. The intersection between features and products indicates whether the feature is present in the product (cell is marked) or not present (cell is not marked). This artifact can provide information about the priority and binding time of the feature as well. The **Feature model** that contains the major features and sub-features of all products. It is a hierarchical view of the relationships among features. In this model, constraints on the features and the indication whether it is mandatory (common to all products) or optional (variability) are visible to the stakeholders. In this model, the *business* and *domain experts* can configure new products by selecting optional features.

IV. INCREASING AGILITY IN THE APPROACH

As stated earlier, we adopted Scrum practices since they provide systematic mechanisms to foster the short iterations, frequent feedbacks, and adaptations in the process, team, artifacts, and so on. It is applied on the *define features* and *analyze commonality and variability* activities, because they need a considerable effort to be accomplished, and the volatility in the domain, technology, or requirements can result in uncertain when perform these activities. The following roles, artifacts, and practices were adopted from Scrum:

- *Roles*. **Scrum master** is a role to help the team to achieve the defined goals through the Scrum principles and practices. **Scope owner (analogy to the product owner)** are the roles that wrap the business and domain experts, and legacy system developer roles. Product owner is part of a scope owner through the product experts. **Team** comprises the scoping and domain experts, SPL developer, inspector, and any other role that defines features and analyzes commonalities and variabilities.
- *Work products*. **Scope backlog** is the artifact that represents the prioritized major features of the scope (all products). **Sprint backlog** is the artifact, sub-set of the scope backlog, which the team works during a sprint. **Dashboard** is the artifact that represents the panel with tasks that can be in one of the following states: to do, doing, or done. In addition, the burndown chart and other useful information can be put in the dashboard as well.
- *Practices*. **Sprint planning** is divided in two meetings. In the first meeting, the scope owner and team review and discuss the goals and the context for the high-priority major features. They agree on the definition of "done", for this approach, the validated feature model, feature list, and product map artifacts represent the idea of "done". The team selects the high-priority major features from the scope backlog and they commit to complete them by the end of the sprint. The team decides how much work they will commit to complete it too. This first meeting focuses on understanding what the scope owner wants. The second meeting focuses on the detailed task planning for how to analyze the commonalities and variabilities of the sprint. **Daily meeting** is a short (15 minutes or less) meeting that happens every workday at an appointed time, where the team reports three aspects about the activities of the sprint: (1) what they were able to get done since the last meeting; (2) what they are planning to finish by the next meeting; and (3) any impediments or problem found. **Review** is a meeting where the team reviews the sprint with the scope owner, addressing the key idea in Scrum (inspect and adapt) about the produced artifacts. **Retrospective** is a

meeting where the team learns with the past experiences (sprint) to improve to the next one, addressing the key idea in Scrum (inspect and adapt) about the performed process.

Fig. 1 shows the main roles, work products, and activities of the approach.

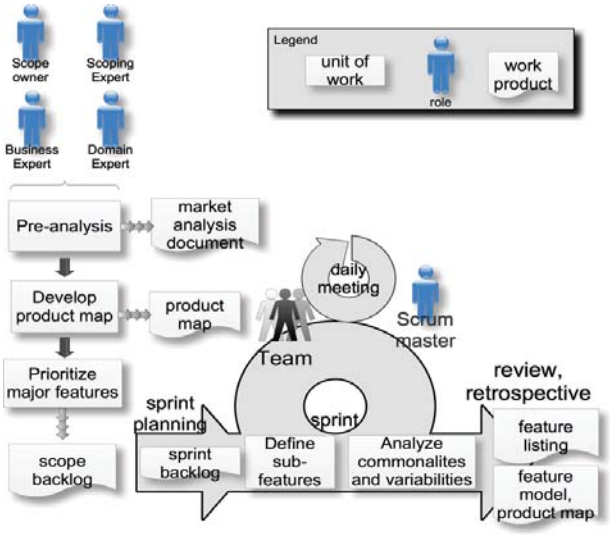


Fig. 1. Overview of the approach

V. Feasibility Study: RescueMe SPL

Our approach is evaluated by a feasibility study on an SPL project, called RescueMe SPL (https://itunes.apple.com/us/app/savi/id590385285?l=pt&ls=1&mt=8). The RescueMe SPL comprises a set of mobile applications to support users in an emergency situation. The basic functionality of a RescueMe application is to send an alert message from the user, in an emergency situation, to his contacts. These emergency situations can be a kidnapping or any other situation that put the user in danger. Based on the message, the contacts can track the user on a Web Map in real time.

One researcher, three Ph. D. students, two master students, and one undergraduate student are conducting the project. The work products are built using textual documents and spreadsheets. All resulting work products of this project are available on the web (Work products available at http://tassiovale.com/rescueme-artifacts.zip).

Regarding implementation, the target platform of the RescueMe SPL is iOS (http://www.apple.com/ios/). Thus, we are using the XCode (https://developer.apple.com/technologies/tools/) as Integrated Development Environment (IDE) and Objective-C as programming language. The implementation of variability in XCode is native, since it provides support for conditional compilation through #ifdef preprocessor directives [30].

A. Results

Some approach results (in terms of roles, activities and work products) for the RescueMe SPL are presented as following.

The involved staff and respective roles are: Expert #1 (Business expert and Scope owner), Developer #1 (Business expert, Domain expert, Product expert, Scoping expert, SPL developer), Developer #2 (Scrum Master), Developer #3 and #4 (Product expert and SPL developer), Developer #5 (Legacy asset developer, Domain expert and Product expert), and Inspector #1 (Inspector).

RescueMe SPL has started by developing a feature model, product map (Fig. 2), and textual descriptions of these features. So far, the product line is consisted of 28 features among five products.

Features / Products			RescueMe Lite	RescueMe Standard	RescuMe Social	RescueMe Pro	RescueMe Ultimate
Access_Control					x	x	x
	Web_Access_Control				x	x	x
	Mobile_Access_Control				x	x	x
Contact			x	x	x	x	x
	Add_Contact		x	x	x	x	x
	Import_Contact		x	x	x	x	x
		Facebook_Import			x	x	x
		Phone_Import	x	x	x	x	x
Destination			x	x	x	x	x
	SMS_Destination		x	x	x	x	x
	Facebook_Destination				x	x	x

Fig. 2. RescueMe SPL – partial product map

The three first activities (pre-analysis, develop product map, and prioritize major features) provided the major features for the product map (Fig. 2). The scope backlog artifact was stemmed from the product map artifact including priority estimative for each main feature.

The last two activities (define sub-features and analyze commonalities and variabilities) provided the feature model and updated the product map (Fig. 2) during the first sprint (named of sprint 0).

During the sprints (sprint 1 and 2), the team began the implementation of the features (prototype). Although there were some refactoring on the product map, the feature model, and feature descriptions, they did not suffer considerable changes.

According to the scope backlog, the sprint 1 involves the implementation of four features: *Contact*, *Import Contact*, *Add Contact* and *Phone Import*. Three features were selected in the second sprint: *Destination*, *SMS Destination* and *Location*. In addition, a set of test cases was specified for each of these features. These features were selected based on the team background with the technology for implementation of the features and importance of the feature for the stakeholder. However, for the future sprints, a prioritization on the other features will performed to guide the choice of the backlog sprint. Figure 4 shows a code sample of the feature-centered variability implementation.

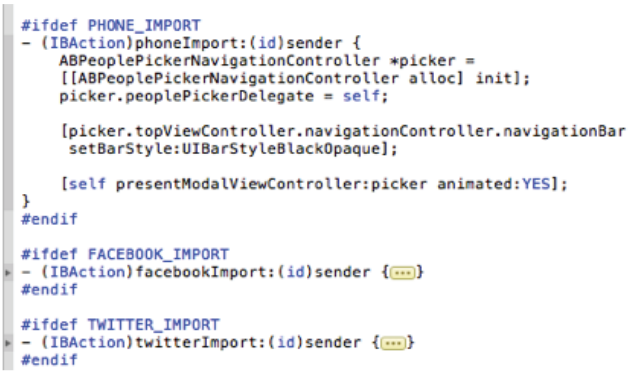


Fig. 4. RescueMe SPL - conditional compilation

B. *Initial Discussion*

The short iterations, with two-week sprints, have provided faster and more frequent feedbacks and moments for improvements. As an example, hidden variabilities could be identified and communicated during the first two-sprints.

The team has not presented problems with the incremental development of the product map and feature model yet. Although, some refactoring will occur in future sprints, the team has showed conviction on the commonalities and variabilities implementation, since, in terms of scope, they were identified and specified comprehensively.

Although new empirical studies are necessary to show that the approach mitigates the management issues (long iterations, poor feedback, and lack of process adaptation), it provided continuous feedbacks and adaptations (improvements) through daily meetings, retrospectives, and reviews. These feedbacks are shared among all stakeholders of the project. For instance, technical problems related to implementation were solved faster and precisely during the daily meetings. The retrospectives have provided sharing of problems and solutions regarding the features for all products (commonalities), and the reviews have provided the validation of the artifacts and functional software. In Noor[18], rapid feedback also occur, however they focused on the way to provide the feedback through the collaborative practices from collaboration engineering. In our case, we focused on the frequency of feedbacks through the short-iterations (two-week sprints) with daily meeting, retrospectives, and reviews.

In Carbon [19], planning game was adopted to deal with feedbacks as well. We believe that there are software development teams that need or prefer this practice than scrum-inspired practices. However, as the scrum method has been more popular [5], an approach with scrum would have more attention. In any case, future works should be performed to evaluate the feedbacks using planning game and scrum practices.

We can contextualize the proposed approach in other studies, as in Ganhan [21], where an output from our approach can be an input for them. They mapped acceptance tests for leaf features from a feature model. As future work, we could also apply their approach in our sprints. In the same way that their work, our approach iteratively and incrementally specifies what are the variabilities in order to provide further configurations of applications. The contrast with traditional upfront SPL approaches, we identified the variabilities as well as the possible variants incrementally. So far, we did not need to do big refactors in the features and variabilities. We believe that in future sprint some changes might emerge, then we might evaluate better.

We did not consider the design and architectural disciplines in the approach. The work of Carbon [31] provides discussions and ideas for future adjusts in our approach. However, in this case, we necessarily must consider other agile practices and principles, such as continuous integration and automated regression testing. As future work, we can try a specific task for SPL platform design in the sprint mixed with those agile practices.

Finally, this approach focuses on commonality and variability analysis in terms of scope for a future SPL. Other approaches [9] do not have the same target. They can address some perspective of feedback, short-iteration, and adaptation, but the focus does not combine the building of commonalities and variability for scoping through the agile management mechanisms for feedback, short-iteration, and adaptation.

An important limitation in this study can be highlighted. The feasibility study is very small (only few variation points in the features) and does not prove that management issues are solved. For bigger scenarios using metrics to evaluate the process, we need to make a new evaluation for the approach in order to discuss the scalability and impact of the approach on the management issues. Thus, we just faced this feasibility study as an evaluation to verify whether the activities and its workflow make sense.

VI. Conclusions and Future Work

The use of Scrum practices into the proposed approach is motivated by the fact that SPL traditional approaches do not offer systematic mechanisms to deal with frequent feedbacks among the stakeholders, short iterations in the process, and adaptations in the approach. These aspects are considered, by agile methodologies, essentials for software development.

This approach deals with frequent feedbacks, short iterations, and improvements in the process, when necessary. These characteristics are fundamentals to deal with volatile scenarios in the domain, technology, or requirements, mainly whether the volatility result in unnecessary effort and demotivation when performing the process activities. As consequence, the approach can reach benefits as less risky (because it tries to see all steps for the development rapidly) and faster return on investment (because it aims to delivery in few sprints weekly). Moreover, this work also aims that the integration between agile and SPL is possible.

As future work, we are performing new sprints to implement the variability management mechanisms (the *#ifdef* preprocessor directives). In addition, we are refining the approach to formally evaluate it in an industrial context with two companies working in the oil and education domains. We believe that this kind of evaluation, using metrics for evaluating efficiency of proposed approach, is very important to prove the feasibility of this approach and increase the maturity of this new research field (Agile and SPL). The evaluation of other

activities, such as design, evolution, and derivation of applications, in the sprint cycles will be investigated as well.

ACKNOWLEDGMENT

This work was partially supported by the National Institute of Science and Technology for Software Engineering (INES - http://www.ines.org.br), funded by CNPq and FACEPE, grants 573964/2008-4 and APQ-1037-1.03/08 and CNPq grants 305968/2010-6, 559997/2010-8, 474766/2010-1 and FAPESB.

REFERENCES

[1] Clements, P. and Northrop, L.: Software Product Lines: Practices and Patterns. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2001)

[2] Pohl, K., Böckle, G., and Linden, F. J. v. d.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag, Secaucus, NJ, USA (2005)

[3] Beck, K. and et. al.: Manifesto for agile software development, http://agilemanifesto.org/ (2001)

[4] Larman, C. and Vodde, B.: Scaling Lean and Agile Development. Thinking and Organizational Tools for Large-Scale Scrum. Addison Wesley, Westford, Massachusetts, USA (2008)

[5] Version-One: The state of agile development: version one survey. http://www.versionone.com/state_of_agile_development_survey/11/ (2012)

[6] Cossentino, M. and Seidita, V.: Composition of a new process to meet agile needs using method engineering (2004)

[7] McGregor, J.: Agile software product lines, deconstructed. Journal of Object Technology, 7(8), 7–19 (2008)

[8] McGregor, J.: Mix and match. Journal of Object Technology, 7(6), 7–16 (2008)

[9] Silva, I. F., Neto, P. A. S. M., O'Leary, P., Almeida, E. S., and Meira, S. R. L.: Agile software product lines: a systematic mapping study. Software, Practice and Experience, 41(8), 899–920 (2011)

[10] Silva, I.F., Neto, P. A. M. S., O'Leary, P., Almeida, E. S., and Meira, S. R. L.: Characterizing software product lines scoping and requirements engineering in a small company: An industrial case study, Journal of Systems and Software (Under evaluation) (2013)

[11] Dybå, T. and Dingsøyr, T.: Empirical studies of agile software development: A systematic review. Information Software Technology, 50, 833–859 (2008)

[12] Cockburn, A.: Agile software development. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2001)

[13] O'hEocha, C., Conboy, K., and Wang, X.: So you think you're agile? In Agile Processes in Software Engineering and Extreme Programming, 11th International Conference, XP 2010, pages 315–324 (2010)

[14] Miller, G. G.: The characteristics of agile software processes. In Proceedings of the 39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems (TOOLS39), TOOLS '01, pages 385–, Washington, DC, USA. IEEE Computer Society (2001)

[15] Abrantes, J. F. and Travassos, G. H.: Characterization of software development agile methods (in portuguese). In Proceedings of Workshop of Fast Development of Applications. Brasilian Symposium of Software Quality (WDRA - SBQS 2007), Porto de Galinhas, PE, Brazil (2007)

[16] Nerur, S., Mahapatra, R., and Mangalaraj, G.: Challenges of migrating to agile methodologies. Communications ACM, 48(5), 72–78 (2005)

[17] Diaz, J., Perez, J., Alarcon, P. P., and Garbajosa, J.: Agile product line engineering - a systematic literature review. Software: Practice and Experience, 41(8), 921–941 (2011)

[18] Noor, M. A., Rabiser, R., and Grünbacher, P.: A collaborative approach for reengineering-based product line scoping. In APLE '06: Proceedings of the 1st International Workshop on Agile Product Line Engineering in conjunction with SPLC 2006. Baltimore, Maryland, USA (2006)

[19] Carbon, R., Knodel, J., Muthig, D., and Meier, G.: Providing Feedback from Application to Family Engineering - The Product Line Planning Game at the Testo AG. Software Product Line Conference, International, 180–189 (2008)

[20] Tourret, R.: Using Agile Methods to Develop Software Product Lines. Master's thesis, Computer Science - University of York, York, England (2006)

[21] Ghanam, Y., Andreychuk, D., and Maurer, F.: Reactive variability management using agile software development. In Agile '10: Proceedings of the international conference on Agile methods in software development, pages 27–34, Orlando, USA (2010)

[22] IEEE: IEEE standard for developing software life cycle processes. IEEE Std 1074-1997 (1998)

[23] Gilb, T.: Principles of software engineering management. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1998)

[24] Ghezzi, C., Jazayeri, M., and Mandrioli, D.: Fundamentals of Software Engineering. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition (2002)

[25] Boehm, B. W.: A spiral model of software development and enhancement. Computer, 21(5), 61–72 (1988)

[26] John, I. and Eisenbarth, M.: A decade of scoping: a survey. In Proceedings of the 13th International Software Product Line Conference, SPLC '09, pages 31–40, Pittsburgh, PA, USA. Carnegie Mellon University (2009)

[27] Silva, I. F., Neto, P. A. M. S., Almeida, E. S., and Meira, S. R. M.: Empirical findings in agile software product lines using a multi-method approach to interpret the evidences. Information Software Technology, (Under evaluation) (2013)

[28] Kang, K. C., Kim, Sajoong, Lee, J., Kim, K., Shin, E., and Huh, M.: FORM: A feature-oriented reuse method with domain-specific reference architectures. Annual Software Engineering 5 (January 1998), 143-168, (1998)

[29] OMG-SPEM, O. M. G.: Software & systems process engineering metamodel specification - spem. http://www.omg.org/spec/SPEM/2.0/ (2008)

[30] Gacek, C. and Anastasopoules, M.. 2001. Implementing product line variabilities. In Proceedings of the 2001 symposium on Software reusability: putting software reuse in context (SSR '01). ACM, New York, NY, USA, 109-117, (2001)

[31] Carbon, R., Lindvall, M., Muthig, D., and Costa, P. "Integrating Product Line Engineering and Agile Methods : Flexible Design Up-front vs . Incremental Design," in *APLE 1st International Workshop on Agile Product Line Engineering*, pp. 1–8, (2006)

[32] Kang, Kyo C., Donohoe, Patrick, Koh, Eunman, Lee, Jaejoon, and Lee, Kwanwoo. Using a Marketing and Product Plan as a Key Driver for Product Line Asset Development. In *Proceedings of the Second International Conference on Software Product Lines* (SPLC 2), Gary J. Chastek (Ed.). Springer-Verlag, London, UK, UK, 366-382, (2002)

[33] Schmid, Klaus. A comprehensive product line scoping approach and its validation. In*Proceedings of the 24th International Conference on Software Engineering* (ICSE '02). ACM, New York, NY, USA, 593-603, (2002)

[34] Kang, Kyo; Cohen, Sholom; Hess, James; Novak, William; and Peterson, A.. Feature-Oriented Domain Analysis (FODA) Feasibility Study (CMU/SEI-90-TR-021). Software Engineering Institute, Carnegie Mellon University, 1990. http://migre.me/er6Ny

[35] van Gurp, J.; Bosch, J.; Svahnberg, M., "On the notion of variability in software product lines," *Software Architecture, 2001. Proceedings. Working IEEE/IFIP Conference on* , vol., no., pp. 45-54, (2001)

Mining Features from the Object-Oriented Source Code of a Collection of Software Variants Using Formal Concept Analysis and Latent Semantic Indexing

R. AL-msie'deen[1], A.-D. Seriai[1], M. Huchard[1], C. Urtado[2], S. Vauttier[2], and H. Eyal Salman[1]

[1]LIRMM / CNRS & Montpellier 2 University, France, {al-msiedee, seriai, huchard, eyalsalman}@lirmm.fr
[2]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 of this SPL must be built as a first step. To do so, it is necessary to mine optional and mandatory features from the source code of the software variants. Thus, we propose, in this paper, a new approach to mine features from the object-oriented source code of a set of software variants based on Formal Concept Analysis and Latent Semantic Indexing. To validate our approach, we applied it on ArgoUML and Mobile Media case studies. The results of this evaluation validate the relevance and the performance of our proposal as most of the features were correctly identified.

Keywords*: Software product line engineering, software variants, feature mining, FCA, LSI.*

1 Introduction

A software product line (SPL) is "a set of software intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and are developed from a common set of core assets in a prescribed way" [1]. A SPL is usually characterized by two sets of features: the features that are shared by all products in the family, which represent the *SPL's commonalities*, and the features that are shared by some, but not all, products in the family, which represent the *SPL's variability*. SPLs are usually described with a *de-facto* standard formalism called *feature model* [1]. A feature model defines all the valid feature configurations. In common software development processes software product variants often evolve from an initial product developed for and successfully used by the first customer. Mobile Media Systems [2] is an example of such a product evolution. These product variants usually share some common features but they are also different from one another due to subsequent customization to meet the specific requirements of different customers [3]. When variants become numerous, switching to a rigorous software product line engineering (SPLE) process is a solution to tame the increasing complexity of all the engineering tasks. To switch to SPLE starting from a collection of existing variants, the first step is to mine a feature model that describes the SPL. This implies to identify the software family's common and variable features. Manual reverse engineering of a feature model for software variants is time-consuming, error-prone, and requires substantial efforts [4]. Assisting this process would be of great help. This paper proposes an approach to mine features from a collection of software product variants in order to define the feature model of the software family[1]. Our approach is based on the identification of the implementation of these features among object-oriented (OO) elements of the source code. These OO elements constitute the initial search space. We use Formal Concept Analysis (FCA) to reduce this search space by first separating common and variable elements and, secondly, dividing the set of variable elements in subgroups. We further use Latent Semantic Indexing (LSI) to define a similarity measure that enables to identify subgroups of elements that characterize the implementation of each possible feature. Our approach is detailed in the remainder of this paper as follows. Section 2 sketches out a background of the used classification techniques. Section 3 presents the principles and main concepts of our approach. Section 4 details the feature mining process step by step. Section 5 describes the experiments that were con-

[1]This work has been funded by grant ANR 2010 BLAN 021902

ducted to validate our proposal. Section 6 discusses related work. Section 7 concludes and provides perspectives for this work.

2 Background: Techniques Used for Classification

This section presents a quick overview of the two techniques – Formal Concept Analysis (FCA) and Latent Semantic Indexing (LSI) – we plan to combine to classify information on software variants in order to extract features from their source code.

2.1 Formal Concept Analysis

Galois lattices and concept lattices [5] are core structures of the Formal Concept Analysis framework (FCA), which enables to extract an ordered set of concepts from a dataset, called a formal context, composed of objects described by attributes. A formal context is a triple $K = (O, A, R)$ where O and A are sets (objects and attributes respectively) and R is a binary relation, *i.e.*, $R \subseteq O \times A$. Several examples of formal contexts are provided in the remaining of this paper. A formal concept is a pair (E, I) composed of an object set $E \subseteq O$ and their shared attribute set $I \subseteq A$. $E = \{o \in O | \forall a \in I, (o, a) \in R\}$ is the extent of the concept, while $I = \{a \in A | \forall o \in E, (o, a) \in R\}$ is the intent of the concept. 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 $C_1 \leq_s C_2$ 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. Let $\mathcal{C}_K$ be the set of all concepts of a formal context K. This set of concepts provided with the specialization order $(\mathcal{C}_K, \leq_s)$ has a lattice structure, and is called the concept lattice associated with K. In our approach, we will consider the AOC-poset (for Attribute-Object-Concept poset), which is the sub-order of $(\mathcal{C}_K, \leq_s)$ restricted to object-concepts and attribute-concepts (AOC-poset ignore concepts empty of declared objects and attributes). AOC-posets scale much better than lattices. Several figures in the remaining of this paper show AOC-posets. For readability's sake, in these diagrams, extents and intents are presented in a simplified form, removing top-down inherited attributes and bottom-up included objects.

2.2 Latent Semantic Indexing

Several IR methods exist such as Vector Space Method (VSM) and Latent Semantic Indexing (LSI) [6]. Both methods assume that software artifacts can be regarded 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. LSI is an advanced IR method. The heart of LSI is singular value decomposition technique (SVD). This technique is used to mitigate noise introduced by stop words like (the, an, above) and to overcome two classic problems arising in natural languages processing: synonymy and polysemy [6]. The effectiveness of IR methods is measured by their recall, precision and F-Measure. For a given query, recall is the percentage of correctly retrieved results to the total number of relevant results, while precision is the percentage of correctly retrieved results to the total number of retrieved results. F-Measure defines a tradeoff between precision and recall with a high value only in cases where both recall and precision are high. All measures have values between [0, 1]. If recall equals 1, all relevant results are retrieved. However, some retrieved results might not be relevant. If precision equals 1, all retrieved results are relevant. However, relevant results might not be retrieved [6]. If F-Measure equals 1, all relevant results are retrieved and only relevant results are retrieved.

3 Approach Basics

This section presents the main concepts and hypotheses used in our approach for mining features from source code. It also shortly describes the example that illustrates the remaining of the paper.

3.1 Goal and Core Assumptions

The general objective of our work is to mine the feature model of a collection of software product variants based on the static analysis of their source code. Mining common and variable features is a first necessary step towards this objective. We consider that "a feature is a prominent or distinctive and user visible aspect, quality, or characteristic of a software system or systems" [7]. Our work focuses on the mining of functional features. Functional features express the behaviour or the way users may interact with a product. As there are several ways to implement features [8], we consider software systems in which functional features are implemented at the programming language level (*i.e.*, source code). We also restrict to OO software. Thus, features are implemented using object-oriented building elements (OBEs) such as packages, classes, attributes, methods or method body elements (local variable, attribute access, method invocation). We consider that a feature corresponds to one and only one set of OBEs. This means that a feature always has the same implementation in all products where it is present. We also consider that feature implementations may overlap: a given OBE can be shared between several features' implementation.

3.2 Features versus Object-oriented Building Elements: the Mapping Model

Mining a feature from the source code of variants amounts to identify group of OBEs that constitutes its implementation. This group of OBEs must either be present in all variants (case of a common feature) or in some but not all variants (case of an optional feature). Thus, the initial search space for the feature mining process is composed of all the OBEs in the existing product variants. For a source code containing n OBEs, the initial search space is the powerset of n deprived of the empty set. As the number of OBEs is high, mining features entails to reduce this search space. Several strategies can be combined to do so:

- separate the OBE set in two subsets, the common features set – also called *common block* (CB) – and the optional features set, on which the same search process will have to be performed. Indeed, as optional (*resp.* common) features appear in some but not all (*resp.* all) variants, they are implemented by OBEs that appear in some but not in all (*resp.* all) variants.
- separate the optional feature set into small subsets that each contains OBEs shared by groups of two or more variants or OBEs that are hold uniquely by a given variant. Each of these subsets is called a *block of variation* (BV). BVs can then be considered as smaller search spaces that each corresponds to the implementation of one or more features.
- identify common atomic blocks (CAB) amongst common block based on the expected lexical similarity between the OBEs that implement a given feature. A CB is thus composed of several CABs.
- identify atomic blocks of variation (ABV) inside of each BV based on the expected lexical similarity between the OBEs that implement a given feature. A BV is thus composed of several ABVs.

All the concepts we defined for mining features are illustrated in the OBE to feature mapping model of Figure 1.

3.3 An Illustrative Example

As an illustrative example, we consider three text editor software variants. *Editor_1* supports core text editing features: *open*, *close* and *print* a file. *Editor_2* has the core text editing features and a new *select_all* feature. *Editor_3* supports *copy* and *paste* features, together with the core ones. In this example, the eventually mined features are presented to better explain some parts of our work. However, we only use the source code of software variants as input of the mining process and thus do not know features in advance.

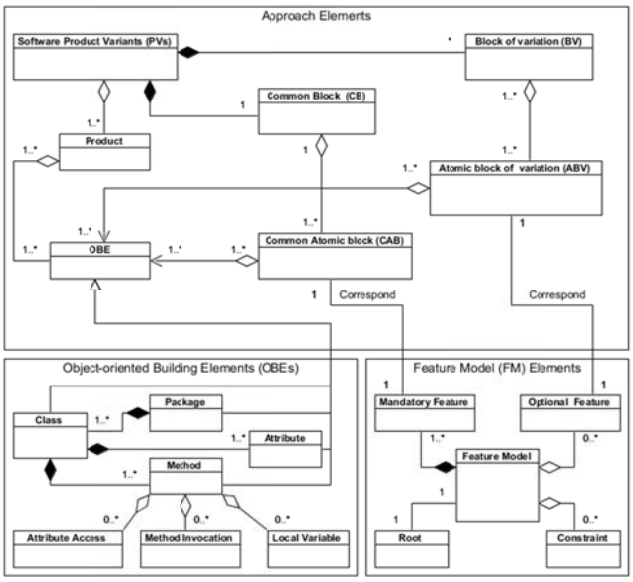


Figure 1: OBE to Feature Mapping Model.

4 The Feature Mining Process

The mapping model between OBEs and features defines associations between these features and the corresponding OBEs. To determine instances of this model, we describe our feature mining process. This process takes the variants' source code as its input. The first step of this process aims at identifying BVs and the CB based on FCA (*cf.* Section 4.1). The second step explores the AOC-poset of BVs to define an order to search for atomic blocks of variation (*cf.* Section 4.2.1). In the third step, we rely on LSI to determine the similarity between OBEs (*cf.* Section 4.2.2). This similarity measure is used to identify atomic blocks based on OBE clusters (*cf.* Section 4.2.3). Figure 2 shows our feature mining process.

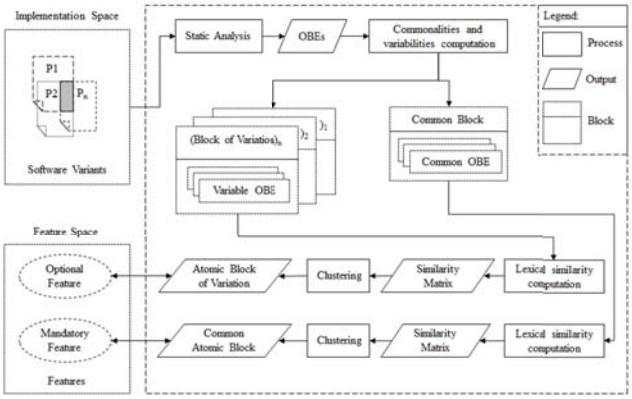


Figure 2: The Feature Mining Process.

4.1 Identifying the Common Block and Blocks of Variation

The first step of our feature mining process is the identification of the common OBE block and of OBE blocks of variation. The role of these blocks is to be sub-search spaces for mining sets of OBEs that implement features.

The technique used to identify the CB and BVs relies on FCA. First, a formal context, where objects are product variants and attributes are OBEs (*cf.* Table 1), is defined. The corresponding AOC-poset is then calculated. The intent of each concept represents OBEs common to two or more products. As concepts of AOC-posets are ordered, the intent of the most general (*i.e.*, top) concept gathers OBEs that are common to all products. They constitute the CB. The intents of all remaining concepts are BVs. They gather sets of OBEs common to a subset of products and correspond to the implementation of one or more features. The extent of each of these concepts is the set of products having these OBEs in common (*cf.* Figure 3).

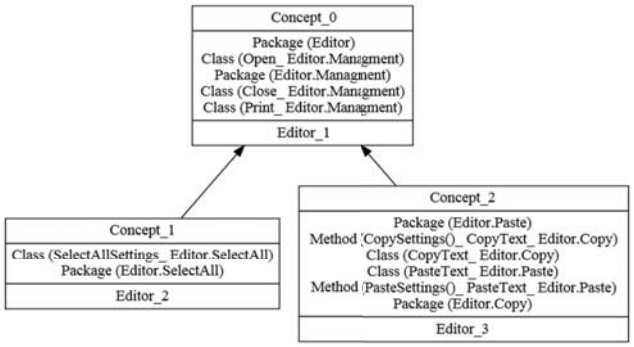


Figure 3: The AOC-poset for the formal context of Table 1.

4.2 Identifying Atomic Blocks

The CB and BVs might each implement several features. Identifying the OBEs that characterize a feature's implementation thus consists in separating OBEs from the CB or from each of the BVs in smaller sets called atomic blocks. Atomic blocks are identified based on the calculation of the similarity between OBEs from the CB or a BV. These similarities result from applying LSI. Atomic blocks are clusters of the most similar OBEs built with FCA as detailed in the following.

Table 1: The formal context for the Text Editor Variants.

	Package(Editor.Managment)	Class (Close_Editor.Managment)	Class (Open_Editor.Managment)	Class (Print_Editor.Managment)	Package(Editor.Copy)	Class (CopyText_Editor.Copy)	Method (CopySettings_CopyText)	Package(Editor.SelectAll)	Class (SelectAllSettings_SelectAll)	Package(Editor.Paste)	Class (PasteText_Editor.Paste)	Method (PasteSettings_PasteText)
Editor_1	×	×	×	×								
Editor_2	×	×	×	×				×	×			
Editor_3	×	×	×	×	×	×	×			×	×	×

4.2.1 Exploring the BV's AOC-poset to Identify Atomic Blocks of Variation

As concepts of the AOC-poset are ordered, the search for atomic blocks of variation (ABVs) can be optimized if exploring the AOC-poset from the smallest (bottom) to the highest (top) block. Results (ABVs) obtained for a concept are used in the exploration of next (*i.e.*, upper) concepts: if a group of OBEs is identified as an ABV, this group is considered as such when exploring the following BV. For Common Atomic Blocks (CAB), there is no such need to explore the AOC-poset as there is a unique CB.

4.2.2 Measuring OBEs' Similarity Based on LSI

OBEs of BVs or of the CB respectively characterize the implementation of optional and mandatory features. We base the identification of subsets of OBEs, which each constitutes a feature, on the measurement of lexical similarity between these OBEs. This similarity measure is calculated using LSI. We rely on the fact that OBEs involved in implementing a functional feature are lexically closer to one another than to the rest of OBEs. To compute similarity between each pair of OBEs in the CB and BVs, we proceed in three steps: building the LSI corpus, building the term-document matrix and the term-query matrix for each BV and for the CB and, at last, building the cosine similarity matrix.

Building the LSI corpus. In order to apply LSI, we build a corpus that represents a collection of documents and queries. In our case, each OBE in the block represents both a document and a query. To be processed, the document and query must be normalized (*e.g.*, all capitals turned into lower case letters, articles, punctuation marks or numbers removed). The normalized document generated by analyzing the source code of an OBE is splitted into terms and, at last, word stemming is performed.

Building the term-document and the term-query matrices for each block. All blocks (the CB and all BVs) are considered and applied the same process. The term-document matrix is of size $m \times n$ where m is the number of terms used in a normalized document corresponding to an OBE and n the number of OBEs in a block. In the same way, a term-query matrix is of size $m \times j$ where m is the number of terms and j the number of OBEs. Each column in the term-query matrix represents a vector of OBEs. Terms for both matrices are the same because they are extracted from the same block.

Building the similarity matrix. Similarity between OBEs in each BV or in the CB is described by a cosine similarity matrix whose columns and rows both represent vectors of OBEs: documents as columns and queries as rows. Similarity is computed as a cosine similarity given by Equation 1, where Q_i is a query vector, D_j is a document vector and W_i and W_j range over weights of query and document vectors, respectively.

$$CosineSimilarity(Q_i, D_j) = \frac{\sum_{i=1}^{n} W_i * W_j}{\sqrt{\sum_{i=1}^{n} W_i^2 \sum_{j=1}^{n} W_j^2}} \quad (1)$$

4.2.3 Identifying Atomic Blocks Using FCA

We then use FCA to identify, from each block of OBEs, which elements are similar. To transform the (numerical) similarity matrices of previous step into (binary) formal contexts, we use a threshold. 0.70 is the chosen threshold value (a widely used threshold for cosine similarity [6]) meaning that only pairs of OBEs having a calculated similarity greater than or equal to 0.70 are considered similar. Table 2 shows the formal context obtained by transforming the similarity matrix corresponding to the BV of *Concept_2* from Figure 3. As an example, in the formal context of this table, the OBE *"Method PasteSetting PasteText"* is linked to the OBE *"Class PasteText Paste"* because their similarity equals 0.99, which is greater than the threshold. However, the OBE *"Method CopySettings CopyText"* and the OBE *"Class PasteText Paste"* are not linked because their similarity equals 0.18, which is less than the threshold. The resulting AOC-poset is composed of concepts the extent and intent[2] of which group similar OBEs.

Table 2: Formal context of *Concept_2*.

	Class CopyText Copy	Class PasteText Paste	Method CopySettings CopyText	Method PasteSetting PasteText	Package Copy	Package Paste
Class CopyText Copy	×		×		×	
Class PasteText Paste		×		×		×
Method CopySettings CopyText	×		×		×	
Method PasteSetting PasteText		×		×		×
Package Copy	×		×		×	
Package Paste		×		×		×

For the text editor example, the AOC-poset of Figure 4 shows two atomic blocks of variation (that correspond to two distinct features) mined from a single block of variation (*Concept_2* from Figure 3). The same feature mining process is used for the CB and for each of the BV.

Concept_0
Class CopyText Copy Method CopySettings CopyText package Copy
package Copy Method CopySettings CopyText Class CopyText Copy

Concept_1
Class PasteText Paste Method PasteSetting PasteText Package Paste
Class PasteText Paste Package Paste Method PasteSetting PasteText

Figure 4: Atomic Blocks Mined from *Concept_2*.

5 Experimentation

To validate our approach, we ran experiments on two Java open-source softwares: Mobile Media[3] and ArgoUML[4]. We used 4 variants for Mobile Media, 10 for ArgoUML. The advantage of having two case studies is that they implement variability at different levels. In addition, Mobile Media and ArgoUML variants are well documented and their feature model is available for comparison to our results and validation of our proposal. Table 3 summarizes the obtained results for each software product variant. For readability's sake, we manually associated feature names to atomic blocks, based on the study of the content of each block and on our knowledge on software. Of course, this does not impact the quality of our results.

Table 3: Features Mined from Mobile Media and ArgoUML Softwares

Case Study	Feature		Evaluation Metrics			
Mobile Media Features	Common	Optional	K	Precision	Recall	F-Measure
Album Management	×		0.05	83%	62%	70%
Splash Screen	×		0.05	71%	57%	63%
Create Album	×		0.05	81%	58%	67%
Delete Album	×		0.05	80%	62%	69%
Create Photo	×		0.05	81%	52%	63%
Delete Photo	×		0.05	78%	63%	69%
View Photo	×		0.05	87%	68%	76%
Exception handling		×	0.03	100%	70%	82%
Edit Photo Label		×	0.02	100%	77%	87%
Favourites		×	0.04	100%	80%	88%
Sorting		×	0.06	100%	78%	87%
ArgoUML Features	Common	Optional	K	Precision	Recall	F-Measure
Class Diagram	×		0.03	72%	56%	63%
Diagram		×	0.06	100%	80%	88%
Deployment Diagram		×	0.05	100%	74%	85%
Collaboration Diagram		×	0.06	100%	67%	80%
Use Case Diagram		×	0.03	100%	64%	78%
State Diagram		×	0.03	100%	69%	81%
Sequence Diagram		×	0.02	100%	67%	80%
Activity Diagram		×	0.06	100%	63%	77%
Cognitive Support		×	0.01	100%	70%	82%
Logging		×	0.02	100%	60%	75%

Results show that precision appears to be high for all optional features. This means that all mined OBEs grouped as features are relevant. This result is due to search space reduction. In most cases, each BV corresponds to one and only one feature. For mandatory features, precision is also quite high thanks to our clustering technique that identifies ABVs based on FCA and LSI. However, precision is

[2]Here, intents and extents are the same. This is because the similarity matrix (and, consequently, the formal context) is symmetric.

[3]http://homepages.dcc.ufmg.br/~figueiredo/spl/

[4]http://argouml-spl.tigris.org/

smaller than the one obtained for optional features. This deterioration can be explained by the fact that we do not perform search space reduction for the CB. Considering the recall metric, its average value is 66% for Mobile Media and 67% for ArgoUML. This means most OBEs that compose features are mined. We have manually identified OBEs which should have been mined and were not. We found that these non-mined OBEs used different vocabularies from mined OBEs'. This is a known limitation of LSI which is based on lexical similarity. Considering the F-Measure metric, our approach has values that range from 63% to 88%. This means that most OBEs that compose features are mined and shows the efficiency of our approach. The most important parameter to LSI is the number of chosen term-topics (*i.e.*, Number of topics (K)). A term-topic is a collection of terms that co-occur frequently in the documents of the corpus. We need enough term-topics to capture real term relations. In our work we cannot use a fixed number of topics for LSI because we have blocks of variation (*i.e.*, partitions) with different sizes. The column (K) in Table 3 shows the K value for each feature.

6 Related work

In our previous work [8] we present an approach for feature location in a collection of software product variants based on FCA by distinguishing between the *common block* (*i.e.*, CB) and *blocks of variation* (*i.e.*, BVs). In this paper we extended our previous work by distinguishing between the common features that appear in the *common block* and the optional features that appear in the same *block of variation* based on the lexical similarity between OBEs. An inclusive survey about approaches linking features and sources code in a single software is proposed in [9]. The approach proposed by Ziadi *et al.* [4] is the closest to ours. They identify all common features as a single mandatory feature. Moreover, they do not distinguish between optional features that appear together in a set of variants. Their approach doesn't consider the method body. Rubin *et al.* [10] present an approach to locate optional features from two product variants' source code. They do not consider common features. They also are limited to only two variants. Xue *et al.* [3] propose an automatic approach to identify the traceability link between a given collection of features and a given collection of source code variants. They thus consider feature descriptions as an input.

7 Conclusion

In this paper, we proposed an approach based on FCA and LSI to mine features from the object-oriented source code of software product variants. We have implemented our approach and evaluated its produced results on two case studies. Results showed that most of the features were identified. The threat to the validity of our approach is that developers might not use the same vocabularies to name OBEs across software product variants. This means that lexical similarity may be not reliable in all cases to identify common and variable features. In future work, we plan to combine both textual and semantic similarity measures to be more precise in determining feature implementation.

References

[1] P. C. Clements and L. M. Northrop, *Software product lines: practices and patterns*. Addison-Wesley, 2001.

[2] L. P. Tizzei, M. Dias, C. M. F. Rubira, A. Garcia, and J. Lee, "Components meet aspects: Assessing design stability of a software product line," *Inf. Softw. Technol.*, vol. 53, no. 2, pp. 121–136, Feb. 2011.

[3] Y. Xue, Z. Xing, and S. Jarzabek, "Feature location in a collection of product variants," in *19th WCRE Conference*. IEEE, 2012, pp. 145–154.

[4] T. Ziadi, L. Frias, M. A. A. da Silva, and M. Ziane, "Feature identification from the source code of product variants," in *15th CSMR Conference*. IEEE, 2012, pp. 417–422.

[5] B. Ganter and R. Wille, *Formal Concept Analysis, Mathematical Foundations*. Springer-Verlag, 1999.

[6] A. Marcus and J. Maletic, "Recovering documentation-to-source-code traceability links using latent semantic indexing," in *25th ICSE Conference*. IEEE Computer Society, 2003, pp. 125–135.

[7] 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.

[8] R. AL-Msie'deen, A. D. Seriai, M. Huchard, C. Urtado, S. Vauttier, and H. E. Salman, "Feature location in a collection of software product variants using formal concept analysis," in *ICSR '13 Conference*. Springer, 2013, pp. 302–307.

[9] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk, "Feature location in source code: a taxonomy and survey," *Journal of Software: Evolution and Process*, pp. 1–54, 2012.

[10] J. Rubin and M. Chechik, "Locating distinguishing features using diff sets," in *27th ASE Conference*, ser. ASE 2012. ACM, 2012, pp. 242–245.

Model-Driven Generation of Context-Specific Feature Models

Thibaut Possompès*, Christophe Dony*, Marianne Huchard*, Chouki Tibermacine*
*LIRMM, CNRS and Montpellier 2 University
Montpellier, France
{possompes, dony, huchard, tibermacin}@lirmm.fr

***Abstract*—Software Product Lines (SPL) aim at deriving software architectures or systems from a software artifact base. Configuring the SPL to derive a new product is now usually done by selecting appropriate software features in a kind of models, called feature models. In some situations, a feature represents a software artifact associated to an element *e* of a context the software product will manage. Such a feature and its associated software artifact may be cloned according to the number of occurrences of *e* in the context and constraints have to be respected. Hence, the feature model proposed to users for configuration has to be adapted in a new dedicated phase according to the context elements. We propose a model-driven engineering approach for transforming a generic feature model according to a context model that a derived software product will manage. More precisely this paper describes an original model transformation able to generate context specific feature models including duplicated features, and removing inappropriate features. Our transformation is validated on a smart building optimization software case study.**

I. Introduction

Configuration options of a software product line (SPL) to generate a new product are nowadays commonly represented with a feature model [9]. A feature represents a component of a product derived from a product line or a functionality that this product provides. A feature model indicates which choices (features) are mandatory or optional in some conditions, and how one choice can impact another one (feature interdependencies and constraints). Software products (applications) are built from an SPL by selecting features from such a feature model. A set of selected features is commonly called a product configuration.

In various situations, some features are semantically associated to elements of a context generic model, describing concepts, that can be present in the context in which (or for which) the application will be deployed. For example in the case of a smart building energy-optimization software, the feature "solar optimization" is semantically associated to the context generic model concept `Solar-panel` and should only be proposed in a configuration feature model if the building to be optimized (described by a model, instance of the context generic model) has some solar panels (instances of `Solar-panel`). As a corollary, such a software could be configured to manage differently each occurrence of a given electronic appliance or physical infrastructure [3].

Hence, a product configuration depends, on the one hand, on the product features, and on the other hand, on the number of occurrences of the context concept instances. In a feature model, cloning features and identifying inappropriate ones is a long and error prone task because each context-specific feature must be checked regarding each context concept instance.

Various approaches have been proposed to configure a feature model according to a context [7], [8]. In this paper we propose an original solution to automatically generate an optimized feature model (called a context-specific feature model), that conforms to a given deployment context. A context-specific feature model only includes features that make sense in the context. The features are appropriately cloned depending on the number of context elements. We propose a model transformation algorithm which uses as inputs a generic feature model (representative of a SPL global set of functionalities), a model of context concepts and an instance of the model of the context.

We validate our proposal in the context of the development of a smart building management system software product line (RIDER[1] project [10]). The project aims at creating a global intelligent system to perform energy optimization.

Section II presents a motivating example from the RIDER project. Section III presents a global view of our approach. Section IV details the algorithms used for performing the adaptation of feature models. Section V discusses the related work, and Section VI concludes this paper and gives several perspectives for this work.

[1]The RIDER project ("Research for IT as a Driver of EneRgy efficiency") – http://rider-project.com/ – is led by a consortium of several companies and R&D laboratories, including IBM and the LIRMM, interested in improving building energy efficiency.

II. Motivating Example – a smart building case study

A RIDER software product purpose is to enhance lighting, heating, ventilation and air conditioning usages to save energy in buildings. A RIDER product is made of interfaces with building management systems (BMS), of several optional modules to add further functions (physical simulation, optimization algorithms, visualization tools, *etc.*), and a component allowing to orchestrate input and output data. The data orchestration component purpose is to decide how to manage incoming data and energy optimization computation results. For example, the physical simulation module requires data related to spaces that are instrumented with temperature and humidity sensors. A RIDER software product uses a representation of the building it will drive. This representation is also called building information model (BIM). It is able to represent static (*e.g.*, blueprints) as well as dynamic (*e.g.*, sensor measures) information.

An instance of the building model is used as a cornerstone to leverage information from building managers and energy optimization experts [2], [6]. It can gather into a single model information such as: 1) 3D geometric data for visualization, 2) electric, 3) Heating Ventilation and Air Conditioning (HVAC), 4) blueprints, 5) various building components along with their size and physical properties for simulation purposes, 6) cost and project management-related information.

Each additional module function is modeled in the RIDER feature model. Some of them are related to the elements of the BIM. For example, 3D information is required to provide 3D visualization features. If it is missing then 3D visualization features are not available. If the information is available on some parts of the building, the visualization features are available only on those parts. When configuring a new product, it is important to know which parts of the building will be properly optimized, and to know which new equipments must be added to allow these features to properly work. More generally, each feature requires to consider if it can be duplicated, which context elements determine how many times it can be duplicated, and which constraints must be satisfied by the context element to make the feature available.

Next section introduces our approach and describes the four models involved in this approach.

III. Approach overview

Let us introduce in this section our terminology and give a general overview of our approach. A *generic feature model* (FM) of a software product line application represents the features globally available in the application (called generic features). Each generic feature can have a semantic relation (depends on) with one or several context model concepts. A *context-specific feature model* (CSFM) represents features relevant to a given context model instance (called context-specific features). Each context-specific feature (CSF) of such a model relates to a generic feature of the FM and, if this generic feature depends on a context concept, to one context concept instance. Our purpose is to automatically generate a CSFM, made of all possible CSFs, by analyzing associations between features of a generic feature model with concepts of a context model (*e.g.*, a building model) and their instances (elements of a concrete building).

The obtained CSFM allows stakeholders to choose CSFs for creating a product configuration adapted to the environment.

Our approach integrates the four models shown in Figure 1. The *context model* (CM) describes the context information that a software product manages. It can be presented by creating a domain specific language (DSL) or a UML model. CM is a set of connected *concepts*. In our case study, this model is the building infrastructure model which contains concepts such as, *Building, Storey, Zone, Space, Sensor*. It is created with the help of context domain specialists. Some concepts are hierarchically related with a specific relation. For example, we have the hierarchical relation *Building* $\rightarrow$ *Storey* $\rightarrow$ *Space* $\rightarrow$ *Sensor*.

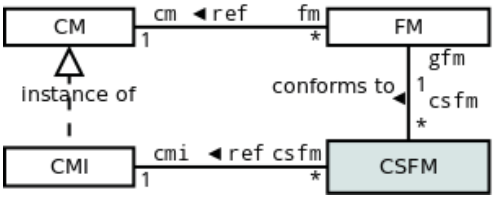


Figure 1. Approach overview

The *generic feature model* (FM) is, for a given CM, a multiplicity-based feature model which is based upon Czarnecki et al. [4] definition. We extended it to make it possible to associate a feature to a CM concept. This association has a multiplicity. It is composed of features (denoted by f) organized in a tree. The root of this tree is denoted by r_f. A feature associated with context concepts can be duplicated according to the instances of those concepts and their multiplicity. In our case study, it is used to describe all the possible features of an energy optimization software. In our approach, this model is not directly used to configure a new product. This model must be first adapted to a specific context,

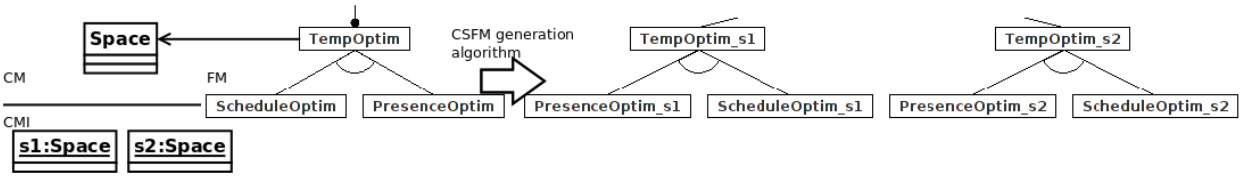


Figure 2. CSFM generation example

which is described by the *context model instance*.

A feature can be associated to none, one or several concepts of the CM. It means that the feature can be duplicated for each instance of an associated concept. A constraint, associated to a feature, can be used to determine which properties must have an instance to duplicate a feature. For example, when CM is written in UML, constraints are in OCL. A given feature f can have a group g gathering its sub-features. A group is used to specify how many grouped features can be selected.

The *context model instance* (CMI) is an instance of the context model. It describes instances of the concepts of CM. If the CM is similar to a UML class diagram, the CMI is similar to a UML instance diagram. The elements of the CMI model are called *instances*. There is also a hierarchical relation between the instances of concepts hierarchically related. The CMI describes the specific context that will be managed by a software product. In our case study, it consists in modeling a specific building that will be managed by a new energy optimization software product, instance of the software product line. It is created with the help of building owners and managers.

The *CSFM* is a kind of feature model resulting from the adaptation of FM to a given context CMI. It is the set of context-specific features (CSF) that can be chosen to build a new product to be used in a given context. A CSF associates a feature f, to an instance i. i must be an instance of one of the concepts associated to f. If f is not associated to a concept then $i = \varnothing$. The CSFs are organized as a tree whose root is denoted by r_φ. A CSF φ can have a group g_φ gathering its sub-features. g_φ must be related to an existing group g of the feature f, f being associated to φ. In our case study, a realistic CMI, of a building b, can have hundreds of instances that have to be considered to create the CSFM to configure the product for b. Our approach proposes an algorithm to generate this model automatically.

The models CM, FM, and CMI are provided as input of our adaptation process. The output of the process is a CSFM to be filled to create a new product configuration. Groups and multiplicities are also adapted in the CSFM. The constraints associated to concepts are checked after having generated the CSFM. We do not detail constraints checking here due to space limitation. Figure 2 depicts excerpts of a CM, CMI, FM, and a $CSFM$. The features *TempOptim, ScheduleOptim* and *PresenceOptim* are associated to the concept *Space* (only one link is shown to simplify the diagram). The $CSFM$ generation algorithm duplicated the feature sub-tree whose root is *TempOptim* two times. The duplicated CSF sub-trees are associated respectively to the instances *s1* and *s2*.

The next section presents the algorithm generating $CSFM$s.

IV. Context-Specific Feature Model Generation Algorithm

The CSFM generation algorithm traverses the feature model in depth-first order. We consider that the models FM, CMI, CM, and $CSFM$ (which is empty at the beginning) are global data common to all following algorithms.

Algorithm 1 initializes the CSFM generation algorithm and returns the resulting CSFM. It creates the root context specific feature and, for each sub-feature of the root feature of the feature model, calls the recursive procedure featureTreeTraversal to build the CSFM.

Algorithm 1: Main procedure of the CSFM generation algorithm
Input: The models *CM*, *CMI*, and *FM*
Result: A *CSFM* model built according to the *CM*, *CMI*, et *FM* models
Initialize an empty CSFM.
r_φ is the root CSF of CSFM, it is associated to the root feature r_f of FM.
foreach *sub-feature* f *of the FM root feature* r_f **do**
`featureTreeTraversal` (*f*, r_φ)
end

The procedure featureTreeTraversal builds the CSFM recursively. It requires two parameters: A feature f for which related CSFs will be created, a CSF φ_{parent} which will be the parent of the created CSFs.

Procedure featureTreeTraversal(f, φ_{parent})
Input: A feature f from which the FM is traversed. A parent CSF φ_{parent}, such that f parent feature is associated to φ_{parent}.
Result: Updates the *CSFM* according to the *CM*, *CMI*, and *FM* models.
if *f is not associated to a concept and φ_{parent} is not associated to an instance* **then**
$\varphi =$ `addSpecificFeature(`$f, \varphi_{parent}, \varnothing$`)`
foreach *sub-feature f' of f* **do**
`featureTreeTraversal(`f', φ`)`
end
if *the CSF parent φ_{parent} is not associated to an instance and f is associated to a concept* **then**
foreach *instance i that is an instance of the concept associated to f* **do**
$\varphi =$ `addSpecificFeature(`f, φ_{parent}, i`)`
foreach *sub-feature f' of f* **do**
`featureTreeTraversal(`f', φ`)`
end
end
if *the CSF φ_{parent} is associated to an instance and f is associated to a concept* **then**
foreach *instance i which is either the same instance that is associated to φ_{parent} and that is an instance of the concept associated to f, or an instance that is hierarchically below the instance associated to φ_{parent} and that is an instance of a concept associated to f* **do**
$\varphi =$ `addSpecificFeature(`f, φ_{parent}, i`)`
foreach *sub-feature f' of f* **do**
`featureTreeTraversal(`f', φ`)`
end
end

The CSFs are created differently in three cases:

1) The evaluated feature f is not associated to any concept and the parent CSF φ_{parent} is not associated to an instance. Then, one CSF is created, and the procedure is called recursively for each sub-feature of f.
2) The parent CSF φ_{parent} is not associated to an instance and the evaluated feature f is associated to a concept. Then, a CSF is created for each instance whose concept is associated to f, and the procedure is called recursively for each instance and for each sub-feature of f.
3) The parent CSF φ_{parent} is associated to an instance and the evaluated feature f is associated to a concept. A CSF is added either with the same instance as φ_{parent} or with each context concept instance which is hierarchically below the instance associated to φ_{parent}. The procedure is then called recursively for each sub-feature of f and for each instance hierarchically below φ_{parent} instance.

The function addSpecificFeature creates, and returns, a new CSF in the CSFM. It requires three parameters: the feature f which will be referenced by the CSF, the parent CSF φ_{parent}, and an instance that will be also referenced by the CSF. As seen before, a CSF references a feature, and either an instance or nothing. First, a new CSF φ is created. Its parent CSF is φ_{parent}, and it is associated to f and i. The lower bound of its multiplicity is equal to the maximum between the lower bounds of the multiplicity on the relationship between the concept whose i is the instance and f, and of the multiplicity on f. The upper bound of its multiplicity is equal to the minimum between the upper bounds of the multiplicity on the relationship between the concept whose i is the instance and f, and of the multiplicity on f.

Then, the procedure addNewCSFtoGroup is called to add the new CSF to a group if f belongs to a group in the FM. If the new CSF is associated to an instance, it also must belong to a group whose multiplicity is the same as the feature associated to the new CSF. This group does not exist in the FM. Its purpose is to transpose in the $CSFM$ the multiplicity of the feature f to guarantee that the number of CSF that can be chosen in a configuration respects f multiplicity.

The procedure addNewCSFtoGroup updates the CSFM by creating groups considering those existing in the FM. It takes two parameters: the new CSF φ, and its parent φ_{parent}. If the feature f associated to φ belongs to a feature group g then φ must also belong to a group g_φ related to g. The group is created only when there is at least two CSF in it. Otherwise, the CSF is either mandatory or optional according to f multiplicity.

V. Related Work

Formal semantics of feature models have been defined in [13] for many different kinds of feature models. We chose a semantics based upon Czarnecki et al. cardinality-based feature models [4] as described in ou previous work [14]. They created a staged configuration process [5] in which they specialize the feature model to restrict the multiplicity of features. It is not applicable to our situation, because a CSFM is not a specialization of a generic feature model. Indeed, each context-specific feature adds information about the context concept

Function addSpecificFeature(f, φ_{parent}, i) : φ

Input: a feature f , a CSF φ_{parent}, and an instance i

Output: Updates the CSFM with a new CSF, and returns the new CSF.

Creates a new CSF φ, sub-CSF of φ_{parent}, associated to the feature f and the instance i. The lower bound of its multiplicity is equal to the maximum between the lower bounds of the multiplicity on the relationship between the concept whose i is the instance and f, and of the multiplicity on f. The upper bound of its multiplicity is equal to the minimum between the upper bounds of the multiplicity on the relationship between the concept whose i is the instance and f, and of the multiplicity on f. The procedure `addNewCSFtoGroup`($\varphi, \varphi_{parent}$) is called to add φ to a group is f belongs to a group in the FM.

if φ *is associated to an instance* **then**
  if *there is no existing CSF group in* φ_{parent} *in which there is other CSF associated to the same feature* **then**
    Adds φ to the CSF group.
  else
    Creates a CSF-group with the same multiplicity as f, and adds φ to this group.
  end
end
return φ

Procedure addNewCSFtoGroup(φ, φ_{parent})

Input: The CSF φ and φ_{parent} such that φ must be a sub-CSF of φ_{parent}.

Result: Updates the CSFM to make φ a sub-CSF of φ_{parent}.

Let f being the feature associated to φ.
if f *belongs to a feature group* g **then**
  if *there is a CSF group* g_φ *in* φ_{parent} *related to* g **then**
    Adds φ to the group g_φ.
  else
    Creates a new CSF group related to the group g, and adds φ and φ' to this group.
  end
end

instance it is associated with. Even if our case study does not require it, we could use our work to automate a staged-configuration process. The generic feature model could be specialized into a refined generic feature model before generating the CSFM, and the CSFM could be specialized into a refined CSFM and configured through a staged configuration process.

There are several solutions allowing to perform product configuration choices according to a given context. Voelter et al. [15] detail an approach where negative and positive variability are used to remove or add concepts to a custom DSL which seems to correspond to our business model. However, their approach could not solve our concerns because we needed to adapt the feature model. We address the opposite concern, we adapt the feature model to an imposed business model.

Acher et al. [1] work in the context of self adaptive and dynamic systems. They bind a context model, modeled with a feature model, with a feature model describing the application. They are interested by runtime adaptation while we are concerned by the design time adaptation. Changes applied in the context feature model are automatically reflected on the application configuration model thanks to ECA rules [12]. We propose to adapt the feature model rather than a configuration model. In our case, the context is a business model provided by some stakeholders. This model is also used by the application to describe the managed data and not specifically created for the product line specification.

Quinton et al. [11] derive software products in the context of applications for mobile phones. They consider a feature model for the application and a feature model for mobile devices. They configure and generate an application model using the application feature model, and then check if the model is consistent with a set of mobile devices represented by the mobile device feature model. We address a different problem: the business model (*e.g.*, a building model) is imposed and we have to propose to the stakeholders a feature model adapted to the business model, in order to allow them to configure an application consistent with their environment.

VI. Conclusion

We presented in this paper an approach able to adapt generic feature models to a business context. It allows us to produce a CSFM according to a context model instance representing the context in which the future product will run. Our approach allows to automatically determine whether each feature related to a context concept can be cloned in a given context by checking

constraints against context concept instances. Hence, each generated product configuration is specific to the instance of the context model it has been made with. Then, it enables us to generate software product architectures and implementations specific to a context. This approach automates a process that would otherwise requires to compare hundreds of features with hundreds of context concept instances.

Our methodology has been designed in a generic way to be reused in different domains. The prototype has been implemented with UML models and UML profiles, for modeling the context model and the generic feature model. We developed a tool as an Eclipse RCP platform. The Eclipse RCP platform takes as input the XML files representing the business model and the feature model. They are generated by an XSLT transformation from the XMI versions of the UML models.

We validated our approach in the RIDER project on a building meta-model used to describe smart buildings. The building meta-model has been modeled as a UML model on which classes and associations were stereotyped to represent navigable elements. The feature model was built with our UML profile for feature models [14].

Next, we intend to create views on the CSFM to facilitate feature selection. They could show features related to a stakeholder concern, or allow choosing several features at the same time, *e.g.*, all the clones of a feature. In future work, we want to enable the configuration of new products according to features existing in other products to facilitate their interoperability.

REFERENCES

[1] M. Acher, P. Collet, F. Fleurey, P. Lahire, S. Moisan, and J. P. Rigault. Modeling context and dynamic adaptations with feature models. In *4th International Workshop Models@ run. time at Models*, volume 9, 2009.

[2] António Grilo and Ricardo Jardim-Goncalves. Challenging electronic procurement in the AEC sector: A BIM-based integrated perspective. *Automation in Construction*, 20(2):107–114, Mar. 2011.

[3] K. Czarnecki, T. Bednasch, P. Unger, and U. Eisenecker. Generative programming for embedded software: An industrial experience report. In D. Batory, C. Consel, and W. Taha, editors, *Generative Programming and Component Engineering*, volume 2487, pages 156–172. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[4] K. Czarnecki, S. Helsen, and U. Eisenecker. Formalizing cardinality-based feature models and their staged configuration. *University of Waterloo*, 2004.

[5] K. Czarnecki, S. Helsen, and U. Eisenecker. Staged configuration using feature models. In *Lecture notes in computer science*, volume 3154, page 266–283, 2004.

[6] Daniel Kullmann and Henrik W. Bindner. Using rules in high-level communication for the control of power systems. In *Proceedings of the 2nd International Conference on Microgeneration and Related Technologies*, 2011.

[7] P. Fernandes, C. Werner, and E. Teixeira. An approach for feature modeling of context-aware software product line. *Journal of Universal Computer Science*, 17(5):807–829, 2011.

[8] Z. Jaroucheh, X. Liu, and S. Smith. Mapping features to context information: Supporting context variability for context-aware pervasive applications. In *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2010 IEEE/WIC/ACM International Conference on*, volume 1, pages 611 –614, 2010.

[9] K. C. Kang, J. Lee, and P. Donohoe. Feature-oriented product line engineering. *IEEE software*, page 58–65, 2002.

[10] T. Possompès, C. Dony, M. Huchard, H. Rey, C. Tibermacine, and X. Vasques. Towards software product lines application in the context of a smart building project. *Proceedings of the 2nd International Workshop on Model-driven Product Line Engineering (MDPLE 2010)*, pages 73—84, 2010.

[11] C. Quinton, S. Mosser, C. Parra, and L. Duchien. Using multiple feature models to design applications for mobile phones. In *Proceedings of the 15th International Software Product Line Conference, Volume 2*, SPLC '11, page 23:1–23:8, New York, NY, USA, 2011. ACM.

[12] Raphael Romeikat, Bernhard Bauer, and Henning Sanneck. Modeling of domain-specific ECA policies. In *Proceedings of the 22nd International Conference on Software Engineering and Knowledge Engineering (SEKE 2011)*, pages 52—58, Miami Beach, USA, July 2011.

[13] P.-Y. Schobbens, P. Heymans, J.-C. Trigaux, and Y. Bontemps. Generic semantics of feature diagrams. *Computer Networks*, 51(2):456–479, Feb. 2007.

[14] Thibaut Possompès, Christophe Dony, Marianne Huchard, and Chouki Tibermacine. Design of a UML profile for feature diagrams and its tooling implementation. In *Software Engineering and Knowledge Engineering (SEKE 2011)*, pages 693—698, Miami, FL, USA, 2011.

[15] M. Voelter and I. Groher. Product line implementation using aspect-oriented and model-driven software development. In *Software Product Line Conference, 2007. SPLC 2007. 11th International*, pages 233–242. IEEE, 2007.

An ADM-based Method for migrating CMS-based Web applications

Feliu Trias
Kybele Research Group
C/Tulipán, s/n, 28933,
Móstoles, Madrid (Spain)
(+34) 91 488 93 59
feliu.trias@urjc.es

Valeria de Castro
Kybele Research Group
C/Tulipán, s/n, 28933,
Móstoles, Madrid (Spain)
(+34) 91 488 81 18
valeria.decastro@urjc.es

Marcos López-Sanz
Kybele Research Group
C/Tulipán, s/n, 28933,
Móstoles, Madrid (Spain)
(+34) 91 488 82 34
marcos.lopez@urjc.es

Esperanza Marcos
Kybele Research Group
C/Tulipán, s/n, 28933,
Móstoles, Madrid (Spain)
(+34) 91 664 74 91
esperanza.marcos@urjc.es

***Abstract*— In the last years many organizations have implemented their Web applications on Content Management Systems (CMSs) because of their efficiency for managing huge amount of digital content. Hence, these platforms have increased their presence in the market and currently we can find a vast number of different CMS platforms. In this context, it is very easy to find organizations which experiment the necessity to migrate their CMS-based Web applications to another CMS platform which meets better their needs. This software migration follows a complex and often very expensive, time-consuming and error-prone process of reengineering. Hence, in the last ten years many methods trying to minimize these disadvantages have appeared in the literature, but none of them is focused in the features of the CMS-based Web applications. To solve it, we present a method for migrating CMS-based Web applications to different CMS platforms in an automatic manner. This method follows the principles of the Architecture-Driven Modernization (ADM), the initiative proposed by the Object Management Group (OMG). To show the feasibility of our method we present a case study where we take a medium-size CMS-based Web application implemented in Drupal and we migrate part of its presentation aspect to Joomla!.**

Keywords-Content Management System; Web application; Architecture-Driven Modernization; Reverse Engineering and Model-driven Engineering.

I. INTRODUCTION

Over the last years, the number of Web applications developed by industry has increased dramatically. Furthermore, the volume of digital content has also grown rapidly. Therefore, organizations have experienced the need of using strong management platforms to maintain their large-scale Web applications and manage all their content in a robust and reliable manner [1]. One of the most popular adopted solutions has been the use of Content Management Systems (CMS) as platforms to base their Web applications [2]. These CMS-based Web Applications allow users to collect, manage and publish content in an integral way. This new kind of Web applications provides features that distinguish them from the traditional Web applications. According to [3] some of these features are: the dynamic creation of content; the explicit separation between content and design; and, the flexibility of the functionality extension.

CMS platforms are evolving continuously, so that we can find a large number of them in the market offering diverse facilities [4]. Organizations often need to migrate from their legacy CMS-based Web application to another target CMS platform because their current CMS platform has become obsolete and it does not meet their needs. Therefore, they find it necessary to start a software reengineering process.

The problem is that pretty often this reengineering process is carried out following an ad-hoc manner that entails expensive costs and high risks for the organization [5]. For this reason, the Object Management Group (OMG) proposes the Architecture-Driven Modernization (ADM) an initiative which advocates for the application of MDA (Model-Driven Architecture) [6] techniques and tools to formalize the software reengineering process. ADM provides several benefits such as reducing development and maintenance costs and extending the life cycle of the legacy systems [7].

After the performance of a literature review, we realized that there are software reengineering methods that try to standardize and formalize the reengineering process, but none of them is focused on the migration of CMS-based Web applications [8] in spite of the specific features of these Web applications and the clear increasing use of them in the market. Although, it is possible to find in the literature methods following the ADM principles, very few can be considered as full-fledged ADM methods.

To solve this gap, in this paper we present a complete ADM-based reengineering method for the migration of a CMS-based Web application to a different CMS platform. Our ADM-based method cope with the three classical reverse engineering phases following a "horseshoe" process [9]: (a) Reverse engineering phase, that consists of knowledge abstraction, (b) Restructuring phase, to modify the system taking into account the features of the target CMS platform. In this phase we define the CMS Model, one of the cornerstones of our method. This model conforms to the CMS Common Metamodel presented in [10]; and (c) Forward engineering phase, to address the generation of the new system containing new features.

To show the feasibility of our method, we apply our method over a medium-size Web application implemented in Drupal [11] with the intention of migrating part of its presentation aspect to Joomla! [10]. We focus in how to extract models from its PHP code and from its configuration files as well as to

migrate the structure of directories and other software artifacts to the target CMS platform.

The rest of this paper is organized as follows: Section II provides an explanation of ADM principles focusing on the standard metamodels. Section III explains the related works. Section IV presents the features of Common CMS Metamodel. Section V presents our ADM-based reengineering method. Section VI shows the feasibility of our method with the application to a medium-size case study and, finally, Section VII presents the conclusions and future works.

II. ARCHITECTURE-DRIVEN MODERNIZATION

The Object Management Group (OMG) proposes the Architecture-Driven Modernization (ADM) an initiative which advocates for the application of MDA (Model-Driven Architecture) [2] principles to formalize the software reengineering process. ADM provides several benefits such as reducing development and maintenance costs and extending the life cycle of the legacy systems [3]. ADM develops seven standard metamodels to represent the information involved in a software reengineering process, but only three of them are available: *Abstract Syntax Tree Metamodel* (ASTM) [4], *Knowledge Discovery Metamodel* (KDM) [5] and *Structured Metrics Metamodel* (SMM) [6]. These metamodels allow developers to save time and effort creating their own metamodels [7]. In this paper we focus on models which conform to ASTM (ASTM models) and KDM (KDM models).

ASTM [13], is the metamodel which represents a low-level view of the system. It allows the representation of the source code's syntax of a software system. To obtain an ASTM model it is necessary to define text-to-model (T2M) transformations that let the mapping between the code's grammar and the ASTM models. These models reduce the complexity to generate models at higher abstract level, such as KDM models.

KDM [14] lets you represent semantic information about a software system. It provides a common interchange format intended to represent existing software assets, allowing tool interoperability at PIM level. KDM comprises several packages (e.g. core, kdm, source, code or action) to improve modularity and separation of concerns. KDM models are generated by using model-to-model (M2M) transformations that extract information from ASTM models.

III. RELATED WORKS

After performing a literature review, we found a set of model-driven methods related to the reengineering of Web applications.

In [24] is presented a method to reverse engineer Web applications in order to extract their conceptual models by using WebML notation. This method extracts models from ASP.NET web pages. In [25] we find a model-driven reengineering method based on the Object-Oriented Hypermedia Design Method (OOHDM) to personalize existent Web applications. This method is applied over Web applications developed over Web application frameworks based on MVC pattern (such as struts, or blueprints: Java, JSP and XML). In [26] is presented a model-driven method allowing the migration between a traditional Web application and a Rich Internet application. This method is applied over Struts-based Web applications. [27] is a model-driven method to reverse engineer user interfaces. It generates models implemented by User Interface Description Language (UsiXML). In [28] is proposed a model-driven semi-automatic redesign method to improve the design of existing Web applications. The method analyzes the client side HTML pages to recover its conceptual model according to the Ubiquitous Web Applications design methodology (UWA). This method allows the migration of Web applications to a MVC-pattern. Finally, [29] proposes a model-driven engineering method to perform reverse engineering of Rapid Application Development (RAD)-built GUIs, which is focused on discovering the implicit layout, and produces a GUI model where the layout is explicit. This method allows extracting models from Oracle Forms and Borland Delphi code.

With this review we realized that ADM principles are embodied in the existing reverse engineering methods since they use models at different abstraction level (mainly at PSM and PIM levels) and try to automate their processes using T2M and M2M transformations, but they scarcely use the standards metamodels proposed by ADM (ASTM and KDM). Just [26] considers models at KDM level. Finally, it is worth noting that [26] uses MoDisco to generate KDM models and [29] implements T2M transformations using Gra2MoL.

IV. CMS COMMON METAMODEL

As it is said in the introduction, our method is focused on CMS-based Web applications which provide with a set of specific features that differ from traditional Web applications according to [3]: (1) *the dynamic creation of content*, content is created and added dynamically by non-technical users of the Web, without requiring the intervention of the webmaster, (2) *separation between content and design*, the page graphical design is stored in a template and the content is stored in a database or a separate document. and (3) *functionality extension*, is in a CMS-based Web application among their most valuable features, achieved through module addition. This ensures quick content deployment and provides means for flexible extension, besides promoting efficiency and reducing development costs. In [10] we present the CMS Common Metamodel which defines the key concepts for modeling CMS-based Web applications. Therefore, the CMS Common Metamodel captures elements such as, theme, vocabulary or module, and other specific elements of the CMS domain. These elements are classified in five different concerns: (1) *navigation*, which considers the elements that define the navigational structure of the Web application, (2) *presentation*, which defines the structure and look-and-feel of the Web pages, (3) *content*, that captures the data and data type of the information managed by the CMS-based web application, (4) *user*, defines the elements related to roles of the users and their permissions and (5) *CMS behavior*, that contains the elements that allow the definition of the different functions performed by the Web application. For lack of space, we present in Figure 1 just an excerpt of the presentation concern.

At the restructuring phase of our ADM-based method we define the CMS Model that conforms to the CMS Common Metamodel. This CMS Model allows developers to represent

the knowledge extracted from the CMS-based Web application in the CMS domain and it lets the refinement and restructuring of this knowledge to a target CMS platform.

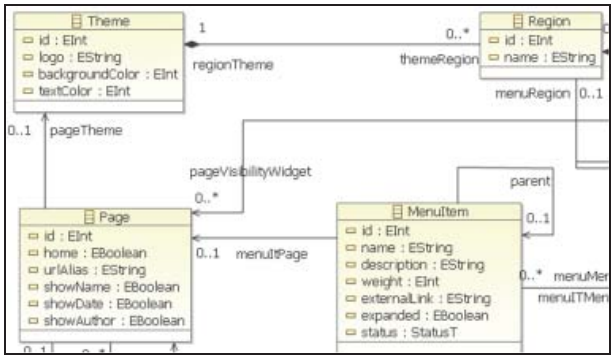


Figure 1. CMS Common Metamodel excerpt

V. ADM-BASED METHOD

The method proposed in this paper is defined as a "horseshoe" process of reengineering composed of a set of steps gathered in the three classical reverse engineering phases as is shown in Figure 2: (1) reverse engineering phase, (2) restructuring phase; and (3) forward engineering phase. In this section, we present these set of steps:

A. *Reverse engineering phase*

This phase is composed of three steps: (1) knowledge extraction, (2) the automatic generation of the KDM models and (3) the automatic obtainment of the CMS Model.

1) Knowledge extraction: In this phase, we define the T2M transformations which allow us to map the code to an ASTM model. To perform these transformations we use Gra2MoL (Grammar to Model Language) [15] and the Xtext framework [16]. Gra2MoL is a rule-based Domain Specific Language (DSL) [17] like ATL [18] that integrates a query language specially tailored to construct models from source code represented as a parse tree. We use Gra2MoL to extract an ASTM model from the configuration files of the CMS-based Web application. Otherwise, we use the Xtext framework to define the code's grammar, such as PHP grammar, from which we obtain a tooling (a metamodel, a parser and an editor) that allows us to implement in Java a model extractor to extract ASTM models from the source code.

2) Automatic generation of KDM models: Mainly, the KDM models are generated from the ASTM models by means of M2M transformations. In our method these transformations are implemented in ATL. The two KDM models that we consider are: the Code Model and the Inventory Model. The Code Model represents the software system's source code at PIM level. The Code Model is composed of elements from the code and action packages of KDM so that it does not contain elements representing specific statements or expressions of a particular programming language. Otherwise, the Inventory Model is a catalog of the system's software artifacts (e.g. source files, images, configuration files and directories) that allow us to represent the architecture of the CMS-based Web application. This model is composed of the elements from the source package of KDM. We use the tool MoDisco [20] to obtain automatically this model. Modisco is an Eclipse plug-in for model-driven reverse engineering which lets generate the Inventory Model directly from the source code.

3) The automatic obtainment of the CMS Model: In this step we extract the knowledge captured in the Code Model and Inventory Model and we generate automatically the CMS Model, conforms to the CMS Common Metamodel. This CMS Model allows us to represent the knowledge within the CMS domain. To obtain automatically this model we define a set of M2M transformations implemented in ATL.

B. *Restructuring phase*

The CMS Model can be manually restructured by the developer taking into account the features of the target CMS platform. As it is shown in Figure 2, we obtain a restructured CMS Model. This CMS model conforms to the CMS Common Metamodel which provides with the required and common elements of the CMS domain.

C. *Forward engineering phase*

In this phase the top-down development process starts. We can highlight three main steps: (1) the automatic generation of the target KDM models, (2) the automatic obtainment of the target ASTM model and (3) code generation. The M2M transformations performed within this phase are implemented in ATL. For clarity reasons, from now on, the models inherited from the reverse engineering phase will be called as *legacy* ASTM models (*legacy* Code Model and *legacy* Inventory Model). On the other hand, the models obtained in this phase will be called as *target* ASTM models (*target* Code Model and *target* Inventory Model).

1) The automatic generation of the target KDM models: From the CMS Model we generate automatically, by M2M transformations, the *target* Code Model and the *target* Inventory Model that represents the new implementation of the CMS-based Web application. Moreover, it is necessary to define horizontal M2M transformations at KDM level. These transformations allow us to obtain extra knowledge to generate correctly these *target* KDM models. As we can see in Figure 2, M2M transformations are defined between the *legacy* KDM models and the *target* KDM models.

2) The automatic obtainment of the target ASTM model: In this task we generate automatically the *target* ASTM model from the *target* Code Model and the *target* Inventory Model by means of M2M transformations. As we can see in Figure 2, to generate this *target* ASTM model is also necessary to obtain extra knowledge from the *legacy* ASTM model at ASTM level by means of horizontal M2M transformations.

3) Code generation: In this step we generate the software artifacts (e.g. source files, images, configuration files and directories) that compose the architecture of the target CMS-

based Web application. These artifacts are generated from the *target* Inventory Model and the code that implements them are obtained automatically from the *target* ASTM model. This automatic generation of artifacts and code is carried out by model-to-text (M2T) transformations implemented by MOFScript [21] as is shown in Figure 2.

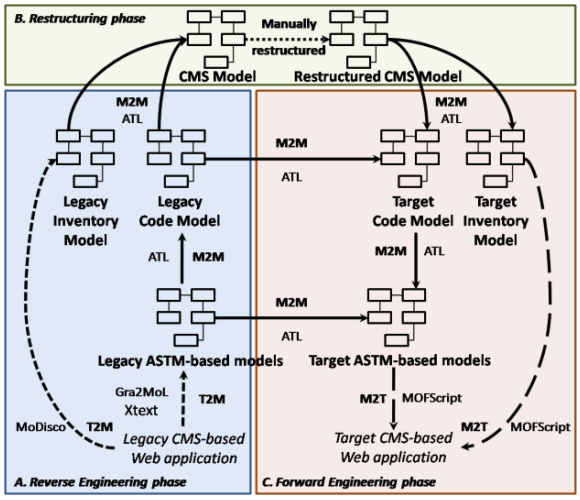


Figure 2. ADM-based method

VI. CASE STUDY

In order to illustrate our ADM-based method, we consider a medium-size CMS-based Web application about health and wellbeing called Websana that allows us to show the feasibility of our method. It is implemented in Drupal and our intention is to migrate part of the presentation aspect of the Web application from Drupal to Joomla!. We have selected these two CMS platforms because they are two of the most used open-source CMS platforms in the market [4].

In Drupal the presentation aspect is represented by *themes* which are directories located in the following path drupal_project_name\site\all\themes within the Drupal project. A *theme* is composed of a set of files and directories that determines the structure and look-and-feel of the Web pages. Two types of files composing a Drupal *theme* are: *.info* files (which are *configuration files* defining the meta-information) and *php* files (*template.php*).

On the other hand, the *themes* in Joomla! (in Joomla! domain are called *templates*) are also represented by directories located in the path joomla_project_name\templates. A *theme* in Joomla! also provides with configuration files, called *templateDetails.xml* and *php* files.

For lack of space, we focus this case study in the following steps of our ADM-based method: (a) knowledge extraction, (b) the automatic generation of the KDM models, (c) the automatic obtainment of the CMS Model, (d) the automatic generation of the target KDM models and (e) code generation.

A. Knowledge extraction

The main goal of this step is to extract the ASTM models from the code of the *.info* files and the *php* files that implements a *theme*. Firstly, to extract the ASTM model from the *.info* files, we implemented a set of T2M transformations by using Gra2MoL. Our first task was to define the grammar that specifies the syntax of the code of the *.info* files using the language ANTLR [22]. This code is composed of sentences with the following structure: *variable* = *value*. Figure 3 shows the transformation rule implemented in Gra2MoL that transforms a set of statements of the *.info* file of the danland *theme* into *VariableDefinition* elements of the ASTM model.

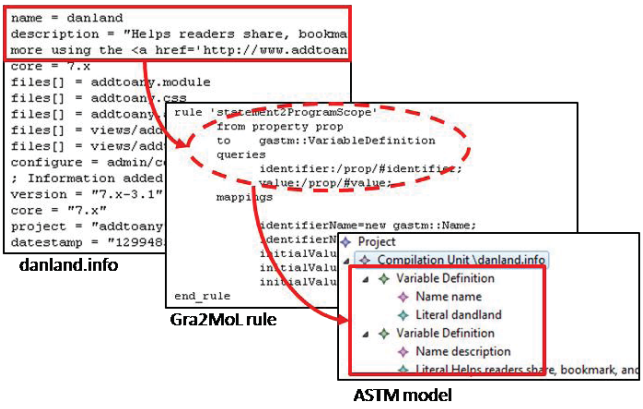


Figure 3. Gra2MoL transformation rule

Secondly, to extract the ASTM model from the *php* code we implemented a model extractor in Java. To implement this model extractor we carried out three activities: 1) definition of the PHP grammar, 2) mapping of PHP grammar elements to elements of ASTM, and 3) implementation of the model extractor. For the definition of the PHP grammar we used the Xtext framework. Using this framework we obtained automatically three artifacts: 1) a metamodel, 2) a textual editor and 3) a PHP parser that allow us to recognize the elements of the PHP grammar from code written in *php*.

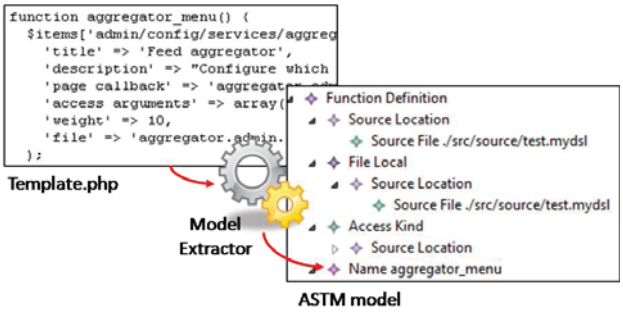


Figure 4. Parser to extract the ASTM model from PHP code

The PHP parser facilitated us the implementation of the model extractor. In the second activity we defined the mappings between the elements of the PHP grammar and the elements of ASTM. At the time of defining these mappings we realized that some elements from the PHP grammar cannot be mapped to elements of ASTM. For that reason, we extended the ASTM with the specific elements of the *php* code (ASTM_PHP). Finally, in the third activity we implemented in Java the model extractor to obtain ASTM models from PHP code. For its implementation we use: 1) the PHP parser

obtained in the first activity and 2) the API in Java obtained automatically from ASTM by using the Eclipse Modeling Framework (EMF) [22], to generate the elements of the ASTM models. Figure 4 shows the transformation of a function definition contained in the *template.php* file (of the danland *theme*) into a *FunctionDefinition* element in the ASTM model.

B. *The automatic generation of the KDM models*

For lack of space, we only show the extraction of the *legacy* Inventory Model by using MoDisco. As it is said previously, this model represents the architecture and software artifacts of the CMS-based Web application. It is composed of *AbstractInventory* elements, e.g. *BinaryFile*, *ExecutableFile*, and *AbstractInventory* relationships. It is worth noting that all the *AbstractInventory* elements are provided with an attribute called *path* which contains the location of the element in the project's structure which is necessary to find out the directories which represent *themes*.

Figure 5.a) shows an extract of the directory hierarchy of Websana. In this case, we see the following *themes*: danland, fusion, marinelli and tao. Figure 5.b) shows an excerpt to indicate the correspondence with the *legacy* Inventory Model.

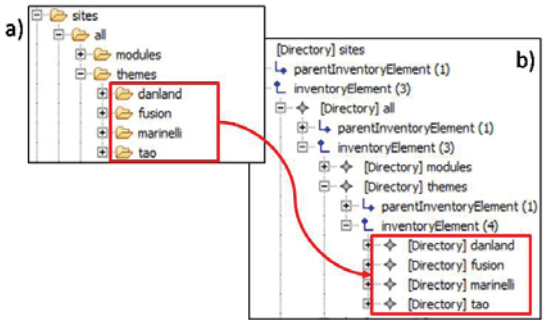


Figure 5. Legacy Inventory Model

C. *The automatic obtainment of the CMS Model.*

The next step was to generate automatically the CMS Model from the *legacy* Inventory Model. The CMS Model represents the extracted knowledge within the CMS domain and allows developers the refinement of the CMS-based Web application according to the features of the target CMS platform. In this case study we select the directories located in drupal_project_name\site\all\themes which represent *themes* and we represent them as *theme* elements in the CMS Model. To do this, we define M2M transformations implemented in ATL. Figure 6 shows the CMS Model by using a hierarchical editor where we can observe the four *themes* generated from the *legacy* Inventory Model.

Figure 6. CMS Model

D. *The automatic generation of the target KDM models.*

In this step we obtained automatically the *target* Inventory Model from the CMS Model. The *target* Inventory Model was generated considering the architecture and the software artifacts of the target CMS platform. We implemented in ATL a set of M2M transformations to generate this Inventory Model. Figure 7.a) shows an excerpt of the CMS Model and Figure 7.b) is an extract of the *target* Inventory Model over Joomla!.

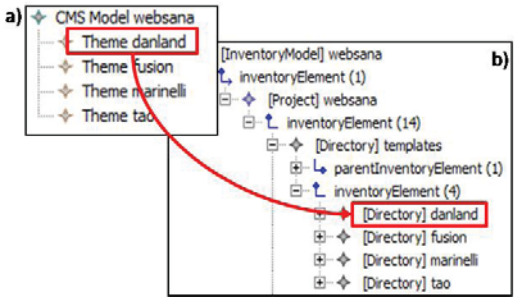


Figure 7. Target Inventory Model

E. *Code generation.*

In this step we generated the architecture and the software artifacts implementing the target CMS-based Web application from the *target* Inventory Model. To do that, we defined M2T transformations implemented in MOFScript. Figure 8 shows a M2T transformation rule that generates the templateDetails.xml file, one of the configuration files of a *theme* in Joomla!.

```
textmodule NewTransformation (in kdm:"http://www.eclipse.org/MoDisco/kdm") {

      kdm.Directory::main () {
        self.inventoryElement->forEach(sf:kdm.SourceFile){
            if (sf.name="templateDetails.xml"){
                  file (self.name+ "_" +sf.name.tolower)
```

Figure 8. MOFScript transformation

VII. CONCLUSIONS

The use of CMS-based Web applications has grown in the last decade because organizations have experienced the necessity of using strong management tools to maintain their large-scale Web applications and manage all their huge amount of digital content in a reliable manner. Besides, these organizations usually carry out migration processes to other CMS platforms which meet better their needs. The problem is that this migration processes are performed following an ad-hoc manner that entails expensive costs and high risks for the organization. In the literature, we find methods that address the formalization of this migration process to reduce costs and risks, but none of them is focused on the specific features of the CMS-based Web applications.

To solve this gap, we present a method for the migration of a Web application from a CMS platform to another one. Our method is based on the ADM principles since it provides powerful techniques to extract knowledge from legacy systems and to automate the migrating process. This method cope with the three classical reverse engineering phases following a "horseshoe" process: (a) Reverse engineering phase, (b) Restructuring phase and (c) Forward engineering phase.

In the reverse engineering phase we obtain the *legacy* ASTM models and *legacy* KDM models. To extract the ASTM models from the source code we define a set of rules in

Gra2MoL and we implemente a model extractor in Java. As we present in the case study, we use Gra2MoL language to extract the ASTM model from the code of the configuration files (*.info files*) and we use the model extractor to extract the ASTM model from *php* files. We have proven the effectiveness of Gra2MoL and its ease of use. The only disadvantage is the lack of documentation. Otherwise, the PHP parser obtained by Xtext has facilitated us the implementation of the model extractor. The two KDM models we use in our method are: Code Model and Inventory Model. To obtain the Code Model we define M2M transformations (implemented in ATL) from the ASTM models and to generate the Inventory Model we use MoDisco.

In the restructuring phase, we propose the use of the CMS Model, which conforms to the CMS Common Metamodel, which allows developers to adapt a CMS-based Web application to a target CMS platform.

During the forward engineering phase, we have required to define horizontal M2M transformations at KDM and ASTM levels to obtain correct models. To generate code we use MOFScript.

As future work we can consider the complete implementation of our method to achieve the migration of all the elements that compose a CMS-based Web application from one CMS platform to another one.

ACKNOWLEDGMENT

This research has been partially funded by the Project MASAI (TIN-2011-22617) from the Spanish Ministry of Science and Innovation.

REFERENCES

[1] S. McKeever, "Understanding Web content management systems: evolution, lifecycle and market," Industrial Management & Data Systems, vol. 103, no. 9, pp. 686–692, 2003.

[2] B. Boiko, "Understanding Content Management," Bulletin of the American Society for Information Science and Technology, vol. 28, no. 1, pp. 8–13, 2001.

[3] R. Vidgen, S. Goodwin, and S. Barnes, "Web Content Management," in 14th Bled Electronic Commerce Conference, 2001.

[4] R. Shreves, "Open Source CMS Market Share," Bali, Indonesia, 2011.

[5] H. M. Sneed and A. Gmbh, "Estimating the Costs of a Reengineering Project," in 12th Working Conference on Reverse Engineering, 2005.

[6] S. J. Mellor, K. Scott, A. Uhl, and D. Weise, "Model-Driven Architecture," in Advances in Object-Oriented Information Systems, 2002, Lecture No., vol. 2426, pp. 233–239.

[7] R. Pérez-castillo, I. García, R. De Guzmán, D. Caivano, and M. Piattini, "Database Schema Elicitation to Modernize Relational Databases," in 14th International Conference on Enterprise Information Systems, 2012, pp. 126–132.

[8] F. Trias, V. De Castro, M. López-sanz, and E. Marcos, "A Systematic Literature Review on CMS-based Web Applications," in ICSOFT, 2013.

[9] E. J. Chikofsky and J. H. Cross, "Reverse Engineering and Design Recovery: A Taxonomy," IEEE Software, vol. 7, no. 1, pp. 13–17.

[10] F. Trias, "Building CMS-based Web Applications Using a Model-driven Approach," in Sixth International Conference on Research Challenges in Information Science (RCIS), 2012, pp. 1 – 6.

[11] "Drupal CMS." [Online]. Available: http://drupal.org/.

[12] "Joomla! CMS." [Online]. Available: http://www.joomla.org/.

[13] "Abstract Syntax Tree Metamodel specification of the OMG." [Online]. Available: http://www.omg.org/spec/ASTM/1.0.

[14] R. Pérez-Castillo, I. G.-R. de Guzmán, and M. Piattini, "Knowledge Discovery Metamodel-ISO/IEC 19506: A standard to modernize legacy systems," Computer Standards & Interfaces, vol. 33, no. 6, pp. 519–532, Nov. 2011.

[15] J. L. Cánovas Izquierdo, J. Sánchez Cuadrado, and J. García Molina, "Gra2MoL□: A domain specific transformation language for bridging grammarware to modelware in software modernization," in MODSE, 2008, pp. 1–8.

[16] E. Moritz and H. Behrens, "Xtext - Implement your Language Faster than the Quick and Dirty way Tutorial Summary," in ACM international conference companion on Object oriented programming systems languages and applications companion, 2010, pp. 307–309.

[17] A. van Deursen and P. Klint, "Domain-Specific Language Design Requires Feature Descriptions," Journal of Computing and Information Technology, vol. 10, no. 1, pp. 1–17, 2002.

[18] "Atlas Transformation Language." [Online]. Available: http://www.eclipse.org/atl/.

[19] R. Pérez-Castillo, I. G.-R. de Guzmán, and M. Piattini, "Knowledge Discovery Metamodel-ISO/IEC 19506: A standard to modernize legacy systems," Computer Standards & Interfaces, vol. 33, no. 6, pp. 519–532, Nov. 2011.

[20] H. Bruneliere, J. Cabot, F. Jouault, and F. Madiot, "MoDisco□ : A Generic And Extensible Framework For Model Driven Reverse Engineering," in International conference on Automated software engineering - ASE '10, 2010, pp. 173–174.

[21] A.-J. Oldevik, J., Neple, T., Grønmo, R., Aagedal, J., Berre, "Toward Standardised Model to Text Transformations," in Model Driven Architecture – Foundations and Applications, 2005, vol. 3748, pp. 239–253.

[22] T. J. Parr and R. W. Quong, "ANTLR: A Predicated-LL(k) Parser Generator," Practice and Experience, vol. 25, no. 7, pp. 789–810, 1995.

[23] F. Budinsky, "Eclipse Modeling Framework," vol. 2nd Editio. Addison-Wesley Professional, 2008.

[24] T. Katsimpa, Y. Panagis, E. Sakkopoulos, G. Tzimas, and A. Tsakalidis, "Application modeling using reverse engineering techniques," in Proceedings of the 2006 ACM symposium on Applied computing - SAC '06, 2006, pp. 1250–1255.

[25] A. Martin, "A Model-Driven Reengineering Approach to Web Site Personalization," in Third Latin American Web Congress, 2005.

[26] R. Rodríguez-Echeverría, P. J. Clemente, J. C. Preciado, and F. Sánchez-Figueroa, "Modernization of Legacy Web Applications into Rich Internet Applications," in International Conference on Web Engineering, 2011, pp. 236–250.

[27] L. Bouillon, Q. Limbourg, J. Vanderdonckt, and B. Michotte, "Reverse Engineering of Web Pages based on Derivations and Transformations," in Third Latin American Web Congress, 2005.

[28] M. L. Bernardi, G. A. Di Lucca, and D. Distante, "Improving the Design of Existing Web Applications," in Seventh International Conference on the Quality of Information and Communications Technology, 2010, pp. 499–504.

[29] Ó. Sánchez Ramón, J. Sánchez Cuadrado, and J. García Molina, "Model-driven reverse engineering of legacy graphical user interfaces," in International conference on Automated software engineering - ASE '10, 2010, pp. 147–150.

BeMoRe: a Repository for Handling Models Behaviors

Youness Bazhar
LIAS/ISAE-ENSMA
Futuroscope, FRANCE
Email: bazhary@ensma.fr

Yamine Aït-Ameur
IRIT/INP-ENSEEIHT
Toulouse, FRANCE
Email: yamine@enseeiht.fr

Stéphane Jean
LIAS/University of Poitiers
Futuroscope, FRANCE
Email: jean@ensma.fr

***Abstract*—With the increasing size of models and their instances, the management of models in databases becomes a necessity. Persistent Model Management Systems (PMMS) aim at providing a persistent environment for the management of instances, models and metamodels. They consist of (1) a database that stores metamodels, models and their instances, and (2) an associated exploitation language for manipulating these different abstraction layers. Several PMMS have been proposed in the literature but they currently mostly focus on the structural definition of models and metamodels in terms of (meta-)classes and (meta-)attributes. The behavioral semantics that consists of associating operations to models and metamodels elements is currently mostly not supported or only partially supported (by a set of predefined hard coded operations or by imposing a single programming language). In this paper, we propose an extension of PMMS to support the definition of behavioral semantics of models and metamodels using a wide range of programming possibilities. Our approach consists of introducing dynamically user-defined operations that can have multiple and heterogeneous implementations (e.g., external programs or web services). As a consequence, this extension enhances PMMS giving them more coverage and further flexibility. Our proposal has been implemented in a PMMS called *BeMoRe* and several experiments have been run to analyze the scalability of this PMMS.**

***Keywords*—model management; meta-modeling; database**

I. Introduction

Models are widely used in software engineering to design software components such as database schemes or user interfaces. This involves operations on models such as code generation, transformation, archiving, versioning, etc. Following the vision of Bernstein [1], several *model management systems* (*MMS*) have been set up during the last decade to manage instances, models and metamodels and support operations on them (e.g., [2], [3], [4], [5]).

With the increasing size of data instances and models in several domains (e.g., in genomics, the Uniprot dataset, www.uniprot.org, gathers more than 200GB of protein sequence resources), the possibility of managing large scale models and instances in MMS has raised a lot of interest. Two main approaches have been followed to increase the scalability of MMS. The first approach consists of connecting a MMS to a database called *model repository* (e.g., EMFStore [6], TERESA model repository [7]). This approach uses a loose coupling between the MMS and the database and has two main drawbacks: (1) most model management operations require loading the whole model and instances in main memory and (2) the database exploitation language does not support the definition, manipulation and querying of models and metamodels (it only supports basic SQL operations). To address these problems, a second approach, followed in our work, has been developed. It consists of extending databases for the management of models and metamodels (e.g., [2]). These systems called *Persistent Model Management Systems* (*PMMS*) are composed of (1) a database that stores metamodels, models and instances and (2) an exploitation language for manipulating models and metamodels.

If PMMS solve the two drawbacks of the loose coupling of a MMS with a database, they currently do not support the same flexibility concerning the definition of behavioral semantics (procedural aspects) of models and metamodels. Indeed PMMS focus mainly on the definition of the structure of metamodels and models but provide a limited support for the definition of operations on models and metamodels. For example, some PMMS provide hard-encoded operators for model management (e.g., Match, Merge, Union [8], [1]), or only give access to the database procedural languages (e.g., PL/SQL) that do not support the manipulation of models and metamodels (they only manipulate relational tables). The most advanced PMMS concerning the definition of metamodels and models behaviors is ConceptBase [2]. Using this PMMS user-defined operations can be defined on models and metamodels as deductive rules implemented with a specific language (PROLOG). However this PMMS lacks the possibility to integrate operations that have already been implemented using a given programming language or provided as an external web service.

In this paper we propose an extension of PMMS to support the definition of behavioral semantics of models and metamodels using a wide range of programming capabilities. This extension has been motivated in a previous paper [9] by presenting a complete state of the art, and applied in the specific context of ontology-based databases in [10]. In this paper we make the following new contributions:

- definition of a set of requirements for a complete PMMS;
- definition of a PMMS including the behavioral aspect;
- implementation of our approach: the *BeMoRe* PMMS;
- first experiments to study the scalability of BeMore.

The remainder of this paper is organized as follows. Section II presents a set of requirements for a complete PMMS justified on a motivating example. Section III gives an overview of the state of the art by analyzing existing PMMS using our requirements. Section IV presents the formal definition of a PMMS handling behavioral semantics of models and metamodels. Section V overviews the implementation of

our approach and Section VI shows the experiments done to study its scalability. Finally, section VII concludes this paper and discusses ongoing works.

II. Requirements for a Complete PMMS

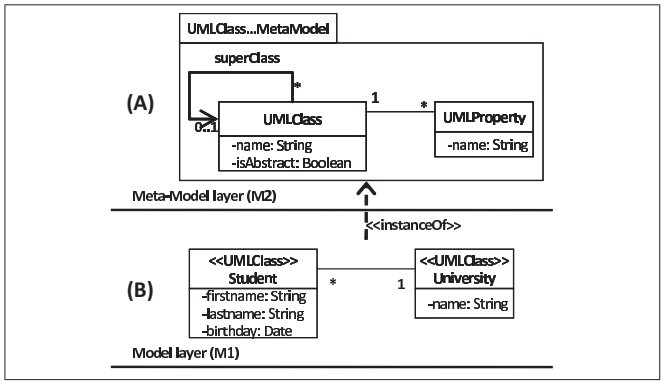


Fig. 1: A motivating example

Figure 1 presents a simple class diagram metamodel (A) and a model (B) conforming to that metamodel. Using this example, we are capable to define the set of requirements identified for a complete PMMS. Due to space limitation we use a very simple example. The interested reader may refer to [9] and [10] for more complex and real motivating examples.

Requirement 1 (*extensible metamodel layer*)

> *PMMS shall offer an extensible metamodel layer so that multiple modeling formalisms can be defined.*

Justification: in our example, we only have the UML class diagram metamodel defined at the metamodel layer. However sofware engineering uses a lot of different models (e.g., entity-relationship, functional, state transition models).

Requirement 2 (*structural and descriptive semantics*)

> *PMMS shall support the definition of structural and descriptive semantics of metamodels and models elements. For instance, the PMMS shall provide constructors of classes, attributes, inheritance and association relationships for defining models and metamodels.*

Justification: following the MOF specification, most models and metamodels (such as the ones of our example) can be expressed with object-oriented constructors.

Requirement 3 (*behavioral semantics*)

> *PMMS shall support introducing operations (functions, procedures) on metamodels and models elements.*

Justification: operations on models and metamodels elements are important to accomplish advanced model management tasks such as model transformation, code generation or constraints checking. For instance, in our example an operation could be defined to export the UML models in XML, or to compute the age of a student.

Requirement 4 (*flexible programming environment*)

> *PMMS shall provide an heterogeneous programming environment to implement operations. Particularly, it shall be able to use external programs written in any language (e.g., Java, C++) and remote services.*

Justification: as it is better to reuse existing pieces of software instead of rewriting them, a PMMS should be able to integrate existing implementations of operations whatever is the programming language used. For example, it is easy to find an existing code that exports an UML model in XML. So a PMMS should allow users to reuse this piece of software to implement an operation that exports UML models.

Requirement 5 (*hot-plug of implementations*)

> *PMMS shall support an immediate usage of the implementations for an operation without restarting the system (warm start).*

Justification: restarting a database system must be avoided for high availability applications. Thus the definition of an implementation of an operation, even if it is a web service, should not require restarting the PMMS (warm start).

III. Related Work

Several PMMS have been proposed. This section analyzes the most relevant PMMS according to our requirements.

ConceptBase [2] is a PMMS based on an object-oriented and deductive database. It is based on the Telos language that supports the definition of multiple abstraction layers with a set of constraints, rules and queries using meta-formulas. Furthermore, ConceptBase provides a set of predefined operators to manipulate simple and complex data types, and gives the possibility to introduce user-defined functions with membership constraints and external implementations. Yet, these implementation can only be done in the Prolog language. Besides external programs have to be stored in a special and internal file system, and requires restarting the server (cold start) in order to support the function newly introduced [11].

GeRoMe [3] is an extension of ConceptBase to define new operators from other ones by combining existing operators. However, this extension does not introduce a more flexible programming environment for these operations.

Rondo [4] is a PMMS that has a fixed and non extensible metamodel layer. It provides conceptual structures to define models and specify the behavioral semantics by providing a set of *primitive* high-level operators for model management and model mappings such as *Match*, *Delete* or *Extract*. Moreover, Rondo supports the definition of derived operators by composing basic and other defined operators.

Clio [5] is a PMMS defined for facilitating the tasks of heterogeneous data transformation and integration. These tasks are facilitated by mapping a source schema to a target schema with SQL statements.

DB-MAIN [12] is a PMMS designed for the management of database evolution. It is based on a fixed hard-encoded metamodel and offers a set of built-in high-level operators

for modifying the database structure and contents when an evolution is required.

OntoDB/OntoQL [13] is a PMMS initially defined for the management of ontologies and ontology models. It includes the OntoDB model repository and the OntoQL meta-modeling language. This PMMS is based on a fixed metametamodel to define and modify metamodels. Concerning the behavioral semantics, OntoDB/OntoQL uses only the PgPL/SQL procedural language of its back-end database management system (PostgreSQL). This language cannot manipulate complex types (e.g., meta-classes or classes) and consequently cannot define high-level operators.

As the previous overview of the state of the art shows, each existing PMMS presents some strengths and some limitations for the definition of structural and behavioral semantics of models and metamodels. The identified limitations are presented in Table I. Hence next section introduces the formal definition of an extension of PMMS to fulfill these requirements.

TABLE I: Synthesis of the state of the art

	Req. 1	Req. 2	Req. 3	Req. 4	Req. 5
ConceptBase	Yes	Yes	Yes	restricted	No
GeRoMe	Yes	Yes	Yes	restricted	No
Rondo	No	Yes	hard-coded	No	No
Clio	No	No	restricted	No	No
DB-MAIN	No	Yes	restricted	No	No
OntoDB/OntoQL	Yes	Yes	restricted	No	No

IV. Proposed Extension of PMMS

A PMMS is composed of a database data model and an exploitation language. The proposed extension of these two parts of a PMMS are presented in Subsection IV-A and Subsection IV-B.

A. Proposed Extension of PMMS Data Model

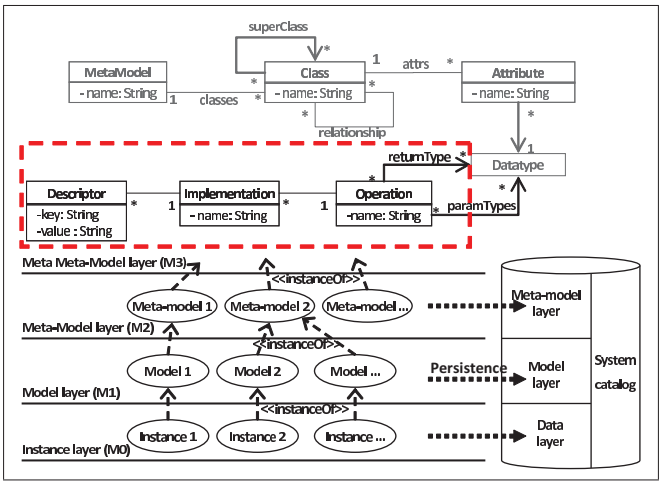


Fig. 2: The proposed data model for PMMS

Figure 2 gives an overview of the extended data model of PMMS that we propose. For conciseness we only detail the metametamodel layer but a similar extension has been be done at the metamodel level to be able to define operations at the different abstraction layers. Our model includes a set of classes that are described by attributes, and single class inheritance relationships are allowed. This part of our data model is usually available in all PMMS that, as we have seen in the previous section, focuses mainly on the structural and descriptive semantics of metamodels elements. The dashed area gives an overview of our proposed extension for the definition of behavioral semantics of metamodels elements. Our model supports the definition of operations with a list of input and an output. Furthermore, an operation can be associated to multiple implementations. Each implementation is itself described by a set of descriptors (couples of *key*, *value*). With these generic set of descriptors a new programming environment can be easily integrated in our approach.

This extension of PMMS can be formally defined by the following sets: MM, CL, ATT, DT, OP, IMP, $DESC$ that represent respectively sets of metamodels, classes, attributes, data types, operations, implementations and implementation descriptors. Table II gives the definition of these sets as well as a subset of constraints concerning these sets.

TABLE II: PMMS formal model

A metamodel is described by a set of classes:
$classes : MM \rightarrow P(CL)$
$\forall mm_i \in MM \Rightarrow \exists! E \in P(CL)/classes(mm_i) = E$
$\forall (mm_i, mm_j) \in MM/i \neq j \Rightarrow classes(mm_i) \cap classes(mm_j) = \emptyset$
A class may have a super class:
$superClass : CL \rightarrow CL$
A class may have inherited attributes from its super class:
$inheritedAttributes : CL \rightarrow P(ATT)$
$\forall cl_i \in CL \Rightarrow \exists! E \in P(ATT)/$
$\quad inheritedAttributes(cl_i) = attributes(superClass(cl_i))$
A class may be described by additional attributes:
$definedAttributes : CL \rightarrow P(ATT)$
$\forall cl_i \in CL \Rightarrow \exists! E \in P(ATT)/definedAttributes(cl_i) = E$
$\forall cl_i \in CL \Rightarrow$
$\quad definedAttributes(cl_i) \cap inheritedAttributes(cl_i) = \emptyset$
$\forall (cl_i, cl_j) \in CL/i \neq j \Rightarrow$
$\quad definedAttributes(cl_i) \cap definedAttributes(cl_j) = \emptyset$
The set of attributes of a class:
$\forall cl_i \in CL \Rightarrow attributes(cl_i) =$
$\quad inheritedAttributes(cl_i) \cup definedAttributes(cl_i)$
An attribute has a data type:
$typeOf : ATT \rightarrow DT$
$\forall att_i \in ATT \Rightarrow \exists! dt_j \in DT/typeOf(att_i) = dt_j$
An operation parameter has an order:
$input : OP \text{ x } \mathbb{N}^+ \rightarrow DT$
$\forall op_i \in OP \Rightarrow input(op_i) \in DT$
An operation can return a result:
$output : OP \rightarrow DT \cup \emptyset$
An operation can have several implementations:
$implementations : OP \rightarrow P(IMP)$
$\forall op_i \in OP \Rightarrow \exists! E \in P(IMP)/implementations(op_i) = E$
$\forall (op_i, op_j) \in OP/i \neq j \Rightarrow$
$\quad implementations(op_i) \cap implementations(op_j) = \emptyset$
An implementation is described by a set of descriptors:
$descriptors : IMP \rightarrow P(DESC)$
$\forall imp_i \in IMP \Rightarrow \exists! E \in P(DESC)/descriptors(imp_i) = E$
$\forall (imp_i, imp_j) \in IMP/i \neq j \Rightarrow$
$\quad descriptors(imp_i) \cap descriptors(imp_j) = \emptyset$

B. Proposed Extension of the PMMS Metamodeling Language

The exploitation language of a PMMS is composed of a model and a metamodel definition, manipulation and query language. Table III presents a subset of basic actions that a PMMS definition language should fulfill. These actions are defined by a signature (*SIG*), a precondition (*PRC*) and a

postcondition (*POC*). We only present in this table the *create* operations but the other operations (*alter*, *update* and *delete*) have been defined as well.

TABLE III: Main actions of the PMMS definition language

creation of a metamodel: SIG: $addMetaModel(mm) = MM'$ PRC: $mm \notin MM$ POC: $MM' = MM \cup \{mm\}$
creation of a class: SIG: $addClass : MM$ x $C \rightarrow MM$ PRC: $cl \notin CL$ POC: $classes(mm) = classes(mm) \cup \{cl\}$
creation of an attribute: SIG: $addAttribute : CL$ x $ATT \rightarrow CL$ PRC: $att \notin ATT$ POC: $attributes(cl) = attributes(cl) \cup \{att\}$
creation of an operation: SIG: $addOperation(op) = OP'$ PRC: $op \notin OP$ POC: $OP' = OP \cup \{op\}$
creation of an implementation: SIG: $addImplementation : OP$ x $IMP \rightarrow OP$ PRC: $imp \notin IMP$ POC: $implementations(op) = implementations(op) \cup \{imp\}$

Concerning querying, most PMMS have a query language whose algebra includes relational-like operators (e.g., projection or selection) for models and metamodels. These algebra should be extended to be able to execute operations that can be defined with our proposed extension. To fulfill this need, we define the *RUN* operator. We only give the signature of this operator in table IV since its semantics depends on the processing done in the corresponding operation.

TABLE IV: Formalization of the RUN operator

RUN : OP x INPUT $\rightarrow$ OUTPUT
INPUT is an expression of input values. $INPUT = (I_C \oplus I_{I_C} \oplus I_{DT})^+ \oplus \emptyset$ $instOf$ is a function that returns the set of instances of a concept. $I_C = instOf(c_1) \cup instOf(c_2) \cup ... \cup instOf(c_n)$ $I_{I_C} = instOf(instOf(c_1)) \cup ... \cup instOf(instOf(c_n))$ I_{DT} represents simple types values (string, boolean, integer, etc.).
OUPUT is the output value. $OUTPUT = I_C \oplus I_{I_C} \oplus I_{DT} \oplus \emptyset$ $\oplus$ is the sum of types operator.

Examples:

If we have an operation `UMLClass2Table` that transforms an `UMLClass` to a `Table`, we can use it for instance to transform the `Student` class to the `T_Student` table. Thus, in this case, the *RUN* operator is invoked as follows:

$RUN(UMLClass2Table, Student)$. It returns the table $T\_Student$.

If we want to compute the age of an instance of the Student class (`Student1`), the *RUN* operator is invoked as follows:

$RUN(computeAge, Student1)$. It returns the value 26.

Next section presents the implementation of our approach on the OntoDB/OntoQL PMMS.

V. Prototyping: the BeMoRe Proposal

Our implementation consists of an extension of the OntoDB/OntoQL PMMS. Let us first introduce this PMMS.

A. The OntoDB Model Repository

The architecture of the OntoDB repository consists of four parts: one part for each abstraction level (data instance, models and metamodels) and one part for the system catalog of the database. These four parts consist of relational tables since this PMMS is based on PostgreSQL. Figure 4 (except the dashed box part) shows the main tables used to store metamodels, models and data of our example in OntoDB. The metamodel layer contains two main tables: `Class` and `Attribute` that store respectively classes and attributes of metamodels. Each class is associated to a corresponding table at the model level where class instances are stored; and similarly, each concept of a model is associated to a table at the data level to store instances.

B. The OntoQL Meta-Modeling Language

OntoQL is a declarative and object-oriented language used to create, modify, drop and query metamodels, models and data. In this section we present the OntoQL statements used for defining the different abstraction layers in OntoDB. Then, in the next section we present the extension of this language we have proposed and implemented for the definition and usage of operations at these different abstraction layers.

1) Metamodel definition: the metamodel part of OntoDB can be enriched to support new metamodels using the OntoQL language. For instance, the metamodel of our example (Figure 1) can be created with the following statements.

Listing 1: Statements for creating the metamodel (A)

```
CREATE ENTITY #UMLClass (
                #name STRING,
                #isAbstract BOOLEAN,
                #superClass REF (#UMLClass));

CREATE ENTITY #UMLProperty (
                #name STRING,
                #itsClass REF (#UMLClass));
```

In this statement the # prefix indicates that this element definition must be inserted in the metamodel level of OntoDB (an element of the model level does not have a prefix).

2) Model definition: once a metamodel is defined, we can create models conforming to that metamodel. For instance, the model of our example is created using the following statements:

Listing 2: Statements for creating the model (B)

```
CREATE #UMLClass University
PROPERTIES (name STRING);

CREATE #UMLClass Student
PROPERTIES (firstname STRING,
            lastname STRING,
            birthday DATE,
            itsUniversity REF (University));
```

3) Instance definition: similarly to the previous step, once models have been created with OntoQL and stored in OntoDB, they can be instantiated to create classes instances with a syntax similar to SQL. Next statements create instances of our example.

Listing 3: Statements for creating instances

```
INSERT INTO University VALUES ('ISAE–ENSMA');

INSERT INTO Student
VALUES ('Dupond', 'Durand', '06/21/1986', 123);
```

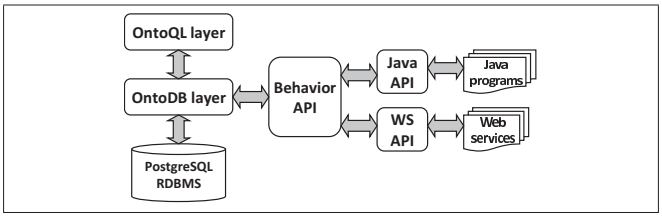


Fig. 3: BeMoRe architecture

Now that we have presented the OntoDB/OntoQL, we can describe the three main steps that we have followed to extend it with our approach (Figure 3). The first step consists of extending the model repository with structures and tables to store operations signatures and implementations descriptions and their dependencies. The second step consists of extending the exploitation language to create and to use operations and implementations. Finally, the third step consists of setting up an application programming interface (API) to make a bridge between the PMMS and the external programming environments. We detail each of these steps in next subsections.

C. Extending the OntoDB Architecture

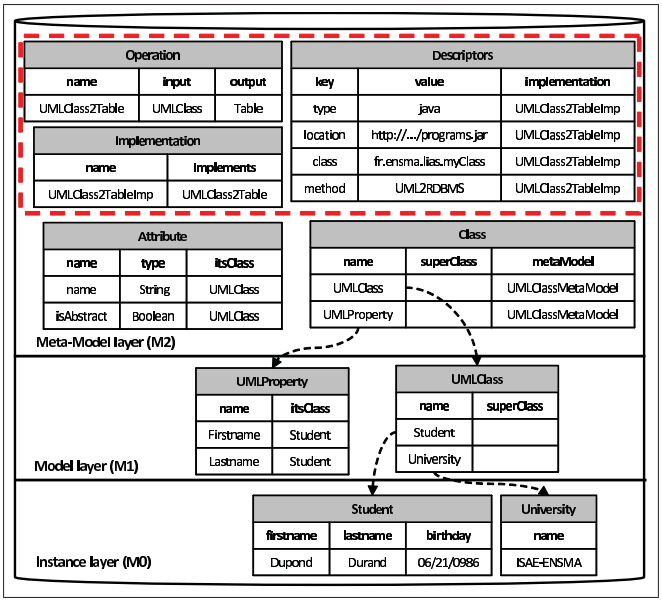


Fig. 4: Representing different model layers in OntoDB

The dashed box part of Figure 4 shows the three main tables resulting from the extension of the metametamodel layer of the OntoDB model repository. The `Operation`, `Implementation` and `Descriptors` tables store respectively operations definitions (the operation name, input and output), their associated implementations, and descriptions of implementations.

D. Extending the OntoQL Meta-Modeling Language

1) CRUD for Operations and Implementations: firstly, we have extended the OntoQL exploitation language with CRUD operations (Create, Retrieve, Update and Delete) to create, read, delete and update operations and implementations. For instance, the syntax to create an operation that transforms an UML class to a table is given below.

Listing 4: Statement for creating an operation

```
CREATE OPERATION #UMLClass2Table
INPUT (REF (#UMLClass))
OUTPUT (REF (#Table));
```

Once an operation is defined, we can define one or many associated implementations. The following statement creates an implementation of the `UMLClass2Table` operation.

Listing 5: Statement for creating an implementation

```
CREATE IMPLEMENTATION #UMLClass2TableImp
DESCRIPTORS (type = 'java',
             location = 'http://.../programs.jar',
             class = 'fr.ensma.lias.UMLClassUtils',
             method = 'class2Table')
IMPLEMENTS #UMLClass2Table;
```

This statement creates an implementation of the `UMLClass2Table` operation. It provides descriptors of a Java program stored outside the database. In particular, these descriptors specify the file location of the external program, the java class where the method is defined and the method to run.

2) Exploiting operations in Query: when an operation and at least one associated implementation are defined, this operation can be invoked in an OntoQL statement:

Listing 6: Example of an operation invocation

```
CREATE #Table T_Student AS
   SELECT #UMLClass2Table(c) FROM #UMLClass AS c;
```

This statement creates a Table (`T_Student`) from the resulting transformation of the `Student` class. Let us explain the process to answer this query. When we face an operation invocation in an OntoQL statement, we look into the repository to check the existence of the called operation, then we verify the compatibility of the arguments types with the operation parameters types. Next, if no implementation is specified in the statement, we look at the default implementation in the implementation table. After this process, we transmit the arguments and the implementation descriptors to the behavior API (see next subsection) in order to run the program and return the result.

3) Choosing a default implementation: we have also extended the OntoQL language with the possibility to define the default implementation if several implementations are available for an operation. Next statement defines a default implementation for an operation.

Listing 7: Specifying the default implementation for an operation

```
SET DEFAULT IMPLEMENTATION #UMLClass2TableImp
FOR #UMLClass2Table;
```

4) Specifying an implementation: we can also explicit the implementation that must be executed for an operation directly in an OntoQL statement. The statement below shows an example of this behavior.

Listing 8: Specifying the implementation to run in a statement

```
CREATE #Table T_Student AS
    SELECT #UMLClass2Table(c) FROM #UMLClass AS c
    USING IMPLEMENTATION #UMLClass2TableImp->#UMLClass2Table;
```

Our implementation requires to make a bridge between the PMMS and external programming environments. The solution we have adopted is presented in next subsection.

E. The Behavior API

An important part of our extension of the OntoDB/OntoQL PMMS consists of defining a mechanism to make the mapping between data types of the OntoDB/OntoQL system, and data types of the external implementations. Thus, we have set up a behavior API (Figure 3) that serves as an intermediate between the OntoDB/OntoQL PMMS and the external programming environments. In particular, it provides generic infrastructures (1) to specify data types correspondences between the two environments, (2) to execute remote programs and services, and (3) to generate a *wrap* that can be plugged on the top of the behavior API . For example, to support web services and Java methods invocation, we have implemented primitives of the behavior API and plugged on it the resulting wraps.

VI. Performance evaluation

As stated before in Section I, we have performed multiple applications of our work (e.g., [10]) in order to validate functionally our approach. In this section, we focus on performance evaluation only.

As a first step to study the scalability of our implementation, we compare the execution time of the following model query (similar results were obtained for a metamodel query).

Listing 9: The query used for our experimentations

```
SELECT computeAge(s) FROM Student AS s
```

We execute these queries using three types of implementations of the *computeAge* function: native stored procedure (*NSP*), external Java program (*EJP*) and local web service (*LWS*) on three different sizes of data (1000, 100000 and 300000 instances). These experiments were run on the OntoDB/OntoQL PMMS based on PostgreSQL 8.2 installed on a standard Intel Core Duo E6550 2.33 Ghz 3GB of RAM desktop machine.

The performance numbers for the query on the three data sizes and for the three implementations are shown in Figure 5. All times presented (in seconds) are the average of three runs of the queries.

As expected the invocation of NSP performs a factor of 4-5 faster than EJP and largely faster than LWS. As the EJP and LWS are called one time for each instance, the time of queries increases nearly linearly with the size of data. To optimize this process, this result suggests to design a Java method or a web services that takes as input a set of data instead of an individual data. A more complete study of the problem of query optimization for PMMS is part of our future work.

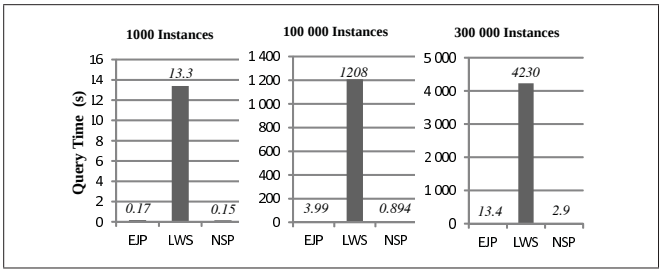


Fig. 5: Performance comparison

VII. Conclusion

In this paper, we have presented an extension of PMMS with a generic and flexible support of behavioral semantics. Our approach consists of providing the capability to introduce dynamically user-defined operations with multiple and heterogeneous implementations (external programs, web services, etc.). Our proposal has been implemented on the OntoDB/OntoQL PMMS and we have run several experiments to study the scalability of this implementation.

This work opens multiple perspectives. One of these perspectives consists of studying how derived operations could be defined using existing ones. Our idea is to be able to combine operations implemented with programs written in different languages and stored in different locations while respecting the order of the execution. Another perspective consists of choosing automatically the more efficient implementations to run when several implementations are available for a given operation. This feature will be especially useful for external web services that are not always available.

References

[1] P. A. Bernstein, A. Y. Halevy, and R. Pottinger, "A vision of management of complex models," *SIGMOD Record*, pp. 55–63, 2000.

[2] M. Jarke, M. A. Jeusfeld, H. W. Nissen, C. Quix, and M. Staudt, "Metamodelling with datalog and classes: Conceptbase at the age of 21," in *ICOODB*, 2009, pp. 95–112.

[3] D. Kensche, C. Quix, M. A. Chatti, and M. Jarke, "Gerome: A generic role based metamodel for model management," *J. Data Semantics*, vol. 8, pp. 82–117, 2007.

[4] S. Melnik, E. Rahm, and P. A. Bernstein, "Rondo: A programming platform for generic model management," in *SIGMOD Conference*, 2003, pp. 193–204.

[5] M. A. Hernández, R. J. Miller, and L. M. Haas, "Clio: a semi-automatic tool for schema mapping," in *SIGMOD Conference*, 2001.

[6] M. Koegel and J. Helming, "Emfstore : a model repository for emf models," in *ICSE (2)*, 2010, pp. 307–308.

[7] "project teresa." [Online]. Available: http://www.teresa-project.org/

[8] P. A. Bernstein and E. Rahm, "Data warehouse scenarios for model management," in *ER*, 2000, pp. 1–15.

[9] Y. Bazhar, "Handling behavioral semantics in persistent meta-modeling systems," in *RCIS*, 2012, pp. 1–6.

[10] Y. Bazhar, C. Chakroun, Y. A. Ameur, L. Bellatreche, and S. Jean, "Extending ontology-based databases with behavioral semantics," in *OTM Conferences (2)*, 2012, pp. 879–896.

[11] M. A. Jeusfeld, C. Quix, and M. Jarke, *ConceptBase .cc User Manual*, Tilburg University, RWTH Aachen, February 2013.

[12] J.-M. Hick and J.-L. Hainaut, "Strategy for database application evolution: The db-main approach," in *ER*, 2003, pp. 291–306.

[13] H. Dehainsala, G. Pierra, and L. Bellatreche, "Ontodb: An ontology-based database for data intensive applications," in *DASFAA Conference*, 2007.

Processing rhetorical, morphosyntactic, and semantic features from corporate technical documents for identifying organizational domain knowledge

Bell Manrique Losada
Faculty of Engineering
Universidad de Medellín
Medellín, Colombia
bmanrique@udem.edu.co

Carlos Mario Zapata Jaramillo
Faculty of Mines
Universidad Nacional de Colombia
Medellín, Colombia
cmzapata@unal.edu.co

***Abstract*—During the requirements elicitation (RE) process, transformations among languages occur from natural language—in which the stakeholders express their domain and needs—to a controlled language. One source of domain information to be used by the transformation process is related to technical documents belonging to the organizations (e.g. technical reports, legacy documents, and procedure manuals). Some properties of such documents are: different representation formats, high degree of ambiguity, and particular linguistics elements. The analysis and processing of such documents becomes complex because of these properties and, in turn, the complexity makes difficult both identifying the domain knowledge and understanding the associated processes. As a solution for an automated transformation, in this paper we define linguistics features to enable identification and composition of information units from a procedure manual, as a central task for the language transformation process in RE. The identified features are classified into rhetorical, morphosyntactic, and semantic features. Also, the features can be used to identify and represent organizational domain knowledge.**

Keywords-Natural language; Requirements elicitation; Domain knowledge; Technical documents; Texts processing

I. INTRODUCTION

Requirements elicitation (RE) is a process from the requirements engineering that involves seeking, uncovering, capturing, and elaborating requirements, based on activities of the business analysis initially performed. Such a process covers functional, behavioral, and quality properties of the software to be developed [1].

Several transformations among languages are needed in the RE process, commonly from natural language (NL)—in which the stakeholders express their domain, expectations, and needs—to controlled languages (CL) in which an analyst specifies the extracted information as models (e.g., domain, conceptual, logic, and process model). According to Mitamura *et al.* [2] CL is used for improving the clarity of expression of the source text and the quality of the analytical process. The domain information expressed by the stakeholders is used throughout the transformation process. Such information can be extracted from documents belonging to the organizations, usually from corporate technical information (e.g. technical reports, legacy documents, process manuals, and procedure manuals). Corporate technical information can describe processes, operations, regulations, guidelines, policies, and rules belonging to a company. By using such information, the analysts can understand the domain and business rules, in order to analyze them and transform them into a controlled discourse, useful in the RE process [3].

Organizational documents have some features like: different formats, heterogeneous information, high grade of ambiguity, and particular linguistics elements. Typically, by processing such documents we can analyze texts and clearly define the articulated information, but the big amounts of information lead the analysts to invest much effort for transforming it into explicit knowledge about organization [4]. In this environment, complexity arises and makes difficult the processing, when we are looking for information related to domain and business processes [3].

As a solution for an automated transformation, in this paper we define linguistic features to enable identification and composition of information units from a procedure manual, as a central task for the language transformation process in RE. We identify linguistic features and we classify them, as: rhetorical, morphosyntactic, and semantic. Also, we use such features as means to identify and represent organizational domain knowledge.

The remainder of this paper is organized as follows: in Section 2 we describe the conceptual framework about transformation process in RE, NLs and controlled languages, and linguistics features of the texts; in Section 3 we review and discuss the state of the art related to our approach; in Section 4 we propose a document pattern defining linguistics features for the model of the procedure manual; finally, in Section 5 we discuss conclusions and future work.

II. CONCEPTUAL FRAMEWORK

A. RE process

RE is the initial phase of requirements engineering. The main goals of RE are discovering, reviewing, documenting, and understanding the user needs and the system constraints. For achieving these goals, an analyst should increasingly and iteratively develop tasks involving the NL analysis and the language representation, respectively [5], including: understanding the domain, capturing and classifying requirements, and negotiating requirements [6].

RE is primarily concerned with the communication between the analyst and the stakeholder (e.g., customer, end-user) as a way to gather the relevant domain information [7], considered the basis of requirements.

B. Transformation process in RE

In the RE process, the analyst should identify and capture most of the information related to a specific domain. The main source is the stakeholder discourse. Supported by several techniques, the analyst obtains such information and uses it to describe the problem domain, analyze the information, and represent the software specifications by using models. To build a model, the analyst identifies conceptual elements, understands their relationships, and represents them by using a controlled language [8]. This process is a translation from a base language to another different. The translator should recognize and understand the NL-based symbols belonging to the domain and convert them into a symbols defined in a lexicon. Inside the translation process, the analyst—after capturing needs and expectations of the stakeholders—represents them by using technical models. The stakeholder validates the captured requirements and recognizes them into those models [9, 10].

C. Natural and Controlled Languages

According to Berry [11], most of the requirements specifications are written in NL. NL is used by humans and it guides the interpretation of what we experience with other senses (e.g., vision and hearing). The natural discourse is not formal and it lacks a structured definition, hierarchy, or external control. Commonly, NL is ambiguous and, therefore, barely adequate for processing and specification tasks, like machine learning [12]. The above reasons make notoriously difficult for a computer to process and understand NL, because a lot of relevant information is neither usually nor explicitly stated. In such cases, either the author or the speaker should imply such information [13].

Controlled languages (CL) restrict the vocabulary, syntax and/or semantics of a NL for reducing its ambiguity and complexity [14]. Rather than necessarily reflecting natural discourse patterns, CL defines terms to be used instead of other terms. Several fields, like knowledge representation, ideally use a formal language, since it has well-defined syntax, unambiguous semantics, and automated reasoning. The CL is used to bridge the gap between a NL and a formal language.

D. Analysis of linguistics features from texts

The linguistics analysis of discourses from documents can be performed by using methods of linguistic discourse analysis and corpus linguistics. We are focused on the following methodologies, based on the usage of language for constructing, interpreting, and exploiting documents.

1) Specialized Discourse Analysis: Specialized text analysis requires a unique process on the discourse analysis field [15, 16]. One way to address discourse analysis comes from the *gender* point of view. According to Swales [17, 18] and Yates [19], Parodi [20] defines genders as variations of a language which operate by means of linguistic features present in a text. Likewise, they are linguistically confined under their communicative purposes, participants involved, production contexts, usage contexts, and discourse organization modes, among others.

According to the above genre theory, we can analyze specialized discourses, because they are written by members of specific communities. According to Cabré [21], such discourse is derived from variables related to the subject and perspective of a topic, of which is important their analysis.

2) Rhetorical Analysis: Rhetorical analysis (RA) is concerned with the discourse construction, giving priority to the communicative intent of each gender [22].

From RA, the gender analysis is discussed in terms of rhetorical moves [18], which refer to the functional parts or sections of a genre. This approach for studying a particular genre comprises the analysis of a text and its description in terms of rhetorical structure (moves). This structure influences and restricts the contents and style [23] and allows the identification of linguistics features.

3) Corpus Linguistics: According to Parodi [20], the corpus linguistics constitutes a set of methodological principles for studying any language domain. Corpus linguistics allows for the description, analysis, and teaching of several types of discourse, from corpus pre-processed with the assistance of information technologies. Sets of linguistic features—operated by genders—can be identified from representative corpus. In this discipline, a corpus is a comprehensive collection of texts that are collected as sets of linguistic data reflecting the actual usage of a language [24]. Corpus-based approaches are focused on the word usage, frequency, collocation, and concordance [25].

III. Related Work

A. Processing of requirements documents in RE

O'Shea and Exton [26] use a content analysis for extracting requirements from a corpus of bug reports. This technique is suitable whenever categories can be defined prior to the texts analysis and it requires a certain extent of domain understanding. Fliedl *et al*. [27] analyzy requirements documents and comparing them with gathering requirements. This approach is good if lengthy texts are available and structured. In Software Product Line domain, authors like Clements [28] proposed processes for reviewing an studying source documents, but only oriented for defining the features surrounding products, instead of including linguistics analysis.

Based on semi-structured documents we found background as follows: RARE project [29] focused on parse texts based on a semantic network assisted by a thesaurus. They combine NLP with faceted classification for analyzing and refining requirements. From texts of non-functional requirements, Cleland-Huang *et al.* [30] work on detection of viewpoints. Meanwhile, Bajwa *et al.* [31] proposed mapping business rules to semantic business vocabulary. The before approaches are based on documents structured or semi-structured. We propose use techniques like they used, but from corporate documents writen in NL.

B. Analysis of linguistics features from documents

We have found approaches from disciplines like *spatial analysis* [32] trying to deal with designing agents for the deeper semantic and pragmatic understanding. From

Semantic web, Lee and Bryant [33] use contextual NLP to overcome the ambiguity and express the domain knowledge, and Lévy *et al.* [34] present an environment that enables semantic annotations of the document textual units (e.g. words, phrases, and paragraphs) with ontological ones (i.e. concepts, instances).

C. Identification of the knowledge domain and business process from documents

In the frame of knowledge engineering Dinesh *et al.* [35] proposed check the conformance to regulations with organizational procedures texts; Aussenac-Gilles *et al.* [36] apply the above on the ontological engineering domain; Rösner *et al.* [37] use techniques to automatically generate multilingual documents from knowledge bases. The described techniques for knowledge acquisition from documents are a good base for using in particular knowledge structures. Compared with our approach, these authors identify information that defines what stakeholders are permitted to do by using phrases previously created.

IV. Feature Identification Process

We define NL patterns as an abstraction of the information statement within the texts. In this proposal, features are part of the sentences appearing in certain positions of the document structure. We characterize the specialized discourse used in a specific corporate technical document called *procedures manual*, for further definition of patterns which guide the discourse.

A. Methodology of document analysis

a) Defining Guidelines. We use a methodology based on corpus linguistics [38], e.g. the work of Parodi [39]. We aim to carry out the descriptive and linguistic analysis of a procedures manual. We choose a particular type of corporate procedure manual called *standard operating procedures* (SOP).

b) Establishing the corpus: We collect a range of diverse SOP documents—taken from different contexts such as medical, forestry, laboratory, and academic—and perform a thorough analysis of the documents in the corpus in terms of content and structure. The corpus used consists of 50 documents written in in different English-speaking countries, which were downloaded from several websites. The SOP's were collected looking for full texts produced, created, or promoted in the real business environment. This analysis was developed from the empirical point of view, following the theory of gender analysis [20].

c) Analysing: The analysis was addressed by the *RA* trend. We selected a corpus sample for analyzing and defining a reference model, according to inductive method from Burdiles [40]. As a result, we proposed an structural pattern of SOPs in terms of macromoves (general structure or major sections) and moves (specific strucutre inside the macromoves).

B. Proposed Pattern

The proposed pattern includes the set of features in several levels of linguistic analysis. The first stage is described in the *subsection a*, corresponding to the rhetorical analysis. The internal analysis of each textual units corresponds with second stage and is presented in the *subsection b* and *c*, corresponding to the morphosyntactic and semantic analysis. Latter two analysis categories were supported by the corpus analysis tools AntConc 3.3.5w®—a freeware concordance program—[41] and TermoStatWeb™ [42].

a. Rhetorical features. These features are reflected in a reference model of rhetorical organization. Such model acts as a pattern. In structural terms, the pattern describes the organizational units of the SOP depending on the macro-moves and moves. In functional terms, the function patterns are related to the communicative purpose expressed by the author in the text.

The preliminar model of rhetorical organization was validated [43], and it changed in its internal structure, from 19 moves to 12 moves, as follows:

- *Macromove 1: Preamble/Overview*
  - Move 1.1 Identifying SOP
  - Move 1.2 Introducing the SOP
  - Move 1.3 Documenting conventions
  - Move1.4 Appointing regulations
  - Move 1.5 Establishing purpose
- *Macromove 2: Development*
  - Move 2.1 Defining procedure purpose
  - Move 2.2 Defining roles and responsibilities
  - Move 2.3 Identifying prerequisites
  - Move 2.4 Specifying procedure
- *Macromove 3: Closure*
  - Move 3.1 Adding supplementary information
  - Move 3.2 Including references

b. Morphosyntactic features. The first analysis of the documents is morphological, lexical, and syntactic.

The features involve morphosyntactic categories that can take a word or group of words, related to the morphology (depending on the word) or syntax (depending on the phrase). The features are described for such categories around of feature object (gender, number, person, tense, verbal mode, type of pronoun, word lemma, and so on). In the Table I, we present the preliminary morphosyntactic features identified. The first column is specifying the category (*C*)—morphological (*M*) or syntactical (*S*). The second and third column presents the *feature object* and the *feature description*. Numbers in last column correspond to a move in which is present the feature object (according to the number of move in previous section).

c) Semantic features. We present the semantic analysis in Table II with the description for each feature identified.

d) Identification of Organizational Knowledge. Based on the above preliminar features, analyzed for each *move* is possible to identify knowledge about organizational domain. A main analysis of this domain is in terms of concepts, prioritized according to frequency of occurrence (value in parentheses) as follows: *Project 1004; information (554), management (398), procedures (389), system (355), use (329), review (303), report (277), process (271), operations (264), data (262), staff (260), time (244), required (237), projects (236), decision (230), manager (228), SOP (227), standard (226), plan (217), personnel (216), implementation (215), manual (215), request (209), activities (207).*

We have identified the following information, which is described in the respective category (first column) in Table III. Its preliminary representation is in n-grams or collocations as sequence of words or terms (T) with major co-occurrence. In some cases we use A to define an actor-term, S a state, or V a verb-term.

i. Roles and actors of procedure [2.2, 2.3, and 2.4 moves].
ii. Responsibilities of procedure actors [2.2, 2.3, and 2.4 moves].
iii. Objectives of organization [1.2, 1.5, and 2.1]
iv. Procedures or functions of agents [2.2, 2.4]
v. Conditions/requirements to develop procedures [1.4, 2.3]

Note, however, that this analysis only has to be performed as part of the initial knowledge acquisition and preliminary processing.

TABLE I. MORPHOSYNTACTIC FEATURES

C	***Feature Object***	***Feature Description***	***Rethor. move***
M	Noun	Extensive use of abstract nouns (e.g. *Management, establishment*)	1.2– 1.5 – 2.1- 2.2
		Most common are uncountable nous (e.g. *information, process, data*)	2.3 – 2.4
	Adjec-tive	Demostratives most mommons: this, that.	1.1-1.2-2.1
		No common using posesives, and scarcely used its or his/her	-
		Undefined. They are rarely used (e.g. *any*)	2.3– 2.4
		The qualifying most common are: *required, apropriate*, *Successful,complete*. Their use is mainly subjective, rather than descriptive	1.4– 2.2– 2.3
	Verb	The copulative verb are the mode most used in SOP. Among them *be, is, are*.	2.2 – 2.3 – 2.4- 1.5
		Can conjugate in subjuntive mode, commonly for describing steps (e.g. *may*).	2.4 – 1.4
		Mainly are conjugated in indicative mode. (e.g. *use, review, report, describe*)	1.1 – 1.5 – 2.1- 2.4
S	Subord ina-tion	Are common place and time subordinate (using *where*, *when, after…*)	2.2 – 2.4
		Adjective or Noun Subordinate Clause are used, usually begins with a relative pronoun (e.g. *than*, *how*) or using *as*, *before*, or *until*	2.2 – 2.4 – 2.3
	Coordi -nation	The most frequent are c*opulative coordinate clauses* to express adding one stamentent to anocher and is introduced by conjunctions (e.g. *and, both, as well as*)	1.4 – 1.5 – 2.2 – 2.3 – 2.4
		The only explanatory coordinate clause found was *that is*.	2.4 – 2.3 – 1.4
	Syntac-tic clauses	Usage of adjective phrases (adjective+ noun) is common mainly in the roles and responsibilities section, and then, in the steps of the SOP (e.g. *appropriate project*)	2.2 – 2.4
	Verbal mood	Main moods: indicative and imperative, used to express capacity or obligation.	All
		Mood in conditional steps: *subjuntive* denoting positiblity/priority (e.g. *is possible, is important, was*…)	2.4 2.3 – 1.4
	Verbal periphr asis	Modal Periphrasis, indicating obligation (e.g. *may+verb*, *must+verb*) and its negative form, or indicating potencial mood (e.g. *Can+verb*, *Go to+verb*…)	2.4 2.3 – 1.4
		Aspectual Periphrasis occurs as: iniciation (e.g. *take*); progresion (*including performing*); habits (*used to, used for*); duration (*is/are +gerund*); Completion (*end of + infinitive*)	2.4
	Person	Prevails first person of plural in the sequence of procedures. In section 'roles an responsibilities' is used third person of singular. Exists low usage of pronouns.	All

TABLE II. SEMANTIC FEATURES

Feature Object	***Feature Description***
Langua-ge Function	*It is representative or referential focusing in the message, external reality, or referent.. The writer makes no judgments about the information or processes.*
Style or narrative mode	*Clear language, accurate, and direct. In most of cases exists clarity of ideas exposition. Descriptive style, due to names, locations, and qualifying objects, people, or processes. Narrative style, it may have actions in a temporal order looking for a specific purpose.*
Termino-logy	*Contains a large number of specific terms belonging to the body of knowledge and a lot of information shared between sender and receiver. Such information is produced by individuals who possess specific knowledge of a subject.*

TABLE III. IDENTIFCATION OF KNOWLEDGE

#	***n-grams***	***Examples***
i	.Project director .Staff .The-ACRONYM	*"Individual staff members are encouraged…"*
ii	.is-responsible-for-V	*"NAA is responsible for appointing independent auditors"*
	.a-shall-T-a-T	*"…Manager shall recruit a sufficient number…"*
	.will-be-V-for/to-T	*"…expense will be approved for reimbursement…"*
	the-use-of-a-T	*"…the use of a Flash Report process…"*
iii	.the-purpuse-of	*"The purpose of the NISN Activity…"*
	.to-address-the-T-V	*"…to address the concerns expressed…"*
	.action/activity-V-by-A	*"…activity approved by the Council."*
iv	.project-activities	*"…documents are properly maintained*
	.project-administration	*and retained for project activities."*
	.the-work-has/be/may	*"…the work has been authorized."*
v	.If-a-T-of-T	*"…If a difference of opinion…"*
	.If-a-T-of-V	*"…If a patient requieres…"*
	.If-the-T-is-T,then-T-is	*"…If the stop is gradual, then X is…"*
	.When-T-S,	"…when dropping an instrument,…"

CONCLUSIONS

In this paper, we proposed a structural, functional, and linguistics features of a SOP, as a corporate technical document which can be processed as input for the knowledge engineering process. Also, the usage of corporate technical documents in requirements elicitation has an incremental growth.

We present a first approach of a pattern, under the concept of a rhetorical organization model. By using the rhetorical analysis method, we show features in terms of functional and structural aspects, based on macro-moves, moves, and functions (communicative purposes). The proposed features comprise three macro-moves, which serve an overall communicative purpose, and 11 moves shaping the organization units of SOP.

The identification of organizational knowledge is achieved in terms of roles, actors and responsibilities of who are involved in procedures, objectives of organization, procedures or functions of agents, and conditions/requirements for developing procedures.

The performed analysis is a part of the initial knowledge acquisition and preliminary processing for understanding of organization domain. The knowledge will then be readily available for use by future applications in a specific domains, to validate the findings and then to automate the process.

ACKNOWLEDGMENT

This work is funded by the Vicerrectoría de Investigación from both the Universidad de Medellín and the Universidad Nacional de Colombia, under the project: "*Método de transformación de lenguaje natural a lenguaje controlado para la obtención de requisitos, a partir de documentación técnica*".

REFERENCES

[1] C. Castro-Herrera, C. Duan, J. Cleland-Huang and B. Mobasher, "Using data mining and recommender systems to facilitate large-scale, open, and inclusive requirements elicitation processes," Proc. of 16th IEEE Inter. requirements eng. conference, pp.165–168, 2008.

[2] T. Mitamura and E. Nyberg, "Controlled English for Knowledge Based MT: Experience with the KANT System," in Proceedings of TMI95, Leuven, pp.158-172, 1995

[3] Z. Zhang, "Effective Requirements Development – A Comparison of Requirements Elicitation techniques," in INSPIRE2007, Tampere, Finland, p.155-160, 2007

[4] S. Stein, Y. Lauer and M. El-Kharbili, "Using Template Analysis as Background Reading Technique for Requirements Elicitation," Available in citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.145.7424

[5] K. Li, R.G. Dewar, and R.J. Pooley, "Requirements capture in natural language problem statements," Heriot-Watt University, 2003. Available in http://www.macs.hw.ac.uk:8080/techreps/docs/files/HW-MACS-TR-0023.pdf

[6] S.Robertson and J. Robertson, Mastering the Requirements Process, 2nd ed., New Jersey: Addison Wesley, 2008.

[7] C.M. Zapata and B. Manrique, "Transforming Natural Language into Controlled Language for Requirements Elicitation: A Knowledge Representation Approach," in Knowledge Representation, Croacia: Intech Ed., 2012.

[8] A. Gangopadhyay, "Conceptual Modeling from Natural Language Functional Specifications," Artificial Intelligence in Engineering, vol. 15, Issue 2, pp. 207-218, 2001.

[9] B. Cheng, "Capturing the Requirements. Michigan State University," 2006. Available in: http://www.cse.msu.edu/~chengb/RE-491/Papers/atlee-chapter4.pdf

[10] C.M. Zapata and F. A. Villa, "La Gramática Básica de UN-Lencep expresada en HPSG," Revista Avances en Sistemas e Informática, Special Ed., vol.5 No.1, pp. 81-92, 2008.

[11] D. M. Berry, "Natural language and requirements engineering - Nu?," in CSD & SE Program University of Waterloo, vol. 20, 2003.

[12] I. Čeh, S. Pohorec, M. Mernik, and M. Zorman, "Robot Learning of Domain Specific Knowledge from Natural Language Sources," in Robot Learning, Croatia: Intech ed., 2010, p. 150.

[13] R. Schwitter, "Controlled Natural Languages for Knowledge Representation," in Proceedings from Coling 2010, vol. 1, pp. 1113–1121, Beijing, 2010.

[14] S. Hoefler and A. Bunzli, "Controlled Natural Language for Knowledge-Based Legal Information Systems," Amsterdam: John Benjamins, 2006.

[15] D. Bieber, "University Language: A Corpus-based Study of Spoken and Written Registers," Amsterdam: John Benjamins, 2006.

[16] C. Nickerson, "The usefulness of genre theory in the investigation of organizational communication across cultures," Journal of Research and Problem Solving in Organizational Communication, vol.1(3), pp.203-215, 1999.

[17] J. M. Swales, "Genre Analysis: English in Academic and Research Settings," Cambridge: Cambridge University Press, 1990.

[18] J. M. Swales, "Research genres. Exploration and applications," Cambridge: Cambridge University Press, 2004.

[19] J. Yates, "Control through communication: The rise of system in American management," Baltimore: Johns Hopkins Un. Press, 1989.

[20] G. Parodi, "Lingüística de corpus: una introducción al ámbito," Rev. de Lingüística Teórica y Aplicada, vol.46 (1), pp. 93-119, 2008.

[21] M.T. Cabré, "La Terminología: Representación y Comunicación. Elementos para una teoría de base comunicativa y otros artículos," Barcelona: IULA Universidad Pompeu Fabra, 1999.

[22] A. Azaustre and J. Casas, "Manual de retórica española," Barcelona: Ariel, 1997.

[23] I. Askehave and J.M. Swales, "Genre identification and communicative purpose: a problem and a possible solution," AppliedLinguistics vol. 22(2), pp. 195-212, 2001.

[24] M. Wynne, "Developing Linguistic Corpora: a Guide to Good Practice," Oxford: Oxbow Books, 2005.

[25] A. O'Keeffe's, "Strangers on the Line: A Corpus-based Lexico-grammatical Analysis of Radio Phone-in," PhD Thesis, University of Limerick, 2003.

[26] P. O'Shea and C. Exton, "The Application of Content Analysis to Programmer Mailing Lists as a Requirements Method for a Software Visualization Tool," 12th International Workshop on Software Technology Practice, pp. 30– 39, 2004.

[27] G. Fliedl, C. Kop, H.C. Mayr, A. Salbrechter, J. Vohringer, G. Weber and C. Winkler, "Deriving static and dynamic concepts from software requirements using sophisticated taggingstar," Data & Knowledge Engineering, vol,61(3), pp. 433–448, 2007.

[28] P. Clements and L. Northrop, "Software Product Lines: Practices and Patterns," Boston: Addison-Wesley, 2002.

[29] J. Cybulski and K. Reed, "Requirements Classification and Reuse: Crossing domains boundaries," in 6th Intl. Conf. on Software Reuse (ICSR'2000), Vienna, Austria: Springer, 1998.

[30] J. Cleland-Huang, W. Marrero, and B. Berenbach, "Goal-Centric Traceability: Using Virtual Plumblines to Maintain Critical Systemic Qualities," IEEE Transactions On Software Eng., vol. 34, No. 5, 2008.

[31] I. S. Bajwa, M. G. Lee and B.Bordbar, "SBVR Business Rules Generation from Natural Language Specification," Artificial Intelligence for Business Agility, vol. 3, 2011.

[32] C. Kray and A. Blocher, "Modeling the basic meaning of path relations," Proceedings of the 16th International Joint Conference on Artificial Intelligence, vol. 1, pp. 384–389, August, 1999.

[33] B. Lee and B. R. Bryant, "Contextual Natural Language Processing and DAML for Understanding Software Requirements Specifications," in 19th International Conference on Computational Linguistics, vol. 1, August, 2002.

[34] F. Levy, A. Guisse, A.Nazarenko, N., Omrane, and S. Szulman, "An Environment for the Joint Management of Written Policies and Business Rules," 22nd Internat. Conference on Tools with Artificial Intelligence. IEEE Computer Society, vol. 2, pp. 142-149, 2010.

[35] N. Dinesh, A. Joshi, I. Lee, and O. Sokolski, "Logic-based regulatory conformance checking," in 14th Monterey Workshop, Scholarly Commons Penn, 2007.

[36] N. Aussenac-Gilles, B. Biébow, and S.Szulman, "Revisiting Ontology Design: A Method Based on Corpus Analysis," in Knowledge Eng. and Knowledge Manag.. Methods, Models, and Tools: 12th International Conference, vol. 1937, pp. 27–66, 2000.

[37] D. Rösner, B. Grote, K. Hartmann, and B. Höfling, "From Natural Language Documents to Sharable Product Knowledge: A Knowledge Engineering Approach," Journal of Universal Computer Science, vol. 3, No. 8, pp. 955-987, 1997.

[38] R. Simpson and J. Swales, "Corpus Linguistics in North America," Selections from the 1999 Symposium, Ann Arbor: University of Michigan Press, 2001.

[39] G. Parodi, "Discurso especializado e instituciones formadoras," Valparaíso: Ediciones Universitarias de Valparaíso, 2005.

[40] G.A. Burdiles, "Descripción de la organización retórica del género caso clínico de la medicina a partir del corpus CCM-2009," Thesis in Linguistics, Universidad Católica de Valparaíso –Chile, 2011.

[41] L. Anthony, "Issues in the design and development of software tools for corpus studies: The case for collaboration," in Contemporary corpus linguistics , London: P. Baker Ed., pp. 87-104, 2009.

[42] P. Drouen, "TermoStat Web 3.0. Désormais utilisable qu'après enregistremen," 2003, Available in: http://olst.ling.umontreal.ca/~drouinp/termostat_web/

[43] C.M. Zapata and B. Manrique, "Designing a structural and functional pattern of a corporate technical document for requirements elicitation: A first approach from corpus linguistics," in Software Eng.: Methods, Modeling, and Teaching, vol. 2, Chapter1, pp. 1–10, 2012.

Swimming Activity Recognition Based on Slow Intelligence Systems

Wen-Hui Chen
Graduate Institute of Automation Technology
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

***Abstract*—Automatic analysis of swimming activities in an aquatic environment is useful but challenging due to the varying complex background. This paper presents a framework for swimmer motion analysis based on slow intelligence systems (SIS). There are five main components in the proposed SIS-based recognition framework: enumeration, propagation, adaptation, elimination, and concentration. A potential application of the proposed framework is the development of an early drowning detection system based on the observations of abnormal swimming activities.**

Keywords: slow intelligence systems; component-based software engineering; motion detection; digital image processing

I. INTRODUCTION

Automatic recognition of swimming activities from video sequences has important applications, such as swimming stroke analysis and drowning detection. The system for automatic swimming activity recognition (ASAR) mainly contains three distinct stages: swimmer detection, swimmer tracking, and activity recognition. Swimmer detection has to be performed before conducting the tracking and recognition process. Accurately detect swimmers from image frames is crucial for the development of a reliable ASAR system.

For computer vision, swimmer detection is the task of separating swimmers from the background in a given image. Background subtraction and temporal differencing are two common approaches applied to object detection in image processing. Background subtraction is suitable for static background and easy to implement [1]. Temporal differencing of the information computed from a sequence of image frames can be employed to reduce the number of false detection [2]. However, these two approaches are not suitable for the application of swimmer detection due to the dynamic and complex background.

Fig. 1 shows an image frame taken from an indoor swimming pool. It can be observed the background image of an aquatic environment is complex with some components that could make vision task become difficult, such as water splashes, ripples, and light reflections. Past researchers have developed some useful approaches to deal with this problem. The authors in [3] utilized local motion and intensity information to detect swimmers in each video frame. In [4], the authors developed a statistical model to represent the background image using a composition of dynamic background patches. The authors in [5] represented the background model in terms of homogeneous blob for an outdoor swimming pool to overcome the water disturbance problem.

Figure 1. The background of an indoor swimming pool with complex environment.

Once the swimmers have been detected, tracking algorithms can then be initiated to compute the correspondence from one frame to the next frame in the tracking process [6]. The dynamic changes of swimmer's poses, occlusion, and illumination variation are three major factors that cause additional technical challenges for activity recognition.

To recognize different swimming activities, a robust algorithm is required to keep track of the swimmers' movement in the tracked image sequence over time. This paper aims to present a concept of applying a slow intelligence framework in activity recognition. The rest of the paper is organized as follows. Section II presents the framework for swimming activity recognition. Section III describes the swimming activity recognition based on SIS. Conclusions are drawn in section IV.

II. THE FRAMEWORK FOR SWIMMING ACTIVITY RECOGNITION

The proposed ASAR framework, as depicted in Fig. 2, contains three main functional modules: swimmer detection, swimmer tracking, and slow intelligence system (SIS) based activity recognition. The functions of each module are described in the following subsections.

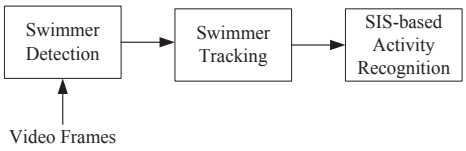


Figure 2. A framework for an automatic swimming activity recognition system.

Swimmer detection is the first stage in the ASAR system. In this stage, a model for representing the background image is required to segment swimmers. The background of a pool area normally contains few limited type of objects, such as the water object and the lane-marking object. Once the object in the pool area that does not belong to the water object is removed, a swimmer is easily to be detected.

Therefore, we need to build a computational model for representing the background so that the swimmers can be effectively separated from the swimming pool. The Gaussian mixture model (GMM) was adopted to represent the water object. The required parameters in GMM were estimated by expectation maximization. Fig. 3 depicts a GMM for representing the water area in a swimming pool.

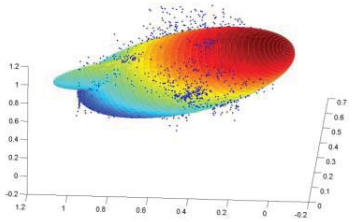
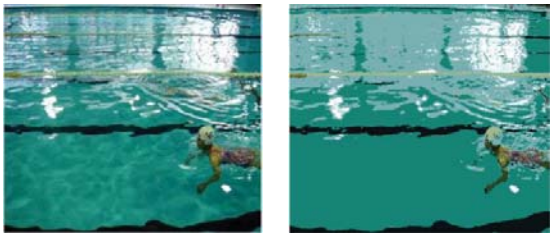


Figure 3. A Gaussian mixture model for the water area.

To reduce the influence of lighting changes, the HSV color space was used during computation as this color space can separate image intensity from the color information. We grouped each pixel of an image frame by intensity using the mean-shift clustering algorithm [7] to divide the image into groups.

Fig. 4 illustrates an example by using mean-shift clustering on an input frame. As it can be seen the resulted image after clustering is more suitable for classification as it contains distinct type of objects.

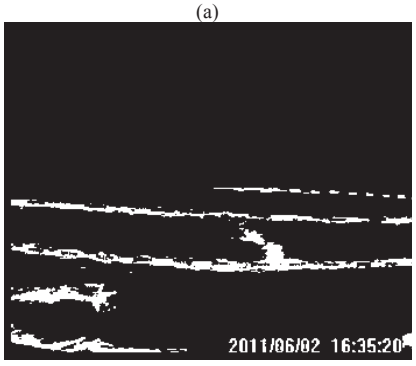


(a) (b)

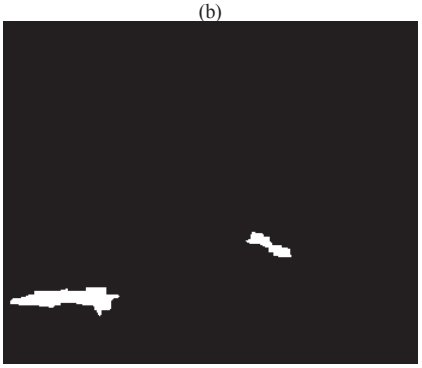
Figure 4. Mean-shift clustering: (a) an original input frame; (b) after mean-shift clustering.

Then, we need to remove noisy components that can undermine swimmer detection, such as splashes, ripples, and lane marking. The splashes and ripples can be identified through searching for high intensity values in the target image, while the lane marking can be identified by using the Hough transform technique. Once the noisy components are removed, the swimmers can be detected. Fig. 5 shows the process of removing noisy components. Note that the two blobs in the resulted image, as shown in Fig. 5(c), are the detected swimmers.

(a)



(b)



(c)

Figure 5. Swimmer detection: (a) an image frame with two swimmers; (b) background segmentation using GMM; (c) swimmers detection by remvoing splashes, ripples, and lane marking.

When swimmers are detected, an optimal estimator using Kalman filter is applied to track swimmers. The basic idea behind Kalman filter is to combine the system model and the measurement model to obtain the best estimation of the

location of the swimmers at each frame [8]. The purpose of swimmer tracking is to obtain the motion features and shape features of the swimmers. Based on these features, the recognition of swimming activities can be accomplished through SIS-based activity recognition.

III. The Proposed SIS-based Recognition System

A. Overview of Slow Intelligence Systems

A slow intelligence system is a new model for building an intelligent system through a process involving enumeration, propagation, adaptation, elimination, and concentration [9]. A slow intelligence system can be considered as a general purpose system characterized by being able to improve performance over time. Fig. 6 shows a typical building block of a slow intelligence system.

As shown in Fig. 6, an SIS is characterized by some functional blocks. It possesses decision cycles in the inference process to find the solution, which enables the SIS to cope with the changes of the environment and achieve the long-term goal.

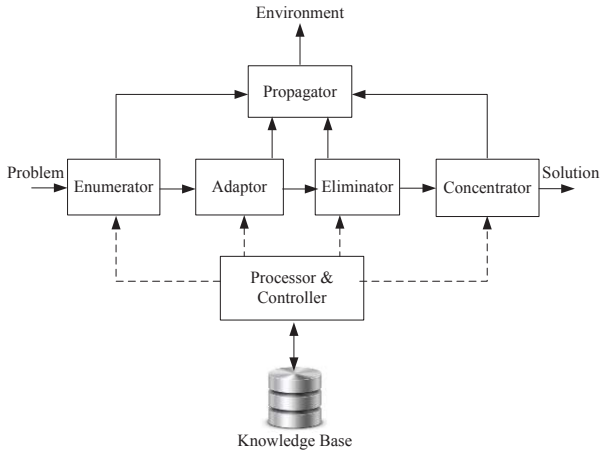


Figure 6. A typical building block of a slow intelligence system

B. SIS-based Activity Recognition

The process of recognizing swimming activities requires observing a sequence of image frames. This is related to the process of slow intelligence systems in their decision cycles for recognizing activities. The proposed SIS framework for swimming activity recognition is depicted in Fig. 7.

Captured by the overhead-mounted camera in a swimming pool, the input video streaming data is stored in an archive and is processed by the data processing unit. The function of the data processing unit is to remove water splashes, ripples, and land marking for swimmer detection. Compared with the background model, the location of swimmers in the foreground can be identified. Then a fitting ellipse is used to represent the shape of a detected swimmer, as shown in Fig. 8.

The input video data can be modeled by a sequence of observation states x_i, representing as a vector $\mathbf{x} = \{x_1, \ldots, x_T\}$. An observation state x_i is characterized by a set of features adapted from [10], including the coordinates of the fitting ellipse, the size of the fitting ellipse, the angle of the ellipse's major axis, and the minor to major axis ratio.

A swimming activity spans a certain amount of time. Therefore, we define an activity as a sequence of consecutive observation states. The SIS-based recognition stage consists of four stages: enumeration, adaption, elimination, and concentration. When feeding into the recognition stage, observation states are represented as a feature vector.

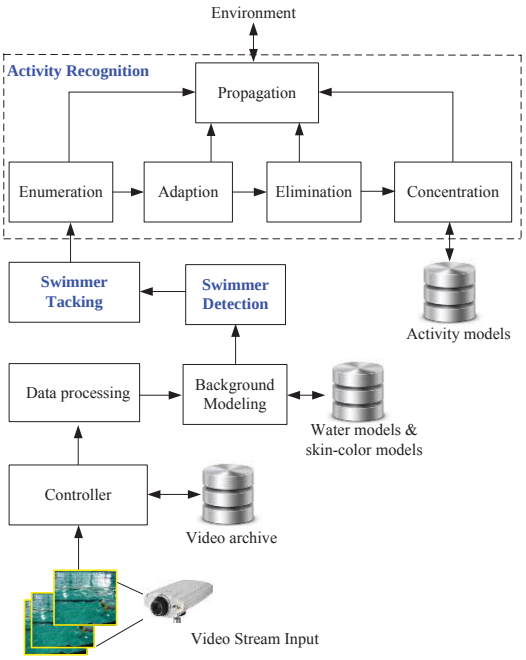


Figure 7. The proposed slow intelligence framework for behavior recognition.

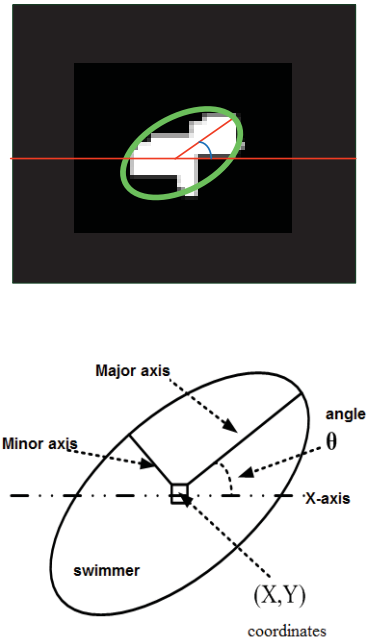


Figure 8. A swimmer is characterized by a fitting ellipse.

Acting as an autonomous entity, the SIS-based recognition unit consistently analyzes observation states, and compares them with activities defined in the activity model through the following processes.

1) Enumeration

All possible swimming activities defined in the activity model are enumerated for evaluation according to the current observation state. The hidden Markov model (HMM) is well suitable for modeling time series data, and is one of the widely used approaches to behavior recognition. In this study, the HMM is adopted to classify different activities.

In this study, five normal swimming activities (treading, freestyle stroke, butterfly stroke, breast stroke, back stroke) and one abnormal activity (drowning) were defined by the trained hidden Markov model as distinct activity classes.

2) Adaptation

The system continuously analyzes and adapts observation states according to the environment changes till one of the six predefined swimming activities is identified. The process of adaption is to suit and update the current states from observed inputs. Six HMMs are constructed for evaluating the most probable activity coincides with the observed inputs. All possible activities will be ranked and adapted to the environment according to HMM inference results.

3) Elimination

As each activity is represented by a set of feature sequences, the recognition of swimming activities from the input video stream can be considered as a pattern recognition problem. In this stage, unsuitable activities are eliminated to narrow down the search space so that more resources can be kept for the concentration stage.

4) Concentration

After ruling out unsuitable activities according to the evaluation of possible activity models, the inference of the SIS-based recognition unit can be more concentrated on searching for the most probable activity, which is beneficial to its performance improvement. The learning and searching process will continue until a solution is reach.

5) Propagation

The system constantly shares information with environment through the propagation process. When a specific activity is detected, such as drowning, a warning will be issued to the lifeguard for providing timely support. A regular camera has its limited viewing angle, so a single camera is usually not enough to cover the whole pool area. Therefore, multiple cameras are required to offer good coverage in practical applications. In this situation, the propagation unit plays an important role to communicate with other slow intelligence systems working towards achieving their goal.

IV. Conclusions and Future Work

In this paper, we have presented a framework of classifying swimming activities from image sequences based on slow intelligence systems. As there are various recognition algorithms that can be selected to design the activity classifiers, a simulation for testing and compare the performance is required before putting it into practical application. As a future study, a simulation will first be provided by using the SIS simulator developed by the University of Pittsburgh [11].

An SIS is characterized by employing super components, in the sense that multiple components can be activated either sequentially or in parallel to search for solutions. We have developed a visual specification approach using dual visual representations, and the user interface to design a component-based SIS system based upon the dual visual representations. We have applied this approach to build a simulator to design the Slow Intelligence System for Social Influence Analysis (SIA) [11]. This SIS simulator can be adapted to classify swimming activities from image sequences based on slow intelligence systems. Then, a recorded video clip from a practical indoor swimming pool will be employed to verify the proposed approach.

Acknowledgement

We would like to thank the National Science Council of the Republic of China for financial support of this research under contract numbers NSC 101-2221-E-027 -013.

References

[1] R. Jain and H. Nagel, “On the analysis of accumulative difference pictures from image sequences of real world scenes,” IEEE Trans. Patt. Analy. Mach. Intell., 1979, 1: 2 206-214.

[2] A. Monnet, A. Mittal, N. Paragios, and V. Ramesh, “Background modeling and subtraction of dynamic,” IEEE International Conf. on Computer Vision, 2003, 2 1305-1312.

[3] K. L. Chan, “Detection of swimmer based on Joint Utilization of Motion and Intensity information,” Conference on Machine Vision Applications, June 13-15, 2011, Nara, Japan.

[4] H. L. Eng, J. Wang, A. H. Kam, and W. Y. Yau, “ Novel region-based modeling for human detection within highly dynamic aquatic environment,” in Proc. IEEE Int. Conference on Computer Vision and Pattern Recognition, 2004, vol. 2, pp. 390-397.

[5] H. L. Eng, K. A. Toh, W. Y. Yau, and J. Wang, “DEWS: a live visual surveillance system for early drowning detection at pool,” IEEE Trans. Circuits and Systems for Video Technology, vol. 18, no. 2, Feb. 2008, pp. 196-210.

[6] A. Yilmaz, “Object tracking: a survey,” Journal of ACM Computing Surveys, vol. 38, no. 4, 2006, pp.1-45.

[7] D. Comaniciu, and P. Meer, “Mean Shift Analysis and Applications,” IEEE Trans. Information Theory, 1999, 21:1 32-40.

[8] L. Jetto, S. Longhi, and G. Venturini, “Development and experimental validation of an adaptive extended Kalman filter for the location of mobile robots,” IEEE Trans. Robot. Automat., 1999, 15:2 219-229.

[9] S. K. Chang, "A General Framework for Slow Intelligence Systems," International Journal of Software Engineering and Knowledge Engineering, vol. 20, pp. 1-15, 2010.

[10] W. Lu and Y. P. Tan, “A vision-based approach to early detection of drowning incidents in swimming pools,” IEEE Trans. Circuits and Systems for Video Technology, vol. 14, no. 2, Feb. 2004, pp. 159-178.

[11] S. K. Chang, Yao Sun and Yingze Wang, “Component-based Slow Intelligence System”, Journal of Internet Technology http://jit.niu.edu.tw, January 2013.

Image Steganography Using Fuzzy Domain Transformation and Pixel Classification

Aleem Khalid Alvi
School of Computing
Queen's University
Kingston, ON, Canada
aleem@cs.queensu.ca

Robin Dawes
School of Computing
Queen's University
Kingston, ON, Canada
dawes@cs.queensu.ca

***Abstract*—Image steganography offers many techniques to hide secret data from an eavesdropper. We use fuzzy logic and image processing techniques to develop a robust and highly imperceptible image steganography scheme. Our proposed scheme uses fuzzy domain transformation of secret data. We exploit image processing techniques for implementation of fuzzy pixel classification for a selection of cover pixels during the embedding of secret data. On the receiving end, un-embedding is performed using stego keys.**

Keywords: image steganography, data hiding, fuzzy logic, fuzzy classification, image processing

I. INTRODUCTION

Digital information hiding was born with advent of digital technology. Nowadays steganography techniques use different types of digital media such as text, image, audio, video, binary, or html files. Modern steganography techniques rely on data hiding techniques using modern media. Cryptography provides data security by applying encryption/decryption techniques. An encrypted message is susceptible to eavesdroppers' attacks if they know of its presence. The best solution is to hide the message existence by embedding it into cover media. Therefore, role of steganography is clear and strong with use of cryptography. Both techniques provide more secure communication between sending and receiving ends. Furthermore, some MS Windows operating system stores admin password in sam (security access manager) hive file (%SYSTEMROOT%\system32\config\sam) using a one-way-hash (*e.g.,* NT LAN Manager hash) [1]. The sam hive's file location is inaccessible to non-administrative users by default. However, it is vulnerable to offline attacks. Tools are available that can recover or simply reset/clear the password [2]. It is best to encrypt the password and hide using steganography. Therefore, combination of both techniques offers safety from attackers [3-4]. In this paper, we propose a unique and robust steganography method using fuzzy techniques. We transform secret data from a spatial domain to fuzzy domain before data hiding. The selection of a pixel from the cover media (an RGB image) depends on fuzzy pixel classification. Image processing techniques are employed for selection process of cover pixels using fuzzy pixel classification.

The rest of the paper is organized as follows. In Section II, we provide a survey on steganography that uses fuzzy logic techniques. In Section III, we describe the experimental setup for our proposed steganography technique. In Section IV, we depict the implementation procedure for the technique. In Section V, we discuss and analyze the results. Finally, we conclude with future research.

II. RELATED WORK

Classification of information hiding methods is based on many attributes such as cover objects, secret objects, hiding techniques, and current technologies. Researchers attempt to develop steganography systems with robustness, security, undetectability, imperceptibility (invisibility or perceptual transparency), and high capacity; however, every method has its own advantages and disadvantages [5]. We measure the capability and quality of steganography methods using these characteristics. Our proposed technique is the combination of domain transformation, data conversion, and substitution based on image properties. It is a kind of private-key steganography. Many steganography techniques can be found in the literature [5-6, 8-10, 11-15, 18]. We discuss steganography techniques that use fuzzy techniques as part of their implementation. Khursheed and Mir [16, 17] were the first to attempt to apply fuzzy logic for hiding data. They endeavor to embed information in a fuzzy logic domain. The advantages are lower computationally expense as compared to other domain transformation methods. Their method provides embedding versatility and safety from common cover attacks, as well as appropriate imperceptibility and payload capacity. However, the secret data is very sensitive and easy to destroy by small changes in the cover without changing any significant visibility.

Toony *et al.* [19] propose an image hiding method. They hide a secret image by employing a fuzzy coding/decoding method. A fuzzy coder compresses each block of the secret image into a smaller block and utilizes model-based steganography to hide the message in a cover image. This causes less distortion in the image and the result is a high quality stego image. The advantages are higher embedding rate and security enhancement.

Hussain *et al.* [20] propose a novel hybrid fuzzy c-means (FCM) and support vector machines (F-SVM) model for image steganography. The F-SVM model provides the capability for embedding the secret-message imperceptible for human visual system (HVS) during payload increment. This hybrid soft computing approach has advantages of complementary features of clustering (using FCM) and classification (using support vector machines).

Goodarzi *et al.* [21] propose a steganography scheme based on the least significant bit (LSB) steganography

mechanism and utilize hybrid edge detector (HED). The HED consists of canny edge detection and fuzzy edge detection algorithms. This scheme resists the HVS, Fridrich's methods, and steganalysis systems, which are based on statistical analysis. In addition, it produces higher quality stego images and high payload capacity.

Every steganography method has its limitations. Petitcolas *et al.* [6] discuss the limitations of some information hiding systems and attacks. Detection and destruction of secret data in a cover medium are considered attacks. They describe many attacks on information hiding techniques. Craver *et al.* [7] describe three kinds of attacks: robustness attacks, presentation attacks, and interpretation attacks.

In Section III, we describe our steganography scheme that utilizes a fuzzy inference system. The system exploits image processing techniques.

III. Experimental Set Up

We use LSB steganography that employs fuzzy logic with image processing techniques. The selection of a pixel for embedding depends on two image properties: silhouette (*i.e.,* edges) and texture (*i.e.,* a pixel contains the entropy value of the 9-by-9 neighborhood pixels around the corresponding pixel in cover image). The process of steganography uses the fuzzy inference system to embed the secret data (*i.e.,* text or image) into the cover image.

A. Fuzzy Image Representation and Domain Transformation

An image is the combination of pixel values. These values are represented in matrix form. An aggregation of these values shows an image to HVS with image properties such as brightness, homogeneity, noisiness, and edginess, *etc.* In general, an image representation in spatial and fuzzy domains is shown in (1) and (2), respectively as follows.

$$I_s = \bigcup_{m=1}^{M}\bigcup_{n=1}^{N} I_{mn} \tag{1}$$

A relationship or transformation of a pixel from the spatial to the fuzzy domain using membership function μ_{mn} is shown below.

$$\mu_{mn} = \mu_x(I_{mn})$$

Hence an image in the fuzzy plane is

$$\mu_f = \bigcup_{m=1}^{M}\bigcup_{n=1}^{N} \mu_{mn}$$

$$\mu_f = \bigcup_{m=1}^{M}\bigcup_{n=1}^{N} \mu_x(I_{mn}) \tag{2}$$

We use the general form of Gaussian membership function for image transformation from the spatial domain into the fuzzy domain as shown in Fig. 1. The specific image transformation function with fuzzifier f_h is shown below taken from Khurshid and Mir [16]:

$$\mu_{mn} = e^{\frac{-(I_{\max}-I_{mn})^2}{2f_h^2}} \tag{3}$$

Where f_h = fuzzifier, I_{max} = maximum pixel value of an image, I_{mn} = any gray level pixel value of an image I. In Fig. 1, $f_h = \sigma$, $I_{max} = b$, and $I_{mn} = x$.

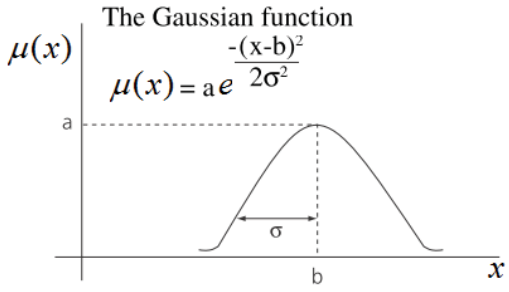


Figure 1. Simple Guassian function graph [23]

Analysis of (3) shows that the difference between I_{max} and I_{mn} changes the value of μ_{mn} significantly. If I_{mn} approaches I_{max} then μ_{mn} approaches 1. Similarly, when I_{mn} approaches 0 then μ_{mn} approaches a finite value c. Where c is shown as follows.

$$c = \mu_{mn}^0 = e^{\frac{-(I_{\max})^2}{2f_h^2}}$$

Therefore, values of μ_{mn} vary from c to 1. The fuzzifier (f_h) is the parameter that has effect on μ_{mn}. The inverse transformation function is required to transform the image back to spatial domain. In subsection B, we describe the embedding process using fuzzy inference system (FIS).

B. Fuzzy Inference System (FIS)

We use Mamdani fuzzy interference system (FIS) for the proposed steganography scheme [24]. Using the fuzzy inference process, a given input (a crisp input) maps to an output (a crisp output) using fuzzy logic methods. The fuzzy inference process requires membership functions, logical operations, and If-Then rules. We implement the FIS in following steps.

Step 1: Fuzzify inputs

Take input as pixel values and determine the degree to which they belong to each of the appropriate fuzzy sets. We use the membership function shown in (4) to transform the image pixel values from spatial domain to fuzzy domain.

Step 2: Apply fuzzy operator.

A fuzzy operator AND is used for combining the antecedents. The purpose of the fuzzy operator is to obtain one number that represents the result of the antecedents for specific rule and then apply a number to the output function.

Step 3: Apply implication method.

We use fuzzy pixel classification using image-processing techniques for LSB embedding in cover image [22]. We use three fuzzy based implication rules as follows:

a) IF (cover pixel = silhouette) AND (cover pixel texture value < M) THEN (Do not use cover pixel for embedding)

b) IF (cover pixel ≠ silhouette) AND (cover pixel texture value ≥ M) THEN (Do not use cover pixel for embedding)

c) IF (cover pixel ≠ silhouette) AND (cover pixel texture value < M) THEN (Use cover pixel for embedding)

In rule (c) the antecedent says that "if cover pixel is a regular pixel" then use this pixel for embedding secret data. Further, M is the mean of all pixels' entropy in the image. The input for the implication process is a single number given by the antecedent, and the output is consequent as part of a fuzzy set.

Step 4: Aggregate all outputs

We aggregate the output in the final cover image by writing all pixels combined in matrix form and store as an image.

Step 5: Defuzzify

Using HVS, visualize the cover image. The visualization after embedding should not differ from the visualization of the original cover image.

C. *Using Image Processing Techniques in Fuzzy Pixel Classification*

In the step 3 of subsection B, we use fuzzy based If-Then rules to apply fuzzy classification of potential cover pixels. The classification is used to select the appropriate cover pixel for embedding secret data. The purpose of cover pixel selection is to produce less disturbance and distortion in the embedded cover image with respect to HVS. We use texture and silhouette (edge) properties of an image.

The texture property is a statistical measure for image pixels. It provides information about the local variability of the intensity values of pixels in an image. For example, smooth texture in specific area of an image shows the range of values will be a smaller in the neighborhood around a pixel. The inverse is rough texture area where the range of pixel values will be larger.

The HVS testing first detects the distortion in edges of an image. Therefore doing no embedding in edges will preserve the edges of the image and increase imperceptibility against an eavesdropper's attack and decrease susceptibility to detection. We use the canny edge detection algorithm for edge detection in images and calculate entropy of each cover pixel by using a block of 9 x 9 neighborhood pixels around a selected pixel for getting texture information. We do not embed in edges and high texture areas to keep image more imperceptible for the HVS testing. The selection can be changed based on the given condition. For example in Fig. 2, we use rule based selection dependent on comparison with the mean texture value (M) of a cover image. Therefore, the capacity in the cover image may be increased if we replace the mean texture value (M) of the cover image with a texture value greater than M. However, this may decrease imperceptibility of the embedded cover image and may increase susceptibility for the eavesdropper.

D. *Methodological Steps*

In subsection B, we describe the FIS for the proposed steganography scheme. Fig. 2 provides step-by-step methodological information using a flowchart representation for embedding process on the sending end of the steganography system.

IV. IMPLEMENTATION

We implement the sending and receiving end algorithms using MATLAB. The stego keys for the receiver are original cover image, extracting software, and size and location of secret payload. It is easily possible to decrease susceptibility and reduce the possibility of cover, statistical, histogram, and profiler attacks if every time the sender communicates using a new image as a cover object. Therefore, the cover image should be unique and can use as a secret key. Similarly starting pixel location for embedding and size of secret data are combined to be used as a secret key.

The value of f_h works as a tuner of the image for embedding and extracting the secret image or data. The value of f_h can be adjusted based on the result of recovering the image. Therefore, the sender must check the recovered secret image before sending the embedded cover image to the destination. In our experiment, we use two different values of f_h for embedding and extracting the secret image. The value f_h = 65 has better results than f_h = 45 in HVS testing. Interestingly, we embed data using f_h = 65 and extract using f_h = 45 and f_h = 95.

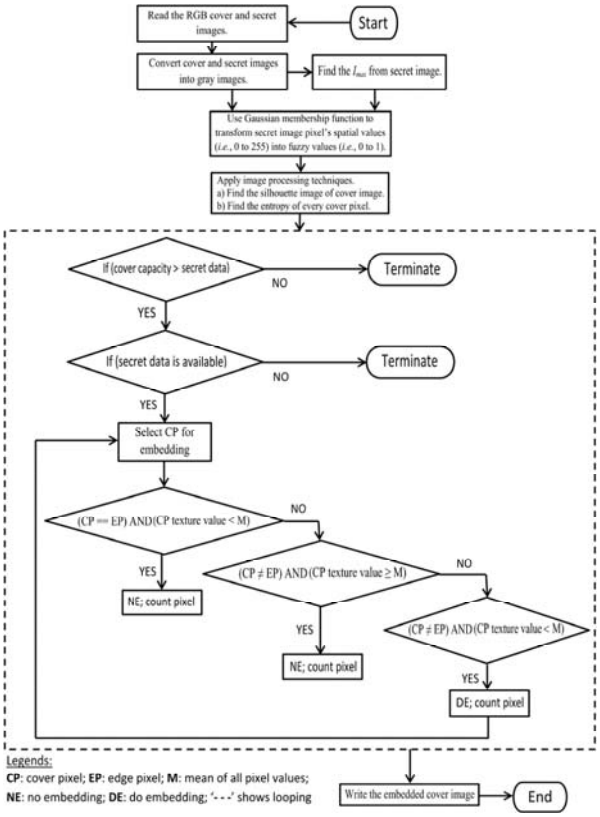


Figure 2. Sending end data embeding flowchart

The result is the successful extraction of secret image; however, at f_h = 45 the secret image has less perceptibility then at f_h = 95. Therefore, f_h work like a tuner for achieving good image quality in extraction process.

Fig. 3 shows the piece of MATLAB code for the implementation of the image processing techniques for fuzzy pixel classification. The embedding is LSB steganography technique; nevertheless, the selection of a cover pixel is based on fuzzy classification.

```
% ######### Image processing based fuzzy classification ##########

for i = 1 : L3
  for  j = 1 : L4
      if          XmnCoverEdge(i,j) == 0 && XmnCovEntropy(i,j) < M
                  % do nothing
                  n1 = n1 + 1;
      elseif      XmnCoverEdge(i,j) == 1 && XmnCovEntropy(i,j) >= M
                  % do nothing
                  m1 = m1 + 1;
      else
                  l = l + 1;
                  o1 = o1 + 1;
                  PmnSi_pixel = de2bi(PmnSi_R2(k,l),8,'left-msb');

                  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                  %%% Three colors data R, G, B of Cover image
                  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

                  cov_pixel = de2bi(XmnCovEmbed(i,j,1),8,'left-msb');
                  cov_pixel(7:8) = PmnSi_pixel(1:2);
                  XmnCovEmbed(i,j,1) = bi2de(cov_pixel,'left-msb');

                  cov_pixel = de2bi(XmnCovEmbed(i,j,2),8,'left-msb');
                  cov_pixel(6:8) = PmnSi_pixel(3:5);
                  XmnCovEmbed(i,j,2)= bi2de(cov_pixel,'left-msb');

                  cov_pixel = de2bi(XmnCovEmbed(i,j,3),8,'left-msb');
                  cov_pixel(6:8) = PmnSi_pixel(6:8);
                  XmnCovEmbed(i,j,3)= bi2de(cov_pixel,'left-msb');

                  if l == L1 , k = k + 1; l = 0; end
      end
      if o1 ==  NumberOfPixelRequireToEmbed ,  break;   end
  end
  if o1 == NumberOfPixelRequireToEmbed ,    break;  end
end
```

Figure 3. Image processing based fuzzy pixel classification algorithm

The use of fuzzy pixel classification creates unique characteristic in steganography scheme. It develops robustness in the steganography system.

Fuzzy transformation of an image provides the fractional values of irrational numbers. For storing these values up to significant number of decimal places needs excessive storage. Therefore, we use only two decimal places of fuzzy data (*i.e.*, fuzzy singleton values) of an image for storing into a cover image. We found that the round off of fuzzy singletons to two decimal places is the maximum round off position for the fuzzy data without losing pictorial information. However, on the receiving end, the transformation of secret data (image) into the spatial domain does not show exactly as the original; however, the result is always found to be appropriate and acceptable for visibility.

V. Analysis and Results

We use ImageJ (ver. 1.46r) software for analysis of results of the proposed steganography scheme. We have selected two standard images of size 512 × 512 as cover object (*i.e.*, Lena and Baboon). The secret images are Tomahawk missile and Spaceship with sizes 160 × 160 and 400 × 400, respectively.

We apply HVS, histogram (a statistical tool), and profile testing for analysis the imperceptibility characteristics of the results. These testing techniques select the best cover object among Lena and Baboon with respect to texture and silhouette (edge) properties. For this purpose, embed Tomahawk missile image (160 × 160) into both cover objects and use testing techniques. We have found the Baboon is the best selection as cover object. After selection of a cover image, we start testing of imperceptibility of the cover object by embedding the secret images in increasing capacity.

The selection process of cover pixels is shown in Fig. 2. We find the number of pixels in the cover object that are appropriate for embedding without producing perceptible disturbance in the texture and silhouette properties of the cover image. Therefore, before we start embedding, we know the available capacity of the cover object. The available capacities based on embedding criteria (shown in Fig. 2) for the selected cover images, *i.e.*, Lena.jpg and Baboon.jpg are 145,313 and 138,518 pixels, respectively.

The available cover object capacity may vary by any change in selection of texture value in an algorithm shown in Fig. 2. Permitting embedding in more cover pixels provides more capacity but on the other hand, it decreases imperceptibility.

TABLE I. Analysis of Proposed Steganograpy Algorithm Using Lena (Cover) and Tomahawk Missle (Secret) Images

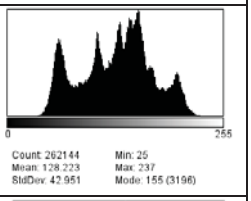
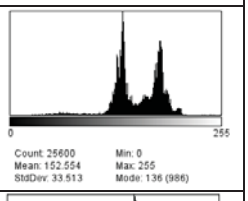
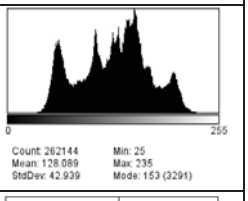
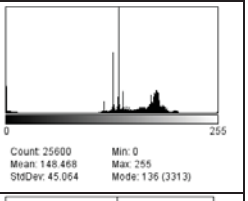
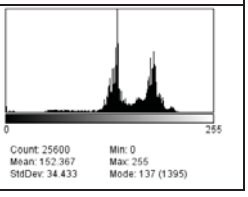
Type		Images	Histograms
Original Cover Image (Lena)			
Original Secret Image (Tomahawk Missile)			
Stego Image (Embedded Cover Image) (f_h = 45)			
Extracted Secret Image	f_h = 45		
	f_h = 65		

Table I shows embedding of the Tomahawk missile image into Lena image. In this case, secret data (160 ×160 =

25600 pixels) is embedded in Lena (available capacity = 145,313) and uses 17.62% of the available cover object capacity. HVS testing shows that original Lena image and Lena stego image has significant difference. The HVS testing uses zooming effect during comparison with the cover image. The difference between cover and stego objects is visible as light shades. This testing cannot be invoked without original cover image.

The histograms for the original Lena and stego Lena images look similar in shape. However, statistical testing shows the difference between their statistical values (*i.e.*, mean and standard deviation values). This can draw the attention of an eavesdropper and increase the susceptibility. The original secret image (*i.e.*, Tomahawk missile) is evidently visible after extraction; however, the extracted image at f_h = 65 has more visibility as compared to the extracted image at f_h = 45. The comparison of histograms of "extracted secret image at f_h = 65" with the "original secret image" shows much similarity to each other.

Table II shows embedding of Tomahawk missile image into Baboon cover image.

In this case, secret data (160 ×160 = 25,600 pixels) is embedded in Baboon cover image (available capacity = 138,518 pixels) and uses 18.48% available cover object capacity.

The histograms for the original Baboon and stego Baboon images are approximately similar in shape. HVS testing shows that the original Baboon image and Baboon stego image has no difference in visibility. However, statistical testing shows the difference between their statistical values (*i.e.*, mean and standard deviation values). However, this is less susceptible to an eavesdropper in comparison to the case shown in Table I, because humans use the HVS at first to judge anything. The original secret image (*i.e.*, Tomahawk missile) is evidently visible after extraction; however, the extracted image at f_h = 65 shows more clear view as compare to the extracted image at f_h = 45. The comparison of histograms of extracted secret image at f_h = 65 with original secret image shows more similarity among each other. Therefore, it shows that Baboon is a more reliable and imperceptible cover image and the value of f_h can be used as a tuning parameter for embedding and extracting a good visible secret image.

TABLE II. ANALYSIS OF PROPOSED STEGANOGRAPY ALGORITHM USING BABOON (COVER) AND TOMAHAWK MISSLE (SECRET) IMAGES

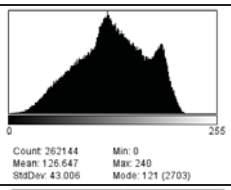
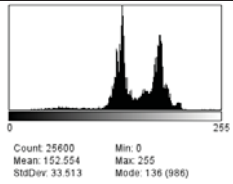
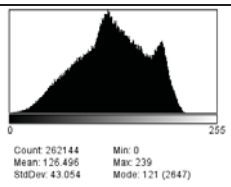
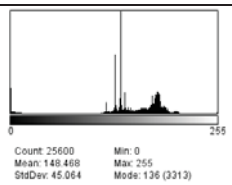
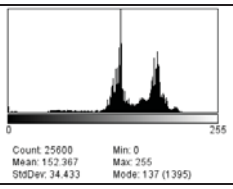
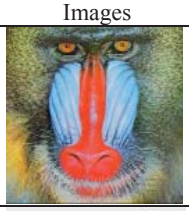
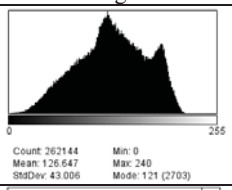
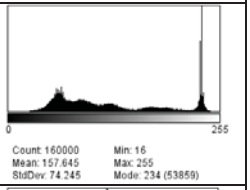
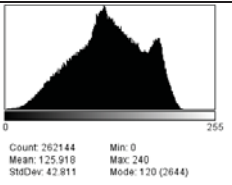
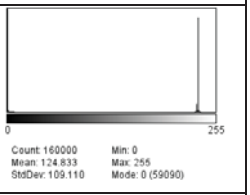
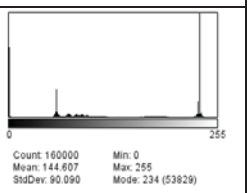
Type		Images	Histograms
Original Cover Image (Baboon)			
Original Secret Image (Tomahawk Missile)			
Stego Image (Embedded Cover Image) (f_h = 45)			
Extracted Secret Image	f_h = 45		
	f_h = 65		

TABLE III. ANALYSIS OF PROPOSED STEGANOGRAPY ALGORITHM USING BABOON (COVER) AND A BIRD (SECRET) IMAGES

Type		Images	Histograms
Original Cover Image (Baboon)			
Original Secret Image (Space ship 400 × 400)			
Stego Image (Embedded Cover Image) (f_h = 45)			
Extracted Secret Image	f_h = 45		
	f_h = 65		

We use profile testing for Lena and Baboon original and stego images and found it useless. Image profile testing provides the graph of selected area of an image; however, the differences between original and stego images are not visible in the graphs (provided that the size of secret data is small).

In Table III, we use 100% cover capacity of Baboon cover image for testing its strength in terms of imperceptibility. We embed the Spaceship secret image of size 400 × 400 pixels. In this case, secret data (400 × 400 = 160,000 pixels) is embedded in Baboon cover image (available capacity = 138,518 pixels). Since secret data is larger than available cover capacity; secret pixels are lost after using all cover capacity in cover image. The lost pixels are visible as a black strip in recovered images at $f_h = 45$ and 65. The HVS testing shows that Baboon cover image is highly imperceptible. However, histogram and statistical results shows significant differences between original and stego images.

VI. Conclusion And Future Work

We proposed steganography algorithm based on fuzzy inference system. Our fuzzy inference system uses fuzzy transformation and pixel classification techniques. The fuzzy pixel classification uses the image processing techniques by exploiting texture and silhouette properties. The results show that exploiting the image processing techniques with fuzzy logic increase imperceptibility in stego image significantly. In future work, more image properties can be considered for a cover image to further strengthen imperceptibility.

Acknowledgment

The authors would like to thank Prof. Hamid R. Tizhoosh, Faculty of Engineering, University of Waterloo for his online image processing tutorial and thoughtful comments on fuzzy representation for an image.

References

[1] Windows Registry Security - Part One, visited on May 13, 2013, http://www.registryon windows.com/registry-security-1.php.

[2] T. Fisher, "7 Free Windows Password Recovery Tools," About.com guide, visited on May 13, 2013, http://pcsupport.about.com /od/toolsofthetrade/tp/passrecovery.htm.

[3] Al-Najjar A.J., Alvi A.K., Idrees S.U., and Al-Manea A. M., "Hiding encrypted speech using steganography," In *Proceedings of the 7th WSEAS International Conference on Multimedia, Internet & Video Technologies*, Lang Congyan (Ed.), vol.7, World Scientific and Engineering Academy and Society, pp.275-281, 2007.

[4] Raphael A.J., and Sundaram V., "Cryptography and steganography – A survey," *International Journal of Compter Technology and Applications*, vol. 2, No. 3, pp. 626-630, 2011

[5] AL-Ani, Z.K., Zaidan, A.A., Zaidan, B.B., and Alanazi, H.O., "Overview: Main Fundamentals for Steganography," *Computer Engineering*, vol.2, pp.158-165, 2010.

[6] Petitcolas, F.A.P., Anderson, R.J., and Kuhn, M.G., "Information hiding-a survey," In *Proceedings of the IEEE* , vol.87, no.7, pp.1062-1078, 1999.

[7] Craver S., Yeo B.-L., and Yeung M., "Technical trials and legal tribulations." *Communications of the A.C.M.*, vol.41, no. 7, pp. 44-54, 1998.

[8] Anderson, R.J. and Petitcolas, F.A.P., "On the limits of steganography," *IEEE Journal on Selected Areas in Communications*, vol.16, pp.474-481, 1998.

[9] Cheddad A., Condell J., Curran K., and Mc Kevitt P., "Digital image steganography: Survey and analysis of current methods," *Signal Processing*, vol.90, pp.727-752, 2010.

[10] Johnson N.F. and Jajodia S., "Exploring steganography: seeing the unseen," *IEEE Computer*, vol. 31, no. 2, pp. 26–34, 1998.

[11] Bender W., Butera W., Gruhl D., Hwang R., Paiz F.J., and Pogreb S., "Applications for data hiding," *IBM Systems Journal*, vol.39, no.3 & 4, pp.547–568, 2000.

[12] Petitcolas F.A.P., "Introduction to information hiding," In: Katzenbeisser S., Petitcolas F.A.P. (Eds.), Information Hiding Techniques for Steganography and Digital Watermarking, Artech House, Inc., Norwood, 2000.

[13] Miaou S., Hsu C., Tsai Y., and Chao H., "A secure data hiding technique with heterogeneous data-combining capability for electronic patient records," in: *Proceedings of the IEEE 22nd Annual EMBS International Conference*, pp. 280–283., 2000.

[14] Fujitsu Ltd., "Steganography technology for printed materials (encoding data into images)," *Tackling new challenges*, Annual Report 2007, Fujitsu Ltd., pp.33, 2007, Access at: http://www.fujitsu.com/downloads/IR/annual/2007/all.pdf.

[15] Provos N. and Honeyman P., "Hide and seek: an introduction to steganography," IEEE Security and Privacy, vol.1, no.3, pp.32–44, 2003.

[16] Khursheed F. and Mir A.H., "Fuzzy logic based data hiding," In *Proceeding of Cyber Security, Cyber Crime, and Cyber Forensics*, Department of Electronics and Communication, National Institute of Technology, Srinagar, India, 2009.

[17] Mir A.H., "Fuzzy entropy based interactive enhancement of radiographic images," In *Journal of Medical Engineering and Technology*, vol.31, no.3, pp.220–231, 2007.

[18] Munirajan V.K., Cole E., and Ring S., "Transform domain steganography detection using fuzzy inference systems," In *Proceeding of IEEE Sixth International Symposium on Multimedia Software Engineering*, pp.286- 291, 2004.

[19] Toony Z., Sajedi H., and Jamzad M., "A high capacity image hiding method based on fuzzy image coding/decoding," In *14th International 'Computer Society of Iran' Computer Conference (CSICC'09)*, pp.518-523, pp.20-21, 2009.

[20] Hussain H.S., Aljunid S.A., Yahya S., and Ali F.H.M., "A novel hybrid fuzzy-SVM image steganographic model," In *Proceeding of International Symposium in Information Technology*, vol.1, pp.1-6, 2010.

[21] Goodarzi M.H., Zaeim A., and Shahabi A.S., "Convergence between fuzzy logic and steganography for high payload data embedding and more security," In *Proceedings of 6th International Conference on Telecommunication Systems, Services, and Applications,* pp.130-138, 2011.

[22] Castiello C., Castellano G., Caponetti L., and Fanelli A.M., "Fuzzy classification of image pixels," In *Proceedings of IEEE International Symposium on Intelligent Signal Processing*, pp.79- 82, 2003.

[23] Ibrahim A., "*Fuzzy Logic for Embedded Systems Applications*," Newnes, pp.213, 2003.

[24] Mamdani, E.H. and S. Assilian, "An experiment in linguistic synthesis with a fuzzy logic controller," In *Proceedings of International Journal of Man-Machine Studies*, vol.7, no.1, pp.1-13, 1975.

Smart Phone Based Indoor Pedestrian Localization System

Lokesh Agrawal
Department of Computer Science and Engineering
Indian Institute of Technology Roorkee
Roorkee, India
lokesh_agrawal@ymail.com

Durga Toshniwal
Department of Computer Science and Engineering
Indian Institute of Technology Roorkee
Roorkee, India
durgatoshniwal@gmail.com

***Abstract*— Indoor Positioning Systems (IPSs) are emerging computing systems that can locate objects or people inside indoor environment. This technology shows assurance for future mobile apps that can be used in malls, museums, hospitals, airports and college campuses for self localization. Despite advances in Global Positioning System (GPS) technology, indoor spaces are still out of reach of satellites. GPS signals are not designed to penetrate most construction materials. An IPS relies on nearby anchors or landmarks, and uses various sensing schemes including artificial vision, Wi-Fi, Bluetooth, Camera images etc. In this paper, we present a system that leverages the camera and Wi-Fi present in the smart phones carried by users, to track them as they traverse in indoor environment. It makes use of a radio map of an indoor environment. A significant challenge that our system surmounts is to estimate user's position without any prior user-specific knowledge, such as the user's initial location. Results obtained after conducting simulations demonstrate the validity and suitability of the proposed algorithm to provide high performance level in terms of position accuracy and scalability.**

Keywords- Localization, Mobile Phones, Sensors, Probability

I. INTRODUCTION

In recent years, improved features of smart-phones like its affordability, portability, better sensors and computational power has made a room for number of useful mobile applications. One such application is localization of an object in an indoor environment. Indoor localization shows promise for significant improvement in quality of living, especially for the visually impaired. By tracking user's location and providing directions audibly, this could help to improve the independence of the visually impaired [1]. Not only this, indoor navigation systems are also needed in complex public areas like shopping malls, museums, airports etc to provide position indication of the user so that user can easily find out where they are and can plan accordingly.

Global Positioning System (GPS) [2] is the most widely used technology that provides directions and locate places of interest anywhere on the earth where there is an unobstructed line-of-sight to requisite number of satellites. However GPS cannot be deployed for indoor use, because line-of-sight between receivers and satellites is not possible in an indoor environment. Hence comparing to outdoor environments indoor environments are more complex and designing applications for indoor navigation system raise new challenges. Some technical report [3], [4], [5] provides an overview of various available technologies that can be used for the design of an Indoor Positioning System. Based on these fundamental technologies, various IPSs have been developed by different companies, universities and research centers [6]. There are several commercial IPS systems such as Cricket [7], BAT [8], Active Badge [9], RADAR [10] etc.

We propose one such GPS-free Indoor Localization System which uses camera and Wi-Fi equipped smart phones for indoor positioning. The core idea involves combining the current strength of smart phones to estimate the location of user inside a closed premise. The key advantage of using our approach is that it does not require any network infrastructure. Neither any dedicated, specialized hardware is required. Users do not have to carry any device apart from what they already have, a smart phone. Our approach utilizes various Wi-Fi RSSI measurements and camera images for position estimation of an object. The remainder of this paper is organized as follows. In section 2 we provide background on previous localization work, followed by a discussion of our system architecture and design in section 3 and section 4 respectively . The report on simulated experiments and the results is discussed in section 5, followed by conclusions and future work in section 6.

II. RELATED WORK

Extensive research has been done to make a mobile phone application for indoor localization. A general survey is found in [6]. Here we provide only a brief survey. There has been a vast literature in pedestrian localization using custom hardware and rigid mounts. Previously specialized footwear's [3], helmets, and other wearable modules [1] fitted with various sensors have been used for the localization task. Unlikely they did not enjoy same success as far as consumer applications are concerned due to the expenses imposed for acquiring specialized hardware and the inconvenience in wearing them. Therefore smart phone implementable indoor localization solution is required which is more likely to obtain broad acceptance. The key advantage a smart phone offers are in terms of no social stigma to use in public, a billion of people already own one [11], easy distribution in the form of an app etc.. Many positioning systems have been developed over the years for indoor location estimation.

As an example GSM signal strengths have been used to determine which floor of a building the user is in [12].

However, accurate tracking within a floor of a building has not been achieved with GSM. Similarly, IR signals have some limitations because of interference from florescent light and sunlight [6]. A disadvantage of Bluetooth-based positioning system is that the system requires installation of Bluetooth tags. We chose to focus on a technique that uses camera images and Wi-Fi RSS along with filters to overcome the drawback of existing techniques.

III. System Architecture

Fig. 1 presents the overall architecture of our proposed system. This is one of the possible architecture, not necessarily the optimal one. We describe the high level flow of the information with the help of this diagram, and present the internal details in the next section. We begin with the preprocessing step where fingerprints of the Wi-Fi RSSI values are stored in a database. Next a user visits an unknown store with mobile phone. The phone senses the ambience automatically. The sensed values are recorded, matched and filtered to estimate the location of a user.

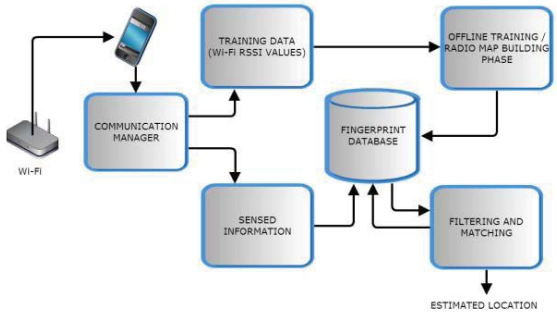


Figure 1 Architectural Overview of Proposed System

IV. System Design

We have studied the possibilities offered by 2 available resources in smart phones: Wi-Fi and camera. In this section we present the two main components of our system: Fingerprint Generation, Matching and Filtering.

A. Fingerprint Generation/ Training Phase

1) Fingerprinting Color Codes using Camera: Localizer discussed is made to work inside smartly designed buildings where each of the pillars inside a building is color coded with a predefined (not necessary for each color to be unique) set of colors. Just like modern metro stations and airports where color lanes and footsteps are made, for showing way to different platforms for the ease of passengers. We take fingerprints of these color codes and store them for location estimation.

2) Fingerprinting Wi-Fi : The consideration of Wi-Fi radios along with filters represents the most reliable approach for indoor localization in our experimental setup. RSSI values from Wi-Fi beacons deployed within a building allow us to obtain a fingerprint or radio map of different locations: the separation between these chosen locations for indoor environments is taken to be around a meter. We will estimate locations through the comparison of the current RSSI measurements with those stored in the radio map.

B. Fingerprint Matching / Online Phase

The mobile terminal infers its location through best matching between the current measurements being received and those previously recorded in the radio map.

1) FingerPrint Matching of Color Codes: Color coded technique discussed here requires less computational overhead as compared to real environment image matching in a map. Because color coded technique requires identifying color of any one pixel of the image captured.

In this approach user needs to sequentially capture few images of color codes mounted on walls and after taking few pictures device learns about its actual position inside the building and acts as a guide to them.

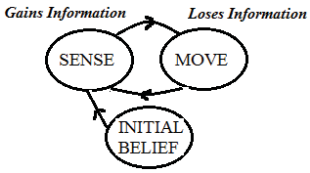


Figure 2 Indoor Localizer DFA

Fig. 2[15] shows the flow of methods inside our localizer algorithm. Before proceeding we describe each of these methods next and then finally describe how they are interacting to form an Indoor localizer.

a) Initial Belief: Initial belief is an initial location probability of an object in an indoor environment. Since mobile device is completely clueless about its location, it believes that every point in indoor environment is equally likely to be its current position.

b) Sensing and Matching: Sense method takes an image of color code mounted on nearby wall. Initially device senses that it is near to a color code X, our algorithm assigns all locations with color code X, a greater probability, whereas assigns all of the other locations, a decreased belief. This represents another probability distribution, called as posterior belief where the function is defined after the device sense measurements has been taken.

c) Filtering: Move method evaluates the belief once a sense method is called and a motion is made. As soon as device moves towards next color code our algorithm shifts its belief according to the motion made. Algorithm adds motion error too by flattening the beliefs to certain limit. This flattening is due to uncertainty in motion. Shifting and flattening of the belief is called as convolution [13]. The result of this convolution is the shifted and flattened belief function.

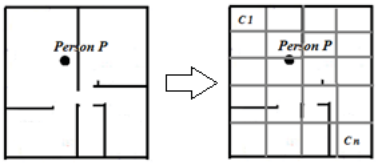
Once a movement is made device senses itself again. Just like the first measurement, sensing of color code will increase the probability function by a certain factor everywhere where there is corresponding sensed color code. So compared to first measurement, when we were in a state of maximum confusion, this time device have some idea of its location before sensing. This prior information, together with the second sensing measurement combines to give us a new

probability distribution. Therefore after second motion device is comparatively sure about its location than it was having after one or zero measurements. As a result after certain number of these iterations device will learn about its actual position in a building.

2) FingerPrint Matching of Wi-Fi RSSI: The approach discussed in this section makes use of particle filters [14] used in autonomous robotics for Indoor Pedestrian Localization. We assume we have certain number of fixed Wi-Fi access points inside a building where we will use our pedestrian localizer.

We use a Wifi-based signal strength fingerprinting approach and filter the result using particle filter. The signal strengths to several Wi-Fi access points (APs) measured by the phone at run time are compared with a signal strength map generated earlier. The difference between the expected reading for that position in the map and the actual signal strength reading is used to adjust the weight of the particle.

Algorithm works as follows:



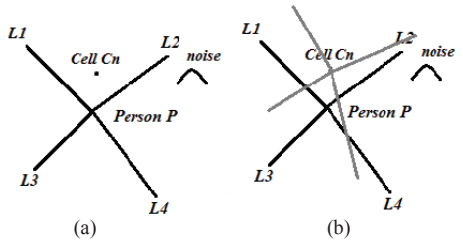
(a) (b)

Figure 3 Floor Plan of Indoor Environment

Consider a layout of an indoor environment as shown in Fig. 3. We divide floor into set of various (1m x 1m) cells, the corners of these cells act as an alias of an object (Person P) which needs to get localized refer Fig. 3(b).

Cell corners are basically a guess to a location of a person. These cells are represented as two tuple (X, Y) and are mapped with vector containing RSSI value and corresponding MAC address. These values together make a single guess. However, this single guess is not sufficient to find out approximate posterior of the person, but rather it is the set of several guesses that together generate an approximate representation for the posterior of the person. The goal of our algorithm is to have these aliases guess where the person might be moving. And also send them to test, a kind of "survival of the fittest test" means cells that are more consistent with the measurements are more likely to survive. As a result, places having high probability will collect more cells, and therefore are more representative of the person's posterior belief. These cells together after certain iterations, comes up with the approximate belief of the person.

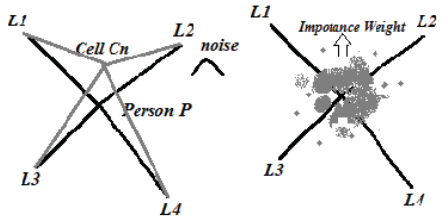
The algorithm maintains set of some random guesses of person location, each represented by a cell corner in Fig. 3(b). Assuming closed smart building contains various access points as L1, L2, L3… Ln and the device can measure the signal strength from these. The Fig. 4 shows the person's location and the landmark location with the measurement noise. Noise is included as there are chances of measurements being too short or too long and its probability is governed by Gaussian.



(a) (b)

Figure 4 Two Dimensional Representation of Landmark

Algorithm now builds a measurement vector with n values that are the signal strength values from landmarks. At this point if some cell corner derives that its coordinates (Cell Cn in Fig. 4) are somewhere other than that of persons' actual coordinate (labeled person P in Fig. 4) our algorithm takes the measurement vector from person's location and apply it to that cell corner. However, this finishes up being a poor measurement for that cell corner Cn see Fig. 4(b). The grey lines in Fig. 5(a) indicate the measurement vector we would have predicted if the cell were a good match for the person's actual location.



(a) (b)

Figure 5 Cells Forming Clusters

Here is the trick which is useful for us from particle filters concept, the mismatch between actual and the predicted measurement leads to an importance weight that tells us how important that specific cell corner is. The larger the importance weight is the more important that cell corner is. When we have a bunch of cell corners, each having its own importance weight; some are very likely, while others look very unlikely as indicated by the size of the cell corners see Fig. 5(b). Next these cell corners survive at random, but their survival probability will be proportional to their weights. That is, a cell corner with a larger weight will survive with higher probability than a cell corner with a smaller weight. Next we apply re-sampling. Re-sampling is a process of randomly drawing new cell corners from the old ones with replacement in proportion to the importance weight.

This means that after re-sampling, the cell corners having higher importance weight will remain intact, while the smaller ones will die out. This "with replacement" policy of selection is very important because it allows us to choose the high probability cells multiple times. This causes these cell corners to cluster around regions with high posterior probability see Fig. 5(b). By using a Gaussian, our algorithm measures how far away the predicted measurements would be from the actual measurements. And finally after certain motion, measurement and re-sampling steps the actual location of the mobile device is estimated.

V. EXPERIMENT AND RESULTS

The algorithm discussed were developed using python and the experimental setup for test was simulation of real indoor environment. Fig. 6 shows the floor plan. Eight pillars were color coded (denoted cc_n) for testing of first approach whereas four landmarks(A, B ,C ,D) were installed at known positions for the second algorithm test. Numbered black dots represent various test locations. The number of cells used for localization was set to 100 and resampling threshold was set to 0.7 times the number of cells.

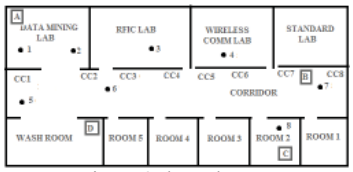
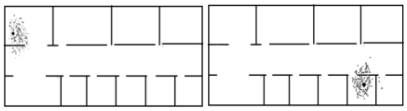


Figure 6 Floor Plan

Fig. 7 shows the distribution of cells once the algorithm converges for the test location 1 and 8.



Test Point 1 Data Mining Lab Test Point 8 Room Number 2

(a) (b)

Figure 7 Localization Results of Particle Filter Approach

The results are plotted in Fig. 8 and 9. The error was calculated using Euclidean distance between true and estimated positions. It is clear that the first algorithm convergence rate depends on the pattern of color codes mounted and therefore it varies for different location. Similarly second approach converges within 4-5 iterations and the total time taken was about few milliseconds per iteration.

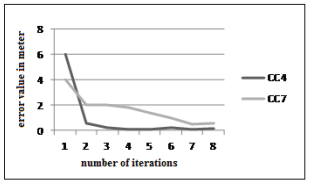


Figure 8 Localization error for Monte Carlo Method

In Fig. 8 the horizontal axis corresponds to number of pictures captured and vertical axis shows value of error in meters. It can be easily seen that results varies in accordance to location.

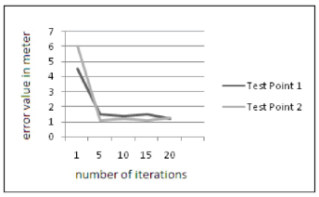


Figure 9 Localization Error for Particle Filter Approach

In Fig. 9 the horizontal axis corresponds to the number of iterations and vertical axis corresponds to value of error in meters. Results show that algorithm takes less than five iterations to converge.

VI. CONCLUSION AND FUTURE WORK

In this paper we demonstrated that our system for indoor localization can be implemented by using current state of the art smart phones. We integrated the sensing capabilities along with filters to deliver accuracy up to (1-1.5) meters without the requirement for complex hardware and dedicated infrastructure that other existing solutions need. In previous section we discussed that we obtained favorable results in terms of accuracy and complexity involved and in fact, to the best of our knowledge, our application is the first one delivering such accurate results by just using only the hardware capabilities of existing smart phones.

There exist several other interesting directions which we can explore in our future work. As an example this application could be made more advantageous by adding an extra feature of getting the shortest path to the destination once the person is localized.

REFERENCES

[1] Nisarg Kothari, Balajee Kannan, and M Bernardine Dias, CMU-RI-TR-11-27 August 2011. *"Study on Robust Indoor Localization on a Commercial Smart-Phone"*. Technical Report Robotics Institute, CMU.

[2] B. Hofmann, H. Wellinhof, and H. Lichtenegger, *"GPS: Theory and Practice"*, Springer-Verlag, Vienna, 1997.

[3] M. Vossiek, L. Wiebking, P. Gulden, J. Wiehardt, C. Hoffmann, and P. Heide, *"Wireless Local Positioning"*, IEEE Microwave Mag., vol. 4, Issue 4, December 2003, 77-86.

[4] J. Hightower and G. Borriello, *"Location sensing techniques"*, Technical Report UW CSE 2001-07-30, Department of Computer Science and Engineering, University of Washington, 2001.

[5] J. Hightower and G. Borriello, *"Location Systems for Ubiquitous Computing"*, IEEE Computer Society Press, vol. 34, no. 8, 2001, 57-66.

[6] Gu. Yanying, A. Lo, I. Niemegeers, *"A survey of indoor positioning systems for wireless personal networks,"* Communications Surveys & Tutorials, IEEE, 11, 1(First Quarter 2009), 13-32.

[7] N. Priyantha, A. Chahaborty, and H. Balakrishnan, *"The cricket location-support system,"* in Proc. the 6th ACM International Conference on Mobile Computing and Networking, Boston, MA, Aug. 2000.

[8] A. Harter, A. Hopper, P. Steggles, A. Ward, and P. Webster, *"The anatomy of a context-aware application,"* in Proc. the 5th ACM International Conference on Mobile Computing and Networking, Seattle, WA, Aug. 1999.

[9] R. Want and A. Hopper, *"Active badges and personal interactive computing objects,"* IEEE Trans. Consumer Electron., vol. 38, no. 1, pp. 10–20, Feb. 1992.

[10] P. Bahl and V. N. Padmanabhan, *"RADAR: An in-building RF-based user location and tracking system,"* in Proc. IEEE International Conference on Computer Communications, 2000, 775–784.

[11] R.K. Bivens *"The internet, mobile phones and blogging."* Journalism Practice 2, no. 1 (2008): 113-129.

[12] M. Dru and S. Saada, *"Location-based mobile services: The essentials"*, Alcatel Telecommunications Review, 2001, 71-76.

[13] www.en.wikipedia.org/wiki/Convolution.

[14] A. Doucet and N De Freitas. *" Sequential Monte Carlo methods in practice."* Edited by Neil Gordon. Vol. 1. New York: Springer,(2001).

[15] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "Monte Carlo Localization for Mobile Robots", IEEE International Conference on Robotics and Automation (ICRA), 1999.

A Formal Cost-Effectiveness Analysis Model for Product Evaluation in E-Commerce

Ran Wei and Haiping Xu
Computer and Information Science Department
University of Massachusetts Dartmouth, North Dartmouth, MA 02747, USA
{rwei, hxu}@umassd.edu

***Abstract*—Due to the inherent nature of e-commerce, customers usually have to take certain level of risks while shopping online. To deal with such risks and their associated uncertainty, most of the e-commerce websites provide product review ranking services to help customers to make purchase decisions. However, such services are typically not reliable because the ranking results are usually based on the averages of review scores given by different reviewers without considering their reliability. In this paper, we propose a formal cost-effectiveness analysis model for product evaluation in e-commerce, which takes the reliability of each review into consideration. We define four pieces of evidence, namely positive reviews, the number of positive reviews, negative reviews, and the number of negative reviews, and combine them using the Dempster-Shafer (D-S) theory. Based on the belief values about the product, we can calculate its effectiveness, and further derive its cost-effectiveness value by considering its minimal price. By ranking various products sold by different vendors based on their cost-effectiveness values, our approach can greatly help customers to make decisions on selecting the most cost-effective products for online purchasing.**

Keywords-E-commerce; product reviews; cost-effectiveness; reasoning under uncertainty; Dempster-Shafer (D-S) theory.

I. INTRODUCTION

As e-commerce techniques are growing rapidly, people today increasingly shop online instead of directly shopping in physical stores. However, because of the inherent nature and complexity of e-commerce environments, the evaluation and selection techniques for purchasing favorable products that fit a customer's needs could be very sophisticated. Customers typically lack the technical knowledge about the product to be purchased. Furthermore, decision making on selecting online products has become more complex due to the variety of brands and tremendous number of similar products available on the electronic market.

To help customers to select the favorable products, many companies, such as Amazon, have attempted to develop suitable and effective product evaluation mechanisms [1]. However, such mechanisms are typically not well employed as expected because the information redundancy and complexity on the review pages usually make customers lose patient or even get confused. Due to this shortcoming, customers often only check the average ratings rather than reading through all the product reviews across multiple web pages.

We noticed that some review ranking services such as the average star ratings were not always reliable. This is because much information related to a product review (e.g., the helpfulness of the review rated by other customers and the qualification of the reviewer) was usually ignored by users, which otherwise could be used as evidential knowledge for evaluating the reliability of the product review. Thus, in our research, we consider such information as hidden knowledge that can be defined as multiple attributes. We first set up the evaluation criteria for each attribute quantified using certain scales. Then we propose a cost-effectiveness analysis model based on the Dempster-Shafer (D-S) theory to rank product alternatives. Note that the D-S theory is a mathematical theory of evidence, which is a powerful tool to support reasoning under uncertainty [2]. Using the Dempster's combination rules, we are allowed to combine various pieces of independent evidence and reach a high-level degree of belief for specific hypotheses. In this paper, we consider the hypotheses whether a product is a favorable one that is worth buying or it is an unfavorable one that is not worth buying. To verify these hypotheses, we divide the available product reviews into two sets, namely the positive reviews and the negative reviews. We calculate the belief values for each set by combining the weighted average review score of a set and its number of reviews as independent pieces of evidence using the D-S theory. Then the two sets' belief values are combined again as independent pieces of evidence to calculate the effectiveness of the product, which can be used to further derive its cost-effectiveness value by taking the product's minimal cost into consideration. By ranking various products sold by different vendors based on their cost-effectiveness values, our approach can greatly help customers to make decisions on selecting the most cost-effective products in online shopping.

II. RELATED WORK

The D-S theory has been used in various areas to support reasoning under uncertainty. Dong *et al.* proposed a practical shill detection mechanism in online auctions using the D-S theory of evidence [3]. The approach takes multiple pieces of evidence from different information layers into account, detects shilling behaviors and assists decision making on shill bidders. Panigrahi *et al.* developed a fraud detection system in mobile communication networks [4]. They utilized the D-S theory to combine multiple pieces of evidence from the rule-based component and compute an overall suspicion score to help users filter suspicious incoming calls. Yang *et al.* presented an evidential reasoning approach that could be used to solve uncertain decision problem with both quantitative and

qualitative attributes [5]. They proposed an alternative way to deal with hybrid multiple-attribute decision-making problems with uncertainty. Different from the above approaches, in this paper, we adopt the D-S theory to develop a cost-effectiveness analysis model. Our approach can be used to combine multiple pieces of evidence to evaluate the quality of a product by calculating its effectiveness value based on the review ratings and their associated information.

There are many previous research efforts related to our approach for supporting decision making under uncertainty. Li *et al.* proposed a grey-based decision-making approach to the supplier selection problem [6]. Their approach employed the grey theory, which was one of the methods for mathematical analysis of systems with uncertain information. Denguir-Rekik *et al.* developed a choquet integral-based decision-making method for propagating possibility distributions using generalized weighted mean aggregation operators [7]. They emphasized on possibility distributions rather than precise quantitative evaluations, and used uncertainty indicators to give a user some idea about other people's variability of the evaluations. Herrera *et al.* proposed a fusion approach for managing information evaluated in different linguistic term sets [8]. The aim of their approach is to manage information assessed in different linguistic term sets together in a decision-making problem with multiple information sources. Huynh *et al.* reanalyzed the evidential reasoning (ER) approach, and proposed a general scheme of attribute aggregation in multiple attributes decision-making problem under uncertainty [9]. They showed that new aggregation schemes satisfied the synthesis axioms, for which any rational aggregation process should grant. Most of the above approaches are based on calculating probabilities of certain events, thus they are not readily scalable for decision making with newly acquired evidence. In contrast, we use the D-S theory that is an evidence-based approach to calculate the brief values about a product, which can be easily refined and updated using Dempster's rule of combination when new pieces of evidence about the product are acquired.

In addition, there are some previous research efforts on product analysis. Cho *et al.* developed a product taxonomy for collaborative recommendation in e-commerce [10]. In their approach, they used web usage mining technique to enhance the quality recommendation and system performance. Sarwar *et al.* proposed an analysis recommendation algorithm that could produce useful recommendations to customers [11]. They used traditional approaches such as data mining and dimensionality reduction techniques to handle large-scale purchase and preference data. Although the above approaches are useful in deriving product recommendations, they require analysis of a large amount of data sets. In contrast, our approach emphasizes on analyzing the review information related to a specific product, thus it is much more efficient than data mining based approaches.

III. Dempster-Shafer Theory

The D-S theory is a probabilistic reasoning method, which was developed to solve problems with uncertainty and incompleteness of available information [2, 3]. Let Θ be a finite set of mutually exclusive possible hypotheses, called the *frame of discernment*. For example, when we consider the domain of product evaluation, each product is considered either *favorable* or *unfavorable* for buying, depending on the nature of the evaluated properties and the quantified values of the review evidence. Thus, the frame of discernment for a product can be defined as Θ = {*favorable*, *unfavorable*}. The power set of Θ that contains all subsets of Θ is defined as 2^{Θ} = {$\varnothing$, {*favorable*}, {*unfavorable*}, Θ}.

In the D-S theory, a belief mass is assigned to each element of the power set 2^{Θ} in the interval between 0 and 1. Thus, the basic mass assignment (*BMA*) function *m* is defined as

$$m : 2^{\Theta} \rightarrow [0, 1],$$

which satisfies the following two requirements:

$$m(\varnothing) = 0 \tag{1}$$

$$\sum_{A \in 2^{\Theta}} m(A) = 1 \tag{2}$$

In Eq. (1), the mass of the empty set $\varnothing$ represents the measurement for zero state, thus it is defined as 0. Eq. (2) represents that the sum of masses of the elements in the power set equals 1. For example, in our product review example, since $m(\varnothing) = 0$, we have $m(\{favorable\}) + m(\{unfavorable\}) + m(\Theta) = 1$. Note that $m(\Theta)$ represents the mass for conflicting states (both favorable and unfavorable in our example, i.e., a hypothesis says that a product is both favorable and unfavorable), thus it can be interpreted as the measurement for uncertainty. For clarification purpose, in the rest of the paper, we use the notation $m(U)$ to represent $m(\Theta)$, where U represents uncertainty.

Another important function for a set of states (or a hypothesis) *A* is called the *belief* function, which is defined as the sum of the masses of all subsets of *A*.

$$belief\,(A) = \sum_{B \subseteq A} m(B) \tag{3}$$

Intuitively, any portion of the belief committed to the hypothesis *A* must also be committed to any hypothesis that it implies. To obtain the total belief in *A*, one must therefore add to $m(A)$ the quantities $m(B)$ for $\forall\ B \subseteq A$. In our example, we have two hypotheses, namely 1) the product is a favorable one; and 2) the product is an unfavorable one, both of which do not have any proper subset except for $\varnothing$. Thus, according to Eq. (3), we have $belief(\{favorable\}) = m(\{favorable\})$ and $belief(\{unfavorable\}) = m(\{unfavorable\})$.

The Dempster's rule of combination is a critical concept to the original idea of the D-S theory. Given two masses m_1 and m_2 for a hypothesis, the combination rule computes a *joint* mass for the two pieces of evidence under the same hypothesis, which can be calculated as follows,

$$m_{1,2}(\varnothing) = 0 \tag{4}$$

$$m_{1,2}(A) = m_1(A) \oplus m_2(A) = \frac{1}{1-K} \sum_{B \cap C = A \neq \varnothing} m_1(B) m_2(C) \tag{5}$$

where $K = \sum_{B \cap C = \varnothing} m_1(B) m_2(C)$.

Eq. (4) says that the combined mass for the empty set $\varnothing$ is zero. In Eq. (5), *K* represents the measure of the amount of conflicts between the two mass sets. This is determined by summing the masses of any pair of sets *B* and *C*, where $B \subseteq \Theta$, $C \subseteq \Theta$, and the intersection of them is empty. Note that in Eq. (5), 1-*K* is used as a normalization factor that has the effect of ignoring conflicts between any pairs of states.

IV. A COST-EFFECTIVENESS ANALYSIS MODEL

A. A Conceptual Model

Our proposed formal cost-effectiveness analysis model for product evaluation in e-commerce can help customers verify the quality of a product, which is reflected by its effectiveness value based on the information collected from an e-commerce website, such as Amazon. We notice that Amazon offered a flexible e-commerce platform, which contains a large amount of useful information that can be used to evaluate the quality of a product. More specifically, not only a customer who has purchased a product online is allowed to give a review star rating as well as review comments to the product, but also his review can be further rated by other customers. Due to the page size limitation, for most of the products, the review information has to be distributed across multiple web pages, which are typically ignored by most users except for the first few pages. In order to support automatic analysis of such useful information for decision making on online purchasing, we treat all reviews and their associated properties as evidence that supports a product as either favorable or unfavorable, and derive our cost-effectiveness analysis model. The conceptual model for cost-effectiveness analysis in e-commerce can be formally defined as a 3-tuple (*P*, *Bel*, *MinC*), where

1. $P = \{p_1, p_2, \ldots, p_n\}$ is a set of product alternatives to be evaluated and ranked, which should have very similar functionality and are within the same price range;
2. *Bel*: $P \rightarrow [0, 1]$ is a belief function employed in our model. Each product alternative has a degree of belief quantifying that the product is worth buying or not;
3. *MinC*: $P \rightarrow R^+$ is a cost function that maps a product alternative to its minimal price, defined as a positive real number. Note that for a particular product *p*, we can calculate its cost-effectiveness value using *Bel*(*p*) and *MinC*(*p*), which can then be used to rank the product alternatives in *P*.

Each product $p \in P$ can be further formally defined as a 6-tuple (*REV*, *S*, *PROP*, *Rel*, *EV*, *M*), where

1. $REV = \{r_1, r_2, \ldots, r_n\}$ is a set of product reviews for product *p*, given by different reviewers;
2. *S*: $REV \rightarrow \{0.2, 0.4, 0.6, 0.8, 1\}$ is the star ranking function for product reviews, where the star rankings of 1 to 5 has been normalized to a value between [0.2, 1];
3. $PROP = \{pr_1, pr_2, \ldots, pr_k\}$ is a set of review properties, which contribute to calculating the reliability of each review;
4. *Rel*: $REV \rightarrow [0, 1]$ is a reliability function for product reviews, which represents the importance and accuracy of each review;
5. $EV = \{ev_1, ev_2, \ldots, ev_l\}$ is a set of evidence used to justify a product as either favorable or unfavorable;
6. $M = \{m: EV \rightarrow [0, 1]\}$ is a set of mass assignment functions, which quantify and assess each piece of evidence into a mass that supports a product as either favorable or unfavorable.

In our research, we classify all available reviews for a product *p* into two groups: 1) a supportive group with a set of positive reviews that support the product as a favorable one (i.e., worth buying); and 2) a non-supportive group with a set of negative reviews that do not support the product as a favorable one (i.e., not worth buying). For example, when the 5-star rating mechanism is used, we would consider a review with 4 or 5 stars as a positive review; while a review with 1, 2, or 3 stars as a negative one. Furthermore, we consider the number of reviews in each group as separate independent pieces of evidence. Thus, we have four pieces of evidence in total that can be used to justify a product is either favorable or unfavorable. The four pieces of evidence are denoted as {*PR*, *NP*, *NR*, *NN*}, where *PR* is a set of positive reviews, *NP* is the number of positive reviews, *NR* is a set of negative reviews, and *NN* is the number of negative reviews. Note that since each group has two pieces of evidence to support its committed hypotheses, we first combine the evidence in each group separately (i.e., *PR* and *NP* for positive reviews, and *NR* and *NN* for negative reviews, respectively), then the mass values for the two groups of reviews are combined again to calculate the belief values about the product.

Fig. 1 shows the framework for processing the review data of a particular product. Once the review ratings and their associated review properties are extracted, the reliability of each review can be calculated. We classify the review data into two groups of evidence, namely the supportive group and the non-supportive group. The two pieces of evidence in the same group are combined using Dempster's rule of combination, and derive the two sets of mass values for supportive evidence and non-supportive evidence, respectively. We consider the two pieces of combined evidence as conflicting evidence, and use Dempster's rule of combination again to calculate the *belief values* about the product, and further derive its *effectiveness* as defined in Section IV.C of this paper. Note that our approach does *not* involve analyzing the actual review comments, but it is envisioned as our more ambitious future research direction.

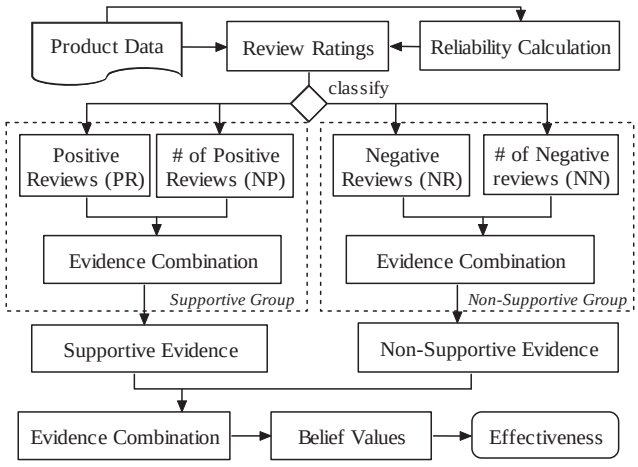


Figure 1. A framework for processing product review data

B. Calculation of Basic Mass Assignments

The first step to calculate the basic mass assignmnets for product reviews is to compute the reliability of each review. The reliability of a review is determined by a number of factors, called *review properties*, which indicate the trustworthiness of a review. We now use Amazon website as

an example to demonstrate how to calculate the reliability of a review. Fig. 2 shows a snapshot of a review. As shown in the figure, the Amazon's website allows a review to be voted as a *helpful* review by its customers. Such information is enclosed in a box, which is labeled as *Helpful Rate* (pr_1) in the figure. The more votes as helpful reviews over the total number of votes, the more reliabile the review is. Thus, we consider the number of helpful votes and the number of total votes as major factors to calculate the reliability of each review. Let the number of total votes be *total_votes* and the number of helpful votes be *helpful_votes*, we calculate the *Helpful Rate* (pr_1) of the review as *help_votes* / *total_votes*. Note that if *total_votes* equals 0, we set $pr_1 = 0$.

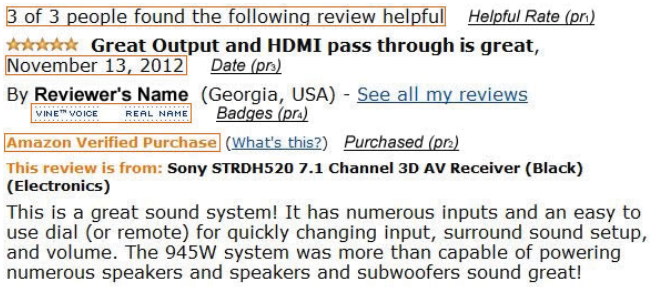


Figure 2. A snapshot of a product review with review properties

We further consider four additional review properties as factors that contribute to computing the reliability of a review. Those factors are *Purchased* (pr_2), *Date* (pr_3), and *Badges* (pr_4) as shown in Fig. 2, and *Top Reviewer Ranking* (pr_5) as shown in Fig. 3. Note that the properties pr_2, pr_4, and pr_5 are actually properties of the reviewer who wrote the review. Since the reliability of a review is closely related to the reliability of the person who wrote the review, in this paper, we also call them review properties.

Figure 3. A snapshot of a reviewer's personal profile

We now provide the detailed descriptions of the review properties pr_2 to pr_5 as follows.

Purchased (pr_2) is a label of a reviewer indicating that the e-commerce company has verified the reviewer has purchased the product. A reviewer who has bought and had a real experience with the product can surely write more reliable reviews than those who do not.

Date (pr_3) is the date when the review was posted. For simplicity, we convert the date into the number of months that have passed since the review was posted. The more recent a review was written, the more useful and reliable the review is.

Badges (pr_4) is the number of badges that a reviewer has been awarded. At Amazon website, there are totally nine types of badges. For example, the REAL NAME badge indicates that the customer used his real name from his credit card. The more badges a reviewer owns, the better review history the reviewer should have. Note that in Fig. 2, the reviewer has a REAL NAME badge, but for privacy purpose, we have removed the reviewer's real name from the figure.

Top Reviewer Ranking (pr_5) of a reviewer reflects the opinions of other customers about the reviewer. A reviewer's *Top Reviewer Ranking* is determined by the overall helpfulness of the reviewer's reviews, factoring in the number of reviews the reviewer has written.

Before calculating the reliability of a review, we first normalize the property values into ones in the range [0, 1]. Table 1 shows the value ranges and the normalized values for the five review properties pr_1 to pr_5.

Table 1. Review properties used to determine review reliability

Property	Description	Value Range	Normalized Value
pr_1	Helpful Rate	[0, 1]	helpful_votes / total_votes 0 (if total_votes = 0)
pr_2	Purchased	{0, 1}	0 → 0 (not purchased) 1 → 1.0 (purchased)
pr_3	Date	[0, +∞)	0~3 months → 1.0 3~6 months → 0.7 6~12months → 0.4 > 1year → 0.1
pr_4	Badges	[0, 9]	no_of_badges / 9
pr_5	Top Reviewer Ranking	[1, +∞)	< 1,000 → 1.0 1,000~10,000 → 0.7 10,000~100,000 → 0.4 >100,000 → 0.1

The reliability *Rel*(r) of a product review r can be calculated as in Eq. (6).

$$Rel(r) = w_1 \times pr_1 + w_2 \times (pr_2 + pr_3 + pr_4 + pr_5) \quad (6)$$

where the weights w_1 and w_2 indicate the importance of the review property pr_1 and the other four review properties pr_2 to pr_5, respectively. Since the review property *Helpful Rate* represents the most important factor to determine the reliability of the review, based on our experience, we set w_1 = 0.6 and w_2 = 0.1, which leads to a reliability in the range [0, 1].

To calculate the *BMA*s for both of the positive and negative reviews, we compute the weighted average star (*WAS*) for the two groups of reviews as in Eq. (7).

$$WAS = \frac{S(r_1) \times Rel(r_1) + S(r_2) \times Rel(r_2) + ... + S(r_k) \times Rel(r_k)}{k} \quad (7)$$

where $S(r)$ and *Rel* (r) are the normalized star ranking and the calculated review reliability for review r, respectively, and k is the total number of reviews in the corresponding group.

Let F = {*favorable*} and ~F = {*unfavorable*}, we have U = $F \cup$~F = {*favorable*, *unfavorable*}. Let WAS_{PR} and WAS_{NR} be the *WAS* values for the groups of positive and negative reviews, respectively, we can calculate the *BMA*s for both groups as in Eqs. (8-9). Note that $m_{PR}(U)$ and $m_{NR}(U)$ refer to the mass values of uncertainty for positive reviews and negative reviews, respectively.

$$\begin{cases} m_{PR}(F) = WAS_{PR} \\ m_{PR}(\sim F) = 0 \\ m_{PR}(U) = 1 - WAS_{PR} \end{cases} \quad (8) \qquad \begin{cases} m_{NR}(F) = 0 \\ m_{NR}(\sim F) = WAS_{NR} \\ m_{NR}(U) = 1 - WAS_{NR} \end{cases} \quad (9)$$

Since the number of reviews can serve as a good indicator of the quality and popularity of a product, we consider the number of reviews in each group of reviews as independent evidence. To assess how the number of reviews has an impact on either supporting that the product is a favorable one or an unfavorable one, we first identify the maximum numbers of reviews in both groups (denoted as N_{NP_max} and N_{NN_max}) among the set of product alternatives P. Then we compare the number of reviews in a given group with the corresponding maximum number of reviews in order to assess its impact on the belief value of the product. We realize that some popular product may have an extremely large number of reviews comparing to others. In this case, the result will be dominated by such maximum number. To avoid this situation, we use logarithm function to narrow down the gaps between the numbers of reviews for different product alternatives. We use the following simple example to illustrate the basic idea. Suppose N_{NP_max} = 2,000 among a set of product alternatives Ω, and for a certain product $\alpha \in \Omega$, n_{NP} = 100. When we compare n_{NP} with N_{NP_max}, the impact of n_{NP} becomes very small, although 100 positive reviews represent a considerable amount of reviews. Now if we try to compare $\log_{10} n_{NP}$ with $\log_{10} N_{NP\_max}$, the gap between them can be significantly narrowed down, and the number of positive reviews, n_{NP} = 100 in this case, can be properly taken into account as a piece of evidence to support product α as a favorable one.

The *BMA*s for the number of reviews in the supportive group can be calculated as in Eqs. (10-12), where $N_{NP\_max} > 0$.

$$m_{NP}(F) = \begin{cases} \dfrac{2\times\log_{10}(n_{NP}+1)}{\log_{10}(N_{NP\_\max}+1)} - 1 & \text{if } \log_{10}(n_{NP}+1) \geq (\log_{10}(N_{NP\_\max}+1))/2 \\ 0 & \text{otherwise} \end{cases} \quad (10)$$

$$m_{NP}(\sim F) = \begin{cases} 1 - \dfrac{2\times\log_{10}(n_{NP}+1)}{\log_{10}(N_{NP\_\max}+1)} & \text{if } \log_{10}(n_{NP}+1) < (\log_{10}(N_{NP\_\max}+1))/2 \\ 0 & \text{otherwise} \end{cases} \quad (11)$$

$$m_{NP}(U) = 1 - m_{NP}(F) - m_{NP}(\sim F) \quad (12)$$

Note that in order to deal with the special case when n_{NP} = 1, in the above equations, we replace n_{NP} and N_{NP_max} with (n_{NP}+1) and (N_{NP_max}+1), respectively. When $\log_{10}(n_{NP}+1) \geq (\log_{10}(N_{NP\_max}+1))/2$, we consider it as a piece of evidence that supports the product is a favorable one. As special cases, when $n_{NP} = N_{NP\_max}$, the mass equals 1, which means the evidence fully support that the product is a favorable one. When $\log_{10}(n_{NP}+1) = (\log_{10}(N_{NP\_max}+1))/2$, the mass equals 0, which means we do not consider the insufficient positive reviews as a piece of evidence to support the product as a favorable one. On the other hand, when $\log_{10}(n_{NP}+1) < (\log_{10}(N_{NP\_max}+1))/2$, we consider it as a piece of evidence that supports the product is an unfavorable one rather than a favorable one due to a lack of positive reviews. As a special case, when n_{NP} = 0, the mass equals 1. This means that when there is no positive reviews, the product must be a low-quality one, which should not be suggested to customers for purchasing.

Similarly, the *BMA*s for the number of reviews in the non-supportive group can be calculated as in Eqs. (13-15), where $N_{NN\_max} > 0$.

$$m_{NN}(F) = 0 \quad (13)$$

$$m_{NN}(\sim F) = \frac{\log_{10}(n_{NN}+1)}{\log_{10}(N_{NN\_\max}+1)} \quad (14)$$

$$m_{NN}(U) = 1 - m_{NN}(\sim F) \quad (15)$$

Note that different from the positive reviews, the negative reviews are always considered as a piece of evidence that supports the product is an unfavorable one. As two special cases, when $n_{NN} = N_{NN\_max}$, the mass equals 1, which means the evidence fully support the product is an unfavorable one; on the other hand, when n_{NN} = 0, the mass value equals 0, which means that there is no evidence (in terms of negative reviews) to show that the product is an unfavorable one.

C. Combination of Evidence

Once the basic mass assignments for each piece of evidence are calculated, they can be combined in a systematic manner to provide a more complete assessment on product quality by reducing the uncertainty involved in individual evidence. The evidence fusion procedure is carried out using the Dempster's rule of combination. As shown in Fig. 1, we first combine the evidence of positive reviews (*PR*) and the number of positive reviews (*NP*) into masses m_{SG} for the supportive group, and the evidence of negative reviews (*NR*) and the number of negative reviews (*NN*) into masses m_{NSG} for the non-supportive group. The corresponding rules of combining evidence for F and $\sim F$ are listed as in Eqs. (16-18) and Eqs. (19-21) for the supportive group and the non-supportive group, respectively.

$$m_{SG}(F) = m_{PR}(F) \oplus m_{NP}(F) \quad (16)$$

$$m_{SG}(\sim F) = m_{PR}(\sim F) \oplus m_{NP}(\sim F) \quad (17)$$

$$m_{SG}(U) = m_{PR}(U) \oplus m_{NP}(U) \quad (18)$$

$$m_{NSG}(F) = m_{NR}(F) \oplus m_{NN}(F) \quad (19)$$

$$m_{NSG}(\sim F) = m_{NR}(\sim F) \oplus m_{NN}(\sim F) \quad (20)$$

$$m_{NSG}(U) = m_{NR}(U) \oplus m_{NN}(U) \quad (21)$$

When the masses for both of the supportive group and non-supportive group have been calculated, we can use Dempster's rule of combination again to combine them into masses $m_{PRODUCT}$ for the product as in Eqs. (22-24).

$$m_{PRODUCT}(F) = m_{SG}(F) \oplus m_{NSG}(F) \quad (22)$$

$$m_{PRODUCT}(\sim F) = m_{SG}(\sim F) \oplus m_{NSG}(\sim F) \quad (23)$$

$$m_{PRODUCT}(U) = m_{SG}(U) \oplus m_{NSG}(U) \quad (24)$$

According to Eq. (3), the belief values for the product hypotheses can be calculated as in the following Eqs. (25-27).

$$belief(F) = m(F) \quad (25)$$

$$belief(\sim F) = m(\sim F) \quad (26)$$

$$belief(U) = m(U) \quad (27)$$

We now use an example to show how the masses for combined evidence can be calculated. Suppose we want to calculate the mass values for the supportive group. According to Eq. (5), we can calculate $m_{SG}(F)$, $m_{SG}(\sim F)$ and $m_{SG}(U)$ as in Eqs. (28-30).

$$m_{SG}(F) = m_{PR}(F) \oplus m_{NP}(F) = \frac{m_{PR}(F)\times m_{NP}(F) + m_{PR}(F)\times m_{NP}(U) + m_{PR}(U)\times m_{NP}(F)}{1-K} \quad (28)$$

$$m_{SG}(\sim F) = m_{PR}(\sim F) \oplus m_{NP}(\sim F) = \frac{m_{PR}(\sim F)\times m_{NP}(\sim F) + m_{PR}(\sim F)\times m_{NP}(U) + m_{PR}(U)\times m_{NP}(\sim F)}{1-K} \quad (29)$$

$$m_{SG}(U) = m_{PR}(U) \oplus m_{NP}(U) = \frac{m_{PR}(U)\times m_{NP}(U)}{1-K} \quad (30)$$

where $K = m_{PR}(F)\times m_{NP}(\sim F) + m_{PR}(\sim F)\times m_{NP}(F)$

Note that since $U \cap F = F$ and $U \cap \sim F = \sim F$, we have $U \cap F \neq \varnothing$ and $U \cap \sim F \neq \varnothing$ as in Eqs. (28-29).

The other two sets of mass values $\{m_{NSG}(F), m_{NSG}(\sim F), m_{NSG}(U)\}$ and $\{m_{PRODUCT}(F), m_{PRODUCT}(\sim F), m_{PRODUCT}(U)\}$ can be calculated in the same way.

According to Eqs. (25-27), the belief value that indicates a product α is a favorable one equals $m_\alpha(F)$, and the value of $m_\alpha(U)$ quantifies the uncertainty that the product α is both favorable and unfavorable. By taking the uncertainty into consideration, we calculate the effectiveness of product α by summing the belief value for a favorable product and 50% of the uncertainty value as in Eq. (31).

$$\begin{aligned} Effectiveness(\alpha) = Bel(\alpha) &= belief(F) + 0.5 \times beilef(U) \\ &= m_\alpha(F) + 0.5 \times m_\alpha(U) \end{aligned} \quad (31)$$

By further taking the price factor into consideration, we can calculate the cost-effectiveness value (i.e., *E/C Ratio*) of product α as in Eq. (32).

$$E/C\ Ratio\ (\alpha) = Effectiveness(\alpha) / Cost(\alpha) \quad (32)$$

where $Cost(\alpha)$ is the normalized cost of product α. Let $P = \{p_1, p_2, \ldots, p_n\}$ be a set of product alternatives to be evaluated and ranked, which have similar functionality and are within the same price range. For $\forall\ \alpha \in P$, $Cost(\alpha)$ can be calculated as in Eq. (33).

$$Cost(\alpha) = MinC(\alpha) / Max(MinC(p_1), MinC(p_2), ..., MinC(p_n)) \quad (33)$$

where $MinC(\alpha)$ is the cost function (defined in Section IV.A) that maps product α to its lowest price offered by one of the online sellers. With the *E/C Ratio* for each product in set *P*, we can rank the product alternatives, and provide users useful insights about the products in online shopping.

V. Case Study

In this section, we demonstrate how our D-S theory based analysis model can be used to analyze data sets collected from Amazon. We use a case study to show how our analysis model can provide more reliable and accurate results than the typical product ranking based on average star ratings (*ASR*).

A. Data Collection

The data used in our case study was collected from recent product records at the Amazon website, but note that the product data such as star ratings, minimal price, and the number of reviews may have changed by the time of this publication. The product review information as well as the reviewer's profile information used in this case study (as demonstrated in Fig. 2 and Fig. 3) can be accessed directly from the product web pages. To ease our data collection task, we developed a Java program that can automatically collect the needed data items from the product web pages, and used them as inputs to our cost-effectiveness analysis model.

Table 2 gives a few examples of our collected raw data, where each row contains the star rating as well as the five review properties. The normalized values for the star rating and the review properties are shown inside the parentheses along with the raw data, which can be calculated according to the normalization rules described in Section IV.B as well as the conversion rules defined in Table 1.

Table 2. Examples of collected raw data and the normalized values

Star Rating	Helpful Votes / Total Votes	Purchased	Date	Badges	Top Ranking
5(1)	118/124(0.952)	1(1)	17(0.1)	1(0.11)	63,027(0.4)
5(1)	83/86(0.965)	1(1)	5(0.7)	2(0.22)	556(1.0)
2(0.4)	71/79(0.899)	0(0)	16(0.1)	0(0.00)	81,258(0.4)
4(0.8)	26/27(0.963)	1(1)	11(0.4)	3(0.33)	292,053(0.1)
3(0.6)	98/116(0.845)	1(1)	16(0.1)	0(0.00)	458,571(0.1)

Note that besides the data items listed in Table 2, we also need to collect additional evidence, such as the number of reviews in each category (positive reviews or negative reviews) and the prices of the product offered by different online sellers, which are all required in our analysis approach.

B. Case Study: Audio/Video Receiver

In this case study, we collected 10 A/V receiver products in the price range $200~$300, which are different in brands, series, star ratings and the number of reviews, but having their average star ratings at least 4. Table 3 lists the 10 products along with some raw data and the analysis results. Among the raw data, "*ASR*" refers to the average star rating of a product that is posted at its corresponding product page; "*# of Reviews*" is the total number of reviews including both positive (4 and 5 stars) and negative (1, 2, and 3 stars) reviews; and "*Price*" refers to the minimal price of a product that is offered by one of the online sellers. As shown in Table 3, the 10 A/V receiver products are sorted according to *ASR*. Based on *ASR*, the first three product alternatives (No.1-3) look like the best choices for purchasing. By further looking into the number of reviews and prices, a customer may select one out of the three options accordingly (e.g., if the customer does not care about the price too much, he may choose product No. 2 for purchasing).

Now with our analysis model, we can calculate the values of *Effectiveness* and the *E/C Ratio* for all 10 product alternatives, which are listed at the last two columns of Table 3. Since the effectiveness value quantifies the quality level of the products, a customer who cares only about quality may select products with the highest effectiveness values. The top three choices are No. 8 with *Effectiveness* 0.845, No. 3 with *Effectiveness* 0.789, and No. 6 with *Effectiveness* 0.768. On the other hand, if the customer cares about both the quality and cost, he may select products with highest *E/C Ratio* values. In this case, the top three choices are No. 8 with *E/C Ratio* 1.056,

No. 10 with *E/C Ratio* 1.032, and No. 3 with *E/C Ratio* 0.913. Note that the ranking results calculated using our analysis model are different from the ranking results based on *ASR*, but our ranking results are more accurate and reliable because our model considers more evidential information before the ranking results are calculated.

Table 3. Product information of ten A/V receivers and the analysis results

No.	ASR	# of Reviews	Price	Product & Brand	Effective-ness	E/C Ratio
1	**5.0**	1	199.99	Yamaha HTR-3064	0.609	0.761
2	**4.5**	101	240.81	Harman Kardon HK 3390	0.767	0.796
3	**4.3**	73	215.99	Onkyo HT-S3500	**0.789**	**0.913**
4	4.3	62	228	Onkyo CS-445	0.642	0.704
5	4.2	18	199.99	Onkyo TX-SR313	0.692	0.865
6	4.2	57	269	Onkyo TX-8050	**0.768**	0.714
7	4.2	66	247.99	Yamaha RX-V471BL	0.764	0.770
8	4.1	153	200	Sony STRDH 520	**0.845**	**1.056**
9	4.1	48	249.95	Yamaha RX-V373	0.758	0.758
10	4.1	81	179.95	Yamaha RX-v371BL	0.743	**1.032**

To verify our analysis results, we look into the raw data collected for our case study. For product No. 1, although it has the highest *ASR* (5.0), but since it has only one positive review, its effectiveness value becomes not high enough comparing to the other product alternatives. For product No. 2, although it has a very good *ASR* as well as a decent number of product reviews, when looking into the review properties of those reviews, we found that many of the reviews are not reliable (e.g., having a very low *Helpful Rate* or even no *Helpful* votes). On the other hand, product No. 8 has relative lower *ASR*; however, it has the highest number of reviews, and most of its reviews are ones with high *Helpful Rates*. Consequently, product No. 8 has the highest *effectiveness* value among the 10 product alternatives.

VI. Conclusions and Future Work

In this paper, we introduce a formal cost-effectiveness analysis model for product evaluation in e-commerce, which was developed using the D-S theory. In our approach, we consider the product reviews as well as their review properties as pieces of evidence to justify whether a product is a favorable one or not. Product data from e-commerce websites such as Amazon is quantified and evaluated using our formal approach. By applying Dempster's rule of combination, we can combine difference pieces of evidence to derive more reliable belief values about the hypotheses on the quality of the product. Due to the nature of the D-S theory, our analysis model can handle uncertain information and reduce the degree of uncertainty appropriately. Thus, our approach produces more reliable and accurate results than conventional ranking mechanisms such as the one based on average star ratings. By ranking the product alternatives properly, our approach can be very effective in assisting customers to evaluate various products, and make purchase decisions on the most cost-effective ones.

In future research, we plan to develop a trustworthy e-commerce platform based on our formal cost-effectiveness analysis model. In the trustworthy e-commerce model, product reviews can be classified into more meaningful groups using data mining approaches as we did in our previous work [12]. The groups of product reviews can then be used as independent evidence for evidence combination using the D-S theory. With more evidence, our approach can significantly reduce the level of uncertainty, and lead to more accurate and reliable product ranking results. In addition, we will consider deploying our trustworthy e-commerce platform into cloud so that it would work in a flexible way and can be more conveniently accessed through the Internet. Finally, we plan to implement our approach on mobile e-commerce platforms such as tablets and smart phones, and provide customers with more friendly and flexible interfaces for mobile commerce.

References

[1] Amazon, *Managing Reviews*, Amazon.com Site Features, 2013. Retrieved on January 1, 2013 from http://www.amazon.com/gp/help/customer/display.html?nodeId=16465311

[2] G. Shafer, *A Mathematical Theory of Evidence*, Princeton University Press, 1976.

[3] F. Dong, S. M. Shatz, and H. Xu, "Reasoning Under Uncertainty for Shill Detection in Online Auctions Using Dempster-Shafer Theory," *International Journal of Software Engineering and Knowledge Engineering (IJSEKE)*, Vol. 20, No. 7, Nov. 2010, pp. 943-973.

[4] S. Panigrahi, A. Kundu, S. Sural, and A. K. Majunmdar, "Use of Dempster-Shafer Theory and Bayesian Inferencing for Fraud Detection in Mobile Communication Networks," *Lecture Notes in Computer Science*, Spring Berlin Heidelberg, Vol. 4586, 2007, pp. 446-460.

[5] J.-B. Yang and M. G. Singh, "An Evidential Reasoning Approach for Multiple-Attribute Decision Making with Uncertainty," *IEEE Transactions on Systems, Man, and Cybernetics (IEEE TSMC)*, Vol. 24, No. 1, Jan. 1994, pp. 1-17.

[6] G.-D. Li, D. Yamaguchi, and M. Nagai, "A Grey-Based Decision-Making Approach to the Supplier Selection Problem," *Mathematical and Computer Modelling*, Vol. 46, No. 3-4, 2007, pp573–581.

[7] A. Denguir-Rekik, G. Mauris, and J. Montmain, "Propagation of Uncertainty by the Possibility Theory in Choquet Integral-Based Decision Making: Application to an E-Commerce Website Choice Support," *IEEE Transactions on Instrumentation and Measurement*, Vol. 55, No. 3, June 2006, pp. 721-728.

[8] F. Herrera, E. Herrera-Viedma, and L. Martínez, "A Fusion Approach for Managing Multi-Granularity Linguistic Terms Sets in Decision Making," *Fuzzy Sets and Systems*, Vol. 114, No. 1, 2000, pp. 43-58.

[9] V. N. Huynh, Y. Nakamori, T. B. Ho, and T. Murai, "Multiple Attribute Decision Making Under Uncertainty: the Evidential Reasoning Approach Revisited," *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans*, Vol. 36, No. 4, 2006, pp. 804-822.

[10] Y. H. Cho and J. K. Kim, "Application of Web Usage Mining and Product Taxonomy to Collaborative Recommendations in E-Commerce," *Expert Systems with Applications*, Vol. 26, No. 3, 2004, pp. 234-246.

[11] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Analysis of Recommendation Algorithms for E-Commerce," *Proceedings of the 2nd ACM Conference on Electronic Commerce*, 2000, pp.158-167.

[12] B. J. Ford, H. Xu, and I. Valova, "A Real-Time Self-Adaptive Classifier for Identifying Suspicious Bidders in Online Auctions," *The Computer Journal (COMPJ)*, Vol. 56, No. 5, 2013, pp. 646-663.

On the Use of Bug and Predicate Signatures for Statistical Debugging

Yiwei Zhang (*Amazon*), Eric Lo (*Hong Kong Polytechnic University*), Ben Kao (*University of Hong Kong*)

Abstract

Recently, data mining techniques have been applied to mine software execution data in order to identify the program statements that are relevant to program bugs. While empirically effective, we observe that the effectiveness of such techniques can be improved by isolating the interferences between bugs using a "signature-based" approach. Our proposed approach is evaluated using real subject programs. Results show that we can pinpoint bugs precisely and support more complicated scenarios than existing work.

I. Introduction

Recently, data mining techniques have been developed to mine software execution data in order to identify the program statements that are relevant to program bugs [8], [5], [6], [1], [4], [3], [2]. Once branch of such data mining techniques is *predicate-based-bug-mining* (PBBM) — a program is first *instrumented* by injecting extra code that evaluates Boolean expressions (called *predicates*) at various program points (called *instrumentation sites*) [5]; upon termination of an execution of the instrumented program, a statistical report is generated that details how often an instrumentation site has been visited and how often a predicate has been evaluated true; various mining techniques are then devised to rank the predicates according to their estimated relevancy to bugs and return top-ranked predicates to developers as debugging hints.

Most PBBM techniques [5], [6], [1] are iterative. First, instrumented predicates are ranked according to their statistical relevancy to bugs. Top-rank predicates are considered good *predictors* of bugs and developers study those predicates manually to locate the bugs. If a bug is really found, it is fixed to produce a new program version. The new version of the program is tested again and a new statistical report is collected. In case the program still contains bugs, the preceding procedure is re-applied. We observe that their effectiveness can be further improved if we first assign the predicates to different clusters based on their *signatures*. Given a subject program G and a set of execution runs $\mathcal{R}$ on G, the signature of a predicate P, denoted by $Sig(P)$, is the set of runs in $\mathcal{R}$ in which P is observed true. Suppose program G contains a certain number of bugs. Let B be one of them. The signature of bug B, denoted by $Sig(B)$, is the set of *failed* runs in $\mathcal{R}$ (i.e., those failed test cases) because of the existence of bug B. Ideally, if predicate P is a perfect predictor of bug B, then for each execution run R_i in $\mathcal{R}$, predicate P should be observed true in R_i if and only if R_i is a failed run because of the existence of bug B. That is:

$$\forall R_i \in R: \quad (R_i \in Sig(P) \Leftrightarrow R_i \in Sig(B)) \\ \Leftrightarrow \quad (Sig(P) = Sig(B)) \tag{1}$$

Finding the perfect predicates can thus be done by inspecting predicate signatures and collect the predicates whose signatures match those of the bugs. There are two issues with this signature-based approach: (i) If there were only one single bug B in the program, $Sig(B)$ is simply the set of all failed runs in $\mathcal{R}$, and thus can be determined. However, if the program contains multiple bugs, the set of failed runs in $\mathcal{R}$ only gives the union of the bugs' signatures; the signature of each individual bug is unknown. (ii) Even if a bug's signature, say $Sig(B)$, is known, the perfect predicate P for bug B could be a complex predicate. A recent work [1] points out that *complex-predicate predictors* (predictors composed by conjunctions and disjunctions of simple predicates) can capture some kinds of bugs better than simple-predicate predictors. However, since a complex predicate is constructed from simple predicates by conjunctions and disjunctions, if there are n simple predicates, then there are about 2^n complex predicates considering conjunction alone. Therefore, searching the perfect predicate P (either simple or complex) is challenging.

In this paper, we propose a signature-based bug localization technique (SIGBOT) to solve these problems.

II. Background and Problem Definition

For a test suite of m test runs $R_1, R_2, \ldots, R_m$, the generated statistical report consists of m vectors in the form $\langle f_i | c_{i,1}, c_{i,2}, \ldots, c_{i,n} \rangle$. In the vector, f_i is the *failure-bit* which is set to '1' if R_i is a failed run (e.g., the program terminates abnormally or returns an incorrect result) and it is set to '0' if R_i is a *successful run*. $c_{i,j}$ is a *counter* that records the number of times that predicate P_j is evaluated true in R_i. For convenience, we say "P_j is *observed true*" if predicate P_j is evaluated to be true at least once during

This work is partly supported by the Research Grants Council of Hong Kong (GRF PolyU 525009, 521012, and HKU712712E.).

the execution of R_i; therefore, $c_{i,j} > 0$ implies that P_j is observed true in R_i. We say "P_j is *observed*" as long as predicate P_j is evaluated, no matter whether it is true or false. If P_j is never observed during the execution of R_i, we set $c_{i,j}$ as "–".

In LIBLIT05 [5], given a (simple) predicate P, two conditional probabilities are computed:

$Pr_1(P)$ = Pr (G fails | P is observed), and
$Pr_2(P)$ = Pr (G fails | P is observed `true`)

The difference of the two probabilities $\Delta(P) = Pr_2(P) - Pr_1(P)$ is taken as one of two measurements of how much the observation of P being true increases the likelihood that the subject program G fails. Intuitively, the larger the probability increment ($\Delta(P)$), the stronger is this lift of likelihood, and therefore the stronger is P's predictive power.

We observe that the notion of probability increment is distorted by the interactions of bugs in the program. For example, consider the program fragment:

```
...
b = a mod 2;//Predicate P:(b = 0) is instrumented here
c = 1/b;  // Bug B1 here
...
d = 1/0;  // Another bug B2 here
```

In this program, a predicate `(b = 0)` is instrumented at the program line `b = a mod 2;`. So, whenever that program line is executed during runtime, the program will also count how many times the predicate `(b = 0)` is evaluated to be true at that program point and write that information into the statistical report. In this program, if a is an even number, b will be 0 and the program fails because of the existence of bug B_1 (a division by 0). Otherwise, the program proceeds and eventually fails because of the existence of bug B_2. Let us assume that a is even in half of the runs. The predicate $P : (b = 0)$ is therefore observed true half of the time. Since the program always fails, we have $Pr_1(P) = Pr_2(P) = 1$ and thus $\Delta(P) = 0$. The zero probability increment implies that P is not bug-relevant. However, if bug B_2 is not there, we have $Pr_1(P) = 0.5$ and $Pr_2(P) = 1$ and so $\Delta(P) = 1 - 0.5 = 0.5$. The significant increase in the probability allows us to identify P as a bug-relevant predicate. This example shows that the statistical inference of LIBLIT05 could be affected by the interactions of bugs in the program. Specifically, assume the program has m bugs ($m > 1$), the statistical predictive power of each predicate calculated in the i-th iteration of LIBLIT05 is affected by the interactions of $m - i + 1$ bugs. For example, during the first iteration of LIBLIT05, the statistical predictive power of each predicate is affected by the interactions of m bugs. After the first bug is fixed (the program still contains $m - 1$ bugs), during the second iteration of LIBLIT05, the statistical predictive power of each predicate is still affected by the interactions of $m - 1$ bugs, and so on until the last round where only one bug is left. In fact, all iterative PBBM techniques [5], [6], [1] are affected by bug interactions (except in their last iteration, where only bug one is left).

ICML06 [8] is a PBBM technique that is resistant to bug interactions. It is not iterative but are based on co-clustering. Empirical results show that it is generally more successful in locating bugs because its statistical inferences are not influenced by bug interactions. However, it requires the full set of predicates as input. Since the number of complex predicates is exponentially large, it is not scalable to handle complex predicates.

In ISSTA07 [1], complex-predicates are introduced and empirical results show that they not only help developers in *locating* bugs, they also provide additional information for developers to *fix* the bugs. However, ISSTA07 is also an iterative approach and thus its statistical inference process may also be influenced by the presence of multiple bugs in each iteration (except for the last iteration).

Now, we give the formal problem definition: Given a faulty program G and its associated statistical report of instrumented simple predicates $\mathcal{P}$, for each bug B_i in G ($i \geq 1$), find a set of predicates $\mathcal{X}_{B_i}$, simple or complex, such that $\mathcal{X}_{B_i}$ leads the developers to locate and to fix B_i.

III. SIGBOT

SIGBOT is a tool that carries out statistical debugging through four steps:

A. Step 1/4: Extracting Pure Predicates

Definition 1 (Simple-predicate signature)**.** Given a simple predicate P, and a set of execution runs $\mathcal{R} = \{R_1, R_2, \ldots, R_m\}$, the **signature of the simple predicate** P **w.r.t.** $\mathcal{R}$ is the set of execution runs in R in which P is observed true. That is, $Sig(P)_{\mathcal{R}} = \{R_i \in \mathcal{R} | P \text{ is observed true in } R_i\}$.

Definition 2 (Complex-predicate signature)**.** Given a complex predicate $P = (P_1 \wedge \ldots \wedge P_c)$, which is the conjunction of c simple predicates, let the signatures of P_1, P_2, ..., P_c w.r.t. the set of execution runs $\mathcal{R}$ be $Sig(P_1)_{\mathcal{R}}$, ..., $Sig(P_c)_{\mathcal{R}}$, respectively. The **signature of the complex predicate** P **w.r.t.** $\mathcal{R}$ is the intersection of all its simple predicates' signatures, i.e., $Sig(P)_{\mathcal{R}} = \bigcap_{i=1}^{c} Sig(P_i)_{\mathcal{R}}$.

Definition 3 (Bug signature)**.** Given a bug B, and a set of execution runs $\mathcal{R} = \{R_1, R_2, \ldots, R_m\}$, the **signature of bug** B **w.r.t.** $\mathcal{R}$ is the set of failed execution runs in $\mathcal{R}$ when bug B is in the program. That is, $Sig(B)_{\mathcal{R}} = \{R_i \in \mathcal{R} | R_i \text{ fails when } B \text{ is in the program}\}$.

In the following, we will drop the subscript $\mathcal{R}$ whenever the context is clear. We define the *size of a predicate* as the

number of simple predicates that it is composed of. If a predicate P is a perfect predictor of a bug B, then ideally, $Sig(P) = Sig(B)$. Since a bug's signature contains only failed runs, the signature of a good predicate should also contain only failed runs.

Definition 4 (Purity)**.** A predicate is **pure** if all execution runs in its signature are failed runs.

The first step of SIGBOT is to extract pure predicates, simple or conjunctive complex, from the test report. Here we describe the procedure of collecting pure predicates.

Given the set of all simple predicates $\mathcal{P}$, there are $O(2^{|\mathcal{P}|})$ conjunctive complex predicates. Examining each of them for purity is infeasible. SIGBOT uses two heuristics to drastically reduce this exponential search space. The first heuristic uses the following monotonicity property of pure predicates: given two predicates P_1 and P_2, we say that P_2 *contains* P_1 if P_2 contains all the simple predicates that occur in P_1.

Property 1 (Monotonicity)**.** *If predicate P_2 contains a pure predicate P_1, then predicate P_2 is pure as well.*

Such a property holds because P_2 contains P_1 implies $Sig(P_2) \subseteq Sig(P_1)$, and so if P_1 is pure, then the signature of P_2 contains only failed runs and hence P_2 must be pure.

Let $\mathcal{Q}$ be a set of pure predicates (simple or complex) that SIGBOT has collected at some point during its predicate-collection procedure. SIGBOT begins by checking all simple predicates for their purity. The set $\mathcal{Q}$ is initialized to contain all of the pure simple predicates. Now consider a pure predicate $P_1 \in \mathcal{Q}$ and another predicate P_2 such that P_2 contains P_1. We note that P_2 must be pure because of the monotonicity property. Although P_2 is pure, it is generally *less useful* than P_1 in identifying a bug. The reason is that P_2 infers only a subset of the failed runs that are inferred by P_1 (because $Sig(P_2) \subseteq Sig(P_1)$). So, if P_1 predicts a bug B, then it is likely that P_2 characterizes only certain special cases of B. For that reason, we call P_2 a *sub-bug predicate*. As a result, given any predicate $P_1 \in \mathcal{Q}$, SIGBOT ignores all other predicates that contain P_1. This pruning strategy tremendously reduces the search space of pure predicates.

The second heuristic concerns the *coverage* of failed runs. Let the set of all failed runs in a test report be $\mathcal{F}$, i.e., $\mathcal{F} = \{R_i \in \mathcal{R} | R_i \text{ fails}\}$. We say that a failed run R_i *is covered by* a predicate P if R_i is in $Sig(P)$. Let $\mathcal{F}_\mathcal{Q}$ denote the set of failed runs that are covered by the predicates collected in $\mathcal{Q}$. Since only pure predicates are collected in $\mathcal{Q}$, we have

$$\mathcal{F}_\mathcal{Q} = \bigcup_{P \in \mathcal{Q}} Sig(P).$$

The objective of SIGBOT here is to find a $\mathcal{Q}$ such that all failed runs are covered, i.e., $\mathcal{F} = \mathcal{F}_\mathcal{Q}$. This mechanism is used because if such a $\mathcal{Q}$ is found, SIGBOT has collected enough predicates (in $\mathcal{Q}$) to infer all the failed cases (bugs) in the program. SIGBOT can thus terminate the predicate extraction step.

Now, if $\mathcal{F} \neq \mathcal{F}_\mathcal{Q}$, there exists a certain failed run R_i that is not covered by any predicates in $\mathcal{Q}$. We note that any predicate P_2 that contains a predicate $P_1 \in \mathcal{Q}$ cannot cover R_i. This phenomenon exists because by the monotonicity property, $Sig(P_2) \subseteq Sig(P_1)$ and so $R_i \notin Sig(P_1) \Rightarrow R_i \notin Sig(P_2)$. Hence, in order to cover R_i, SIGBOT must compose complex predicates out of those simple ones that have not already been collected in $\mathcal{Q}$. These simple predicates are given by the set $\mathcal{L} = \mathcal{P} - \mathcal{Q}$. SIGBOT then determines all *pure size-2 complex predicates* that are conjunctions of some simple predicates in $\mathcal{L}$. That is, SIGBOT determines $\mathcal{M} = \{P = P' \wedge P'' | P', P'' \in \mathcal{L}, \text{ and } P \text{ is pure}\}$. The predicates found in $\mathcal{M}$ are then added to the set $\mathcal{Q}$. If $\mathcal{F} - \mathcal{F}_\mathcal{Q}$ remains non-empty, SIGBOT determines all pure *size-3* predicates that are conjunctions of three predicates in $\mathcal{P} - \mathcal{Q}$, and so on. SIGBOT thus extracts pure complex predicates incrementally and of gradually increasing sizes.

B. Step 2/4: Clustering Predicates

The pure predicates collected in Step 1 may have great similarity among themselves. For example, consider the code fragment:

```
read (a);
b = 2*a;
...
```

We note that the predicates $P_1 : (a = 0)$ in line 2 and $P_2 : (b = 0)$ in line 2 are equivalent conditions and so they should have the same signatures. Moreover, if P_1 is a good predictor of a bug B, so is P_2. The second step of SIGBOT is to cluster the pure predicates collected in the set $\mathcal{Q}$ into clusters such that predicates with the same signature are put into the same cluster.

Definition 5 (Cluster signature)**.** Given a cluster C, the signature of C, denoted by $Sig(C)$, is the signature of any predicate in C. That is $Sig(C) = Sig(P)$, where P belongs to C.

Since all predicates in a cluster have an identical signature, *cluster signature* is well-defined. The clustering process serves the purpose of reducing a relatively large number of predicates (in $\mathcal{Q}$) into a relatively small number of distinct clusters (signatures). This mechanism allows the subsequent steps of SIGBOT to be executed efficiently.

C. Step 3/4: Evaluating and Filtering Predicates

The third step of SIGBOT measures the quality of the collected predicates. In particular, SIGBOT addresses the sub-bug problem and the "super-bug" problem in this step.

```
1  int discriminant(int a, int b, int c) {
2    return b * b - 4 * a * c;
3  }
4
5  void quadratic_equation_solution(int a, int b, int c){
6    int d = discriminant(a, b, c);
7    if (d < -1) {printf("no solution"); return;}
8    if(a != 0){
9      double x1 = (-1 * b + sqrt((double)d)) / (2 * a);
10     double x2 = (-1 * b - sqrt((double)d)) / (2 * a);
11     printf("x1=%.2f,x2=%.2f", x1, x2);
12   }
13   else
14     printf("x1=%.2f", -1 * c / b);
15 }
16
17 int main(int argc, char* argv[]){
18   ...
19 }
```

Figure 1. An Example Program

1) The sub-bug problem: We have reasoned that clusters' signatures characterize bugs. However, we note that a cluster (and its signature) may be predicting only part of a bug's behavior. To illustrate this situation, let us consider the program shown in Figure 1. The necessary condition for "triggering" the bug is $(d \geq -1) \wedge (d < 0)$. Suppose that we have instrumented the predicates $(d \geq -0.5)$ and $(d < -0.5)$. It is easy to see that both complex predicates $(d \geq -0.5) \wedge (d < 0)$ and $(d \geq -1) \wedge (d < -0.5)$ are pure predicates. However, each one of them is characterizing only about half of the failed cases caused by the bug. We call a cluster whose predicates characterize only part of a bug a *sub-bug cluster*.

In a sense, we can consider the problem of PBBM an example of classification analysis. In particular, we identify features (predicates) that can accurately predict the failures of the program. As in any classification technique, one has to deal with the *overfitting problem*, which occurs when the classification model is too specific such that it can be used to classify only a small number of cases. The problem of sub-bug clusters can be seen as an example of the overfitting problem: the predicates of a sub-bug cluster specify overly restrictive conditions that are applicable to only part of the failed cases of a bug.

To deal with the overfitting/sub-bug cluster problem, in SIGBOT, we can rank the clusters based on the sizes of their signatures, measured in their *generality*:

$$generality(C) = \frac{\log |Sig(C)|}{\log |\mathcal{F}|},$$

where $|Sig(C)|$ is the number of (failed) runs in $Sig(C)$ and $|\mathcal{F}|$ is the total number of failed runs in the test report. A larger generality value implies a cluster whose signature covers a larger portion of the failed cases — less likely be a sub-bug cluster.

2) The super-bug problem: A predicate is a *super-bug predicate* if it infers the failed cases of more than one bug. A super-bug predicate is thus not specific enough in helping developers to locate and to fix bugs. Interestingly, the super-bug problem can also occur even if we do not consider disjunctive complex predicates. To illustrate this case, consider the code fragment:

```
unsigned int a;
...
b = 100/a;  // Bug B1: a division by 0 if a=0
...
c = sqrt(a-2); // Bug B2: square root on a
               // negative number if a = 1
...
```

In this program, variable a is a non-negative integer. The program fails due to two individual bugs: bug B_1 occurs when a=0 and bug B_2 occurs when a=1. We see that the conjunctive complex predicate $(a \geq 0) \wedge (a \leq 1)$ is a pure but super-bug predicate because it infers the failed cases of both bugs B_1 and B_2. A cluster is called a *super-bug cluster* if its predicates are super-bug predicates.

To deal with the super-bug problem, we make the following observation. Consider a super-bug cluster C and its signature $Sig(C)$. If the predicates in C infer two distinct bugs B_1 and B_2, then we should be able to find two failed runs R_1 and R_2 in $Sig(C)$ such that R_1 and R_2 failed because of the existence of bugs B_1 and B_2, respectively. Now, if there is a predicate P_1 that predicts only B_1, then P_1 is more specific and is thus a better predicate than those in cluster C. We note that R_2 should not appear in the signature of P_1 because P_1 predicts only B_1 and R_2 is a failed case due to B_2. Similarly, if there is a predicate P_2 that only predicts B_2, then R_1 should not appear in $Sig(P_2)$. Let C_1 and C_2 be the clusters containing P_1 and P_2, respectively. If we consider the three clusters, namely, C, C_1 and C_2, we see that runs R_1 and R_2 co-occur in $Sig(C)$ but they do not co-occur in either $Sig(C_1)$ or $Sig(C_2)$. The *correlation* of the occurrences of R_1 and R_2 in various clusters' signatures is thus weak.

If C is not a super-bug cluster, then all of its predicates predict only one bug, say B. In this case, runs in $Sig(C)$, say R_1 and R_2, should all be the failed cases of B. Now, given another cluster C', if the predicates of C' predict B, R_1 and R_2 should co-occur in $Sig(C')$; otherwise, both R_1 and R_2 should not occur in $Sig(C')$. The *correlation* of the occurrences of R_1 and R_2 in various clusters' signatures is thus strong.

Based on the preceding observation, SIGBOT evaluates whether a cluster C is a super-bug cluster by considering the correlation of the runs in the signature of C: the higher the correlation, the less likely is C a super-bug cluster.

Given two runs R_i and R_j, let $\mathcal{C}_p(R_i, R_j)$ be the set of clusters such that a cluster C is in $\mathcal{C}_p(R_i, R_j)$ iff $Sig(C)$ contains both R_i and R_j. Also, we define $\mathcal{C}_n(R_i, R_j)$ to be the set of clusters such that $C \in \mathcal{C}_n(R_i, R_j)$ iff $Sig(C)$ contains only one of R_i and R_j. So, each cluster in $\mathcal{C}_p(R_i, R_j)$ is a *positive example* of R_i and R_j co-occurring while each cluster in $\mathcal{C}_n(R_i, R_j)$ is a negative

example.

Given a cluster C in $\mathcal{C}_p(R_i, R_j)$, if the signature of C contains only a small number of runs (i.e., $|Sig(C)|$ is small), then it is not likely that two random runs be grouped into $Sig(C)$ simply by chance. Hence, the fact that R_i and R_j do occur together in $Sig(C)$ provides a strong evidence that the two runs are highly correlated. In contrast, if $|Sig(C)|$ is large, having R_i and R_j occur together in $Sig(C)$ gives only a weaker evidence that the runs are correlated. Therefore, SIGBOT counts a *weighted* number of positive examples with the following formula:

$$positive(R_i, R_j) = \sum_{C \in \mathcal{C}_p(R_i,R_j)} \max\left\{\left(1 - \frac{|Sig(C)|}{|\mathcal{F}|}\right), \epsilon\right\} \quad (2)$$

That is, each $C \in \mathcal{C}_p(R_i, R_j)$ is weighted by $\max\{(1 - |Sig(C)|/|\mathcal{F}|), \epsilon\}$. Note that a smaller (resp. larger) $|Sig(C)|$ gives a larger (resp. smaller) weight $1 - |Sig(C)|/|\mathcal{F}|$. The ϵ in the formula is a small constant that serves as a sanity bound so that positive examples of very large clusters (with large $Sig(C)$) are not effectively ignored.

Negative examples in $\mathcal{C}_n(R_i, R_j)$ are also weighted. However, a *large* cluster $C \in \mathcal{C}_n(R_i, R_j)$ gives a *strong* evidence that R_i and R_j are negatively correlated. This evidence is strong because if one run, say R_i, is included in $Sig(C)$, then if $|Sig(C)|$ is large, it is *not likely* that the other run R_j is excluded from $Sig(C)$ by chance. So the fact that only one of the runs is included in a large $Sig(C)$ gives a strong evidence that the two runs are negatively correlated. SIGBOT counts a weighted number of negative examples by:

$$negative(R_i, R_j) = \sum_{C \in \mathcal{C}_n(R_i,R_j)} \max\left\{\frac{|Sig(C)|}{|\mathcal{F}|}, \epsilon\right\} \quad (3)$$

Again, a small ϵ is used as a sanity bound so that the contribution of very small clusters are not ignored. The correlation of a given pair of runs R_i, R_j is then computed by:

$$correlation(R_i, R_j) = \frac{positive(R_i, R_j)}{positive(R_i, R_j) + negative(R_i, R_j)} \quad (4)$$

To recap, our objective is to evaluate a cluster, say C, on how specific its predicates are in identifying a bug. Our approach is to measure the correlation of the runs in C's signature. We note that if a run R_i occurs only in $Sig(C)$ and not in any other clusters' signatures, then R_i is very specific to the bug associated with C. In this case, when we measure how specific cluster C is, run R_i should be given more weight. On the other hand, if R_i occurs in the signatures of many different clusters, then its appearance in a signature and thus its association with other runs does not come as a surprise. We should therefore give it a lesser weight in our measurement of specificity. To capture this idea, we compute an *inverse-cluster-frequency* (ICF) score for each run R_i by:

$$ICF(R_i) = \log \frac{|\mathcal{C}|}{|\{C_k \in \mathcal{C} | R_i \in Sig(C_k)\}|}, \quad (5)$$

where $\mathcal{C}$ is the set of all clusters. ICF is similar to the concept of *inverse-document-frequency* (IDF) used in traditional information retrieval (IR) systems, where frequently occurring keywords are generally given smaller weights in answering IR queries.

SIGBOT evaluates how specific a cluster C is in inferring a bug by aggregating the correlation of every pair runs R_i and R_j in C. These correlation values are weighted by the *significance* of the runs as captured by their ICF. The evaluation is summarized by the *specificity value*:

$$specificity(C) = \frac{\sum_{R_i,R_j \in Sig(C)} ICF(R_i) \cdot ICF(R_j) \cdot correlation(R_i, R_j)}{\binom{|Sig(C)|}{2}}. \quad (6)$$

Finally, SIGBOT evaluates the *quality* of a cluster C by computing the harmonic mean of C's generality score and specificity score.

3) Pruning low-quality clusters: After a quality score is assigned to each cluster, SIGBOT prunes the low-quality ones. Since the number of bugs in the program is unknown, the number of clusters to be retained is unknown. SIGBOT's approach is to retain as few clusters as possible as long as the complete set of failed runs ($\mathcal{F}$) is covered. SIGBOT prunes low-quality clusters incrementally and iteratively.

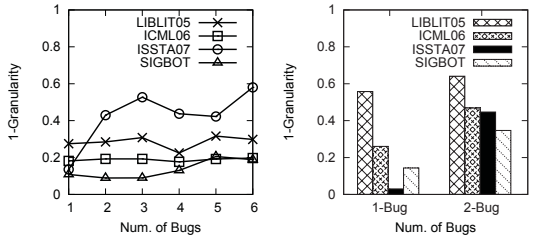
D. Step 4/4: Recommending Predicates

After the first three steps, SIGBOT has identified a small number of clusters, each is likely associated with a single bug of the subject program. The final step of SIGBOT is to recommend one predicate for each such cluster. Since the signature of a cluster is likely equal to that of a single bug, the bugs in the program no longer interact with each other and we can employ an existing PBBM technique to select the best predicate from each cluster. In our current implementation, we apply LIBLIT05 in this step.

IV. Empirical Study

In this study, we study the effectiveness of the techniques when they are applied to programs with multiple bugs. We use the program *Space* from the SIR Repository as the subject program. *Space* has 9,564 LOC. In addition to the original (bug-free) version, SIR provides 33 faulty versions and a test suite of 13,585 test cases. Each faulty version contains a single bug that has been discovered during the program's development.

To evaluate the various techniques under a multiple-bug environment, we carried out a bug-scale-up experiment. We synthesize q-bug versions of the subject program by arbitrarily picking q bugs from the pool of 33 bugs and manually inserting them into the original version. 10 q-bug versions are created and the effectiveness of the various



(a) Varying the number of bugs (b) Effects of complex bugs

Figure 2. Controlled experiments on *Space*

techniques on these 10 versions are recorded and averaged. In our experiment, we scale q from 1 to 6 (so a total of 60 program versions are generated).

Figure 2 shows the results our SIGBOT and the three state-of-the-art PBBM techniques LIBLIT05, ISSTA07, and ICML06, under the 1-Granularity distance metric [7]. This metric is used to measure the effectiveness of a PBBM technique by comparing the average distance between buggy statements and the statements that contain the predictor predicates in a graph-based representation of the program. Lower 1-Granularity distance means the technique is more effective.

Figure 2a shows that the effectiveness of SIGBOT is the highest among all of the comparing techniques. LIBLIT05 and ISSTA07 are less effective when the number of bugs in the program increases. SIGBOT and ICML06 have good effectiveness when the program has multiple bugs. We observe that the curves for them stay low and flat across the range $2 \leq q \leq 6$. On the contrary, ISSTA07 is much less stable over this range. This result shows that the effectiveness of the technique is very much affected by the interaction of multiple bugs in the program. In summary, when multiple bugs exist in the subject program, SIGBOT and ICML06 perform much better than iterative PBBM techniques. Between SIGBOT and ICML06, we see that SIGBOT generally has an edge over ICML06 as reflected by the fact that in many of the cases shown in Figure 2a, the curve for SIGBOT stays below that of ICML06. This advantage of SIGBOT is even more pronounced when the program contains bugs that are best predicted by complex predicates, which we are going to discuss next.

Though "complex bugs" exist, they happen less frequently than "simple bugs" and they are difficult to be synthesized. To evaluate SIGBOT on subject programs with "complex bugs", we generate program versions that contain only complex bugs. To achieve that goal, we analyze the pool of 33 bugs and retain only those complex ones. To decide whether a bug is complex or not, we use the following mechanism: as ISSTA07 considers complex bugs but LIBLIT05 does not, we apply both LIBLIT05 and ISSTA07 to each bug B in the bug pool. If the complex predicate (P_{ISSTA}) recommended by ISSTA07 is significantly better in quality than the simple predicate (P_{LIBLIT}) recommended by LIBLIT05, we classify B as a complex bug. In our experiment, by "significantly better", we require that $2 \times$ *1-granularity*$(P_{ISSTA}, B) \leq$ *1-granularity*(P_{LIBLIT}, B), i.e., the complex predicate is at least twice closer to the bug than the simple predicate. With this screening, 2 complex bugs are retained in the bug pool.

Figure 2b shows the results of the experiment. The figure contains two sets of bars: one set for the 1-bug version and another set for the 2-bug version of the program. Let us first look at the 1-bug scenario. From the figure, we see that ISSTA07, which returns complex predicates in an iterative fashion, is effective. SIGBOT is more effective than LIBLIT05 and ICML06. As we move to the 2-bug scenario, the performance of ISSTA07 degrades due to the fact that its statistical inference (in the first round) is imprecise due to the presence of multiple bugs. In this case, SIGBOT is the winner. When bugs are complex ones, ICML06, which returns simple predicates, is much less effective when compared with SIGBOT.

V. Conclusion

In this paper we propose SIGBOT, which is capable of suggesting good simple and complex predicates to predict bugs. Results show that SIGBOT performs well with the existence of multiple bugs.

References

[1] Arumuga Nainar et al. Statistical debugging using compound boolean predicates. In *ISSTA*, 2007.

[2] Chilimbi et al. Holmes: Effective statistical debugging via efficient path profiling. In *ICSE*, pages 34–44, 2009.

[3] Hsu et al. Rapid: Identifying bug signatures to support debugging activities. In *ASE*, pages 439–442, 2008.

[4] Jones et al. Debugging in parallel. In *ISSTA*, pages 16–26, 2007.

[5] Liblit et al. Scalable statistical bug isolation. In *SIGPLAN*, Chicago, Illinois, 2005.

[6] Liu et al. Statistical debugging: A hypothesis testing-based approach. *IEEE TSE*, 32(10):831–848, 2006.

[7] Zhang et al. Evaluation metric for multiple-bug localization with simple and complex predicates. In *APSEC*, 2012.

[8] Zheng et al. Statistical debugging: simultaneous identification of multiple bugs. In *ICML*, 2006.

Bacterio^{ORACLE}: An Oracle suggester tool

Pedro Reales Mateo and Macario Polo Usaola
Alarcos research group
University of Castilla-La Mancha
Ciudad Real, Spain
{pedro.reales, macario.polo}@uclm.es

***Abstract*— One of the main challenges during testing is to design good oracles. An oracle is one or more statements in a test that evaluate the behavior of the system. This paper presents a tool, Bacterio^{ORACLE}, based on mutation testing that helps testers to write good oracles. The tool generates oracle suggestions automatically analyzing information obtained from killed mutants. These suggestions are useful to discover errors and can be translated into executable oracles easily. Thus, this tool can be very useful during the testing process in order to create good oracles.**

Keywords- Bacterio; oracle; testing; mutation; tool; suggestions.

I. INTRODUCTION

Software testing is one of the most important tasks in software development process for ensuring quality. During testing, a tester exercises the software under tests in order to find faults. Simplifying the testing process, the tester has to specify some test requirements in order to design test cases that fulfill them. To define those test requirements, usually a testing technique (such as decisions coverage [1], input space partitioning [2] or mutation testing [3]) is used. These techniques specifies some conditions (i.e. to cover all the code statements of the system under test) that the tests must fulfill.

When the test requirements are stated, the tester has to design the tests cases. To do it there are some options. One option is to define a test as a set of interactions that must be performed by a user. Thus, the tester has to interact with the system to check its behavior. However, this option has a important disadvantage, the tests are not automatic and cannot be executed easily.

A better option is to use a XUnit framework, as for instance, JUnit [4] for Java or NUnit [5] for .Net. These frameworks provide*ff* a set of features that allow testers to automate tests, therefore tests can be executed automatically, which is very important to reduce costs of testing and increase its effectiveness (not only for testing during the development, but also for maintenance and regression testing).

Typically, to make an executable test script for object-oriented technologies, the tester has to define five things:

1. Set up the environment where the tests will be executed.
2. Create an instance of the element under tests.
3. Exercise one or more functionalities.
4. Check with an oracle test passes or fails.
5. Clean up any change in the environment done by the test.

Figure 1 shows an example of a JUnit test case written in Java. This test evaluates the *add* method of a supposed calculator. As Figure 1 shows, JUnit provides two special methods to set up and clean up the environment (*setUp* and *tearDown* methods). These two methods are executed before and after each test. Also, JUnit provides a set of methods to evaluate the operation of the calculator (*assert* methods).

```
public class TestCal extends TestCase {
    Calculator c;
    public void setUp(){
        //Set up the environment
        c = new Calculator();
    }
    public void testAdd(){
        int r = c.add(3, 9);
        assertTrue(r == 12);
    }
    public void tearDown(){
        //Clean up the environment
    }
}
```

Figure 1. JUnit test case example.

Two very important elements to make tests are:

1) To **determine the test data** of the method that will be executed. There are several techniques (for instance, Pair-Wise [6]) and tools (for instance, testooj [7] or CTWEb [8]) that helps testers and automate this task.

2) To **determine the oracles** of the tests. The oracles are the statements that check if the system works properly (*assert* methods of Figure 1). Unlike the previous task, it is very difficult to determine the oracles automatically since they depend on the domain of the system and the test data used. Therefore, in practice, oracles are designed by hand [9].

One of the main problems when oracles are defined is their *completion*. Usually, the elements checked after the execution of a method are not enough, i.e. some elements that should be checked are not checked because the tester does not consider them important or s/he just does not know they exist. Another important problem related with oracles is their *correction*. After the execution of a method, an oracle compares the value of an element with the supposed value that should have. Thus, if the element does not have the supposed value there is an error in the system. However, the supposed value must be provided by the tester that can make mistakes calculating it.

The tool presented in this paper, Bacterio^{ORACLE}, is related with the definition of oracles of the tests. This tool implements

a novel technique based on mutation testing that provides suggestions to define oracles automatically.

The automatically generated suggestions try to solve the *completion* problem, since they are based on killed mutants, and try to reduce the *correction* problem, since they provides some valuable information that should be checked by the tester.

The paper is organized as follows: Section 2 provides some background of mutation testing. Section 3 describes briefly how oracles suggestions are obtained. Section 4 presents Bacterio^{ORACLE} and its architecture. Section 5 shows how to use the oracle suggested. And Section 6 shows some conclusions and future works.

II. BACKGROUND ON MUTATION TESTING

Mutation testing is a very effective testing technique. With mutation testing, a tester creates copies of the system under tests with small syntactic changes (mutants) that can suppose an error. Then s/he design tests cases that has to be able to identify those syntactic changes (kill mutants). Thus, as more mutants are killed, better are the designed test cases.

Typically, a mutation analysis is composed of three tasks:

1) Mutant generation. The tester creates copies of the system and applies mutation operators (well-formed rules) to introduce the syntactic changes.

2) Mutant executions. After design the tests, the tester execute each test against the original system and each mutant and annotate what mutants are killed (A mutant is killed when a test obtains an output different from the original output).

3) Result analysis. The tester calculates the mutation score (Figure 2) that gives the quality of the executed test suite. Equivalent mutants (mutants which syntactic changes do not suppose a fault, thus there are not any test able to kill them) must be identified in order to calculate the mutation score.

$$MS(P,T) = \frac{K}{M - E}$$

,where: P = program under tests
T = test suite
K = Number of killed mutants
M = Total number of mutants
E = Number of equivalent mutants

Figure 2. Mutation score formula.

To perform a mutation analysis is costly task and automatic tools are a must. There are several available mutation tools that implement different mutation cost reduction techniques. These tools can reduce costs of mutation testing and help testers to perform mutation analysis.

The technique that implements the tool presented in this paper, Bacterio^{ORACLE}, is based on mutation testing. However, the implemented technique does not use mutation testing to evaluate the quality of a test suite, but to gather valuable information that can be useful to design oracles.

III. ORACLE SUGGESTIONS BASED ON MUTATION

As commented in section I, during the oracle definition in object orientation, the tester can commit two kinds of mistakes: the oracles can be uncompleted and they can contain errors. The technique implemented in Bacterio^{ORACLE} tries to solve automatically the first kind of mistakes: completeness.

Let's suppose that a tester has to test the method *add* of a supposed calculator (Figure 3) and s/he decides to exercise the method with values 3 and 9, thus s/he writes the test of Figure 4. At this point, oracles have to be defined, thus the tester writes an oracle to check that the result is 12 (the final test was shown in Figure 1).

```
public class Calculator {
    private int operationsDone;
    public int add(int a, int b){
        int r = a+b;
        this.operationsDone += 1;
        return r;
    }
    public int getOperationsDone(){
        return this.operationsDone;
    }
}
```

Figure 3. Calculator source.

```
public class TestCal extends TestCase {
    Calculator c;
    public void setUp(){
        //Set up the environment
        c = new Calculator();
    }
    public void testAdd(){
        int r = c.add(3, 9);
    }
    public void tearDown(){
        //Clean up the environment
    }
}
```

Figure 4. JUnit test case without oracles.

Although this oracle looks right and enough, the tester made an error because s/he did not check the field *operationsDone* of the class *Calculator*, which counts the total number of operations done (note the extreme simplicity of this example; in a more complex system this situation can easily go unnoticed). Therefore, if there is an error in the calculation of the *operationsDone* value, this error would not be discovered because the field is not checked.

To solve this completeness problem, Bacterio^{ORACLE} uses mutation testing to identify what fields and return values should be checked after the execution of a method.

Fields and return values are identified with killed mutants. Bacterio^{ORACLE} uses strong mutation, but, unlike other mutation tools like MuClipse [10], assertions are not used to consider a mutant killed (since mutation is used to design assertion and they are not available). Instead of assertions, Bacterio^{ORACLE} considers that the output of the system has two components: the return value and the states of the objects that remain in memory. Thus, a mutant is considered killed after the execution of a method if the return value or any field value of any object in memory is different from the values obtained when the method of the original version is executed.

With this approach, when a mutant is killed, Bacterio^{ORACLE} is able to determine what element (a field value or the return value) is responsible of the dead of the mutant. This information indicates that the identified element is important and must be checked.

Thereby, after the execution of a test against all the mutants, Bacterio^{ORACLE} can identify all the important fields and return values that must be checked after each executed method. Note that in this process only mutants that are killed by the executed test are interesting, so it is not necessary to identify equivalent mutants.

Figure 5 shows the code of two mutants of the calculator (Figure 3). The first mutant calculates wrong the addition and the second mutant calculates wrong the total executed operations. Applying the approach implemented in Bacterio^{ORACLE}, after the execution of the test of Figure 1 against the original system and the two mutants, Bacterio^{ORACLE} will identify that: (1) mutant one is killed because of the return value of the *add* method and (2) mutant two is killed because of the value of the *operationsDone* field. Therefore, Bacterio^{ORACLE} would suggest to the tester that it is important to evaluate the return value of *add* and the *operationsDone* field of the called object.

Version	Code

Mutant1	`public class Calculator {`
	`  private int operationsDone;`
	`  public int add(int a, int b){`
	`  int r = a-b;`
	`  this.operationsDone += 1;`
	`  return r;`
	`  }`
	`  public int getOperationsDone(){`
	`  return this.operationsDone;`
	`  }`
	`}`

Mutant2	`public class Calculator {`
	`  private int operationsDone;`
	`  public int add(int a, int b){`
	`  int r = a+b;`
	`  this.operationsDone -= 1;`
	`  return r;`
	`  }`
	`  public int getOperationsDone(){`
	`  return this.operationsDone;`
	`  }`
	`}`

Figure 5. Mutants of Calculator.

The second kind of error that can appear during the definition of oracles is related with the correction. To define an oracle it is necessary to compare an observed value with the supposed value. Thus, the supposed value must be calculated and provided by the tester in order to define the oracle. Unfortunately, most of the techniques to generate oracles of the literature suppose that the original system is correct. Therefore, the generated oracles must be checked by the tester in order to determine if the generated supposed value is correct.

The technique implemented in Bacterio^{ORACLE} has the same disadvantage. Bacterio^{ORACLE} not only identifies the fields that must be checked, but also provides values (values observed in execution of the original system) that can be easily used to write oracles. However, the provided values must be checked by the tester in order to determine whether they are correct. Thus, although the correction problem is not solved, the provided values joint to the fields that contain them help testers to define correct oracles.

So, after the execution of the test of Figure 1 against the original system and the two mutants (Figure 3 and Figure 5), Bacterio^{ORACLE} will identify that the result value of the method *add* should be 12 and the value of the *operationsDone* field should be 1. Now, the tester takes responsibility for determining if 12 and 1 are correct values. If they are correct, the oracle suggestions can be translated into executable oracles and it means that the test did not find any error in the system. If they are not correct, it means that the test found an error in the system.

IV. Bacterio^{ORACLE} Tool

Bacterio^{ORACLE} is an extension of the Java mutation testing tool Bacterio [11]. This extension is composed by four modules: first, a module to instrument test cases (this instrumentation provides a new special implementation of strong mutation based on return values and objects states); second, a module to gather killing information from the instrumented tests; third, a module to analyze the gathered killing information to make the oracles suggestions; and fourth, a module with a graphical user interface that shows the oracle suggestions. Figure 6 shows the architecture of Bacterio^{ORACLE} and relations between modules.

A. *Bacterio*

Bacterio [11] is a standalone application written in Java. It is designed to support mutation analysis at unit, integration and system levels for Java systems and test cases written in JUnit or UISpect4j format.

In order to reduce costs of mutation, Bacterio implements several cost reduction techniques: selective mutation [12], mutant sapling [13] and high order mutation [14] to reduce the number of mutants; byte code translations [15], mutant schema [16], MUSIC [17] and parallel execution [18] to reduce costs of generating and executing mutants; and strong mutation [3], weak mutation [19], flexible weak mutation [20] and functional qualification [21] to reduce costs of execution and allow different mutation types.

Bacterio automates the mutant generation and execution tasks and calculates the mutation score automatically, the tester only has to provide tests and identify equivalent mutants. For the equivalent mutant identification, Bacterio helps testers providing code comparisons of the original system and mutants.

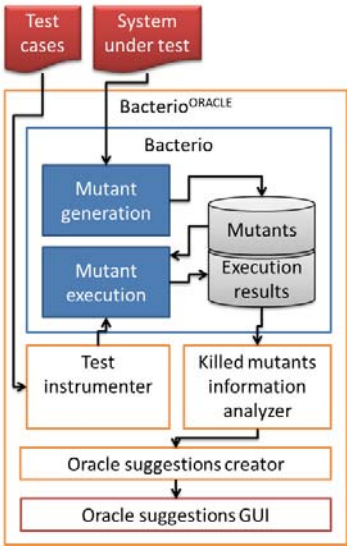


Figure 6. Bacterio^{ORACLE} architecture

Bacterio is used by Bacterio^{ORACLE} to create mutants and execute instrumented test cases against them in order to obtain information of the killed mutants.

B. Test Instrumenter

This module instruments test cases to store into the execution results all the information relative to return and field values. Later, this information is analyzed by other modules to create the oracle suggestions.

Under strong mutation, a mutant is considered killed if its outputs are different from the original outputs. With the instrumentation, the mutant is killed if the return value or any field value is different from the values of the original system. Thus, the instrumentation of the tests must provide enough information to compare all values of the original system and each mutant at the end of the execution. Moreover, this values comparison should be done after each method call in the tests, since the final oracle suggestions are obtained for each call. Therefore, the process of Figure 7 is followed to instrument a test case.

Figure 8 shows the instrumentation done in the test case of Figure 4. In the figure we can see that the called object and the return value are analyzed and store. The arguments are not analyzed because they are basic types, and therefore after the execution of the method they will have the same value. Finally, the Figure 8 shows that at the end of the test the field *c* of type *Calculator* is analyzed. This last analysis is necessary sometimes (in the example it is redundant) because there can be static methods without arguments (or with only basic types arguments) that do not return a value, thus they only change the internal state of fixtures.

An important issue during the execution of tests is how field values are analyzed and store (since a field value can be a complex value, i.e. an object instance). Bacterio^{ORACLE} creates from each analyzed value (basic or complex value) a code based in the EBNF grammar of Figure 9. This grammar describes a value as a simple value or as a set of fields or a matrix of values. This structure is able to translate into code an object which fields are also objects.

For each method call in the test case, after the call:

1- If the called method belongs to an object, analyze the field values of the object (note that static method does not bellow to an object).
2- If the called method returns a value, analyze the value (if the value is an object, analyze the field values of the object).
3- For each argument,
   If the argument is an object, analyze the field values of the argument.
4- For each local variable and each field of the test class, analyze the values (if the value is an object, store the field values of the object).

Figure 7. Process to instrument test cases.

```
public class TestCal extends TestCase {
    Calculator c;
    public void setUp(){
        //Set up the environment
        c = new Calculator();
    }
    public void testAdd(){
        int r = c.add(3, 9);
        analyze(c);
        analyze(r);

        analyze(c);
    }
    public void tearDown(){
        //Clean up the environment
    }
```

Figure 8. instrumented test case.

```
LOG     := NL  OA? ARGS? RE? EX? | NL  L?
NL      := Id_Method
ARGS    := ARG | ARG ARGS
OA      := # o  OCA %
ARG     := # a N  OCA %
RE      := #r  OCA% | #r FCA%
EX      := #e : id_ClassException FCA%
L       := #l : id_Local OCA% | #l : id_Local FCA%
OCA     := {IDO FS} | {m : IDO: OCAM} | {m : IDO:} |
                { IDO : n} | {m : IDO: FCAM} | {IDO}
OCAM    := (pos OCA) | (pos OCA) OCAM
FCAM    := (pos FCA) | (pos FCA) FCAM
FS      := F | FS F
F       := (idField  OCA) | (idField  FCA)
FCA     :=  : tipo : N
IDO     := id_Class_N
N       := number
```

Figure 9. EBNF grammar to store analyzed values.

Figure 10 shows an example of the code generated form the execution of the tests of Figure 8 against the original system.

```
tests.TestCal.testAdd$11#o{1_1(2:3:1)}%#r:3:12%
tests.TestCal.testAdd$4#l:5{6_1(7{1_1(2:3:1)})(8:9:10)}%
```

Figure 10. EBNF code generated from the original version.

C. Killed mutants information analyzer

When the EBNF codes are obtained from killed mutants, they are compared with the obtained code from the original system by the *killed mutants information analyzer* module. The goal of this process is to determine the elements that are responsible of the dead of each mutant. As commented in section III, this information indicates what elements (result values or field values) are important and, therefore, they should be analyzed. In these comparisons there are two possibilities:

1- The compared elements are two primitive types. In this case, if the observed values are different, the compared elements are stored jointly the original value, since they are responsible of the dead of the mutant.
2- The compared elements are two complex types. In this case two complex observed values are different if one of the next conditions is met: (1) if the types of the elements are different, (2) if one of the compared elements is null or (3) if any field is different.

Following this approach, Bacterio^{ORACLE} is able to store information like showed in TABLE I. The table contains the differences obtained from the EBNF codes produced from the test of Figure 8.

TABLE I. DIFFERENCES OBTAINED FROM KILLED MUTANTS

Mutant	Method	Element	Value
mutant1	*add*	return	12
mutant2	*add*	CalledObject.operationsDone	1

In the TABLE I shows that mutant1 (Figure 5) was killed because the return value of the method *add* was different from the return value of the method *add* of the original system. Also, the table shows that mutant 2 (Figure 5) was killed because the value of the field *operationsDone* of the object that executed the method *add* was different from the original value.

D. Oracle suggestions creator

When the killed information is analyzed and the elements responsible of the dead of the mutants are identified, the oracle suggestions are created by the *oracle suggestions creator* module.

In this last step, all the elements related with the same method are group and an oracle suggestion for each element is created, thus in case of several mutants are killed because of the same element, only one oracle suggestion is created from the element.

Also, the proper comparison method is used in each oracle suggestion. For example, if an element is an integer value, the operator "==" is used, however, if the element is a String value the method ".equals()" is used.

TABLE II shows the oracle suggestions obtained from TABLE I. We can see that Bacterio^{ORACLE} suggests that the tester should compare the return value with 12 and field *operationsDone* with 1.

TABLE II. ORACLE SUGGESTIONS.

Oracle suggestions
assertTrue (resultValue == 12);
assertTrue (CalledObject.operationsDone == 1);

E. Oracle suggestions GUI

This last module takes care to show oracle suggestions to the user with a graphical user interface. Figure 11 shows the interface showed with the oracles obtained in the TABLE II. This graphical user interface allows testers to select the executed tests and filter the showed oracles by types and by mutation operator.

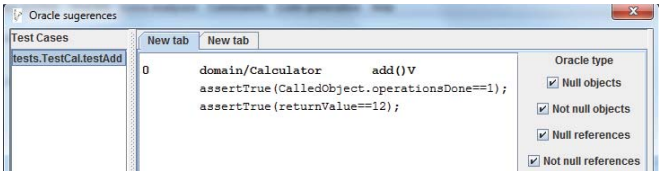


Figure 11. Grafical user interface to show oracle suggestions.

This final module is important to help tester to analyze the oracle suggestions, translate them into executable oracles and check the correctness of the suggestions.

V. USES OF THE ORACLE SUGGESTIONS

The information provided by Bacterio^{ORACLE} is useful for two tasks: (1) errors discovering, and (2) oracles writing.

A. Errors Discovering

As commented in section I, the goal of tests is to discover errors. However, without oracles, a test cannot discover errors (unless an exception is launched) because the behavior of the system is not checked.

The oracle suggestions provided by Bacterio^{ORACLE} contain information relative to the behavior of the system. Since the proposed comparisons are created from the observed values of the original system, they represent the actual behavior of the system. Thus, if any suggestion is wrong (i.e. if a suggestions says "return == 10", but the result should be 12), it means that there is an error in the system. Moreover, the tester has some information to find the bug because the suggestion indicates the element that has the wrong value (i.e. the return).

```
domain/Calculator       add()V
assertTrue(CalledObject.operationsDone==2);
assertTrue(returnValue==12);
```

Figure 12. Hypotetical wrong suggestion.

For example, Figure 12 shows hypothetical wrong suggestions for the test of Figure 8. The first suggestion says that the *operationsDone* is 2, however the number of operations done is one, therefore the field should contain 1. This indicates to the tester that there is an error in the system and this error is related with the field *operationsDone*.

B. *Writing executable oracles*

When the tester has evaluated all the suggestions and has determined that all of them are right (so the test did not discover any error in the system) the proposed suggestions can be translated into executable oracles.

For it, the tester only has to copy the suggestions in the correct place in the tests, and replace the name of each compared element by the variable name used in the tests and the access method when the element is a field. For example, the suggestions of Figure 11 can be translated into executable oracles as show the Figure 12.

```
public class TestCal extends TestCase {
    Calculator c;
    public void setUp(){
        //Set up the environment
        c = new Calculator();
    }
    public void testAdd(){
        int r = c.add(3, 9);
        assertTrue(c.getOperationsDone()==2);
        assertTrue(r==12);
    }
    public void tearDown(){
        //Clean up the environment
    }
}
```

Figure 13. Executable oracles obtained from suggestions.

VI. CONCLUSIONS

This paper presents a tool, Bacterio^{ORACLE}, which is able to generate automatically suggestions to define oracles of the tests. To obtain the suggestion, this tool uses mutation testing. From the killed, the tool obtains some important information relative to the elements that produces that the mutant die. Then, this information is translated into oracle suggestions.

Although the suggestions must be analyzed by the tester and translated into executable code, they are useful to discover errors and write oracles.

As a future work, we plan to modify the *oracle suggestions creator* module to create executable oracles directly and improve the *oracle suggestions GUI* module in order to provide better information to the tester.

The tool Bacterio^{ORACLE} can be found at http://alarcos.esi.uclm.es/per/preales/Pedro_Realess_Web_Page/Tools.html.

ACKNOWLEDGMENT

This work and the Bacterio^{ORACLE} tool are partially supported by the Spanish FPU program (AP2009-3058) and by the project GEODAS-BC (TIN2012-37493-C03-01) from MEC.

REFERENCES

[1] H. Zhu, P. A. V. Hall, and J. H. R. May, “Software unit test coverage and adequacy,” *ACM Comput. Surv.*, vol. 29, no. 4, pp. 366–427, Dec. 1997.

[2] P. Ammann and J. Offutt, *Introduction to software testing*. Cambridge University Press, Cambridge, UK, 2008.

[3] R. A. DeMillo, R. J. Lipton, and F. G. Sayward, “Hints on Test Data Selection: Help for the Practicing Programmer,” *Computer*, vol. 11, no. 4, pp. 34–41, Apr. 1978.

[4] “JUnit. A programmer-oriented testing framework for Java.,” 21-Feb-2013. [Online]. Available: http://junit.org/.

[5] NUnit, “NUnit, Testing Resources for Extreme Programming,” 2005. [Online]. Available: http://www.junit.org.

[6] K.-C. Tai and Y. Lei, “A test generation strategy for pairwise testing,” *IEEE Transactions on Software Engineering*, vol. 28, no. 1, pp. 109 – 111, Jan. 2002.

[7] M. Polo, M. Piattini, and S. Tendero, “Integrating techniques and tools for testing automation,” *Software Testing, Verification and Reliability*, vol. 17, no. 1, pp. 3–39, 2007.

[8] M. Polo and B. Pérez, “A framework and a web implementation for combinatorial testing,” 2010. [Online]. Available: http://alarcosj.esi.uclm.es/CTWeb.

[9] A. Bertolino, “Software testing research: Achievements, challenges, dreams,” presented at the International Conference on Software Engineering, 2007, pp. 85–103.

[10] B. H. Smith and L. Williams, “On guiding the augmentation of an automated test suite via mutation analysis,” *Empir Software Eng*, vol. 14, no. 3, pp. 341–369, Jun. 2009.

[11] P. R. Mateo and M. P. Usaola, “Bacterio: Java Mutation Testing Tool,” presented at the International Conference on Software Maintenance (ICSM2012), 2012, pp. 646–649.

[12] A. J. Offutt, A. Lee, G. Rothermel, R. H. Untch, and C. Zapf, “An experimental determination of sufficient mutant operators,” *ACM Trans. Softw. Eng. Methodol.*, vol. 5, no. 2, pp. 99–118, Apr. 1996.

[13] K. N. King and A. J. Offutt, “A Fortran language system for mutation-based software testing,” *Softw. Pract. Exper.*, vol. 21, no. 7, pp. 685–718, Jun. 1991.

[14] P. R. Mateo, M. P. Usaola, and J. L. F. Alemán, “Validating 2nd-Order Mutation at System Level,” *IEEE Transactions on Software Engineering*, vol. 39, no. 4, 570-587.

[15] Y.-S. Ma, J. Offutt, and Y. R. Kwon, “MuJava: an automated class mutation system,” *Software Testing, Verification and Reliability*, vol. 15, no. 2, pp. 97–133, 2005.

[16] R. H. Untch, A. J. Offutt, and M. J. Harrold, “Mutation analysis using mutant schemata,” in *Proceedings of the 1993 ACM SIGSOFT international symposium on Software testing and analysis*, New York, NY, USA, 1993, pp. 139–148.

[17] P. R. Mateo and M. P. Usaola, “Mutant Execution Cost Reduction, through MUSIC (MUtant Schema Improved with extra Code),” IEEE Fifth International Conference on Software Testing, Verification and Validation, 17, 2013.

[18] P. R. Mateo and M. P. Usaola, “Parallel Mutation Testing,” *Software Testing, Verification and Reliability*, 2012.

[19] A. J. Offutt and S. D. Lee, “An empirical evaluation of weak mutation,” *IEEE Transactions on Software Engineering*, vol. 20, no. 5, pp. 337 –344, May 1994.

[20] P. R. Mateo, M. P. Usaola, and J. Offutt, “Mutation at the multi-class and system levels,” *Science of Computer Programming*, 78(4), 364-387.

[21] N. Bombieri, F. Fummi, G. Pravadelli, M. Hampton, and F. Letombe, “Functional qualification of TLM verification,” presented at the Design, Automation and Test in Europe, DATE’09, 2009, pp. 190–195.

Managing Corrective Actions to Closure in Open Source Software Test Process

Tamer Abdou
CSE Department
Concordia University
Montréal, Québec, Canada
t_moh@encs.concordia.ca

Peter Grogono
CSE Department
Concordia University
Montréal, Québec, Canada
grogono@encs.concordia.ca

Pankaj Kamthan
CSE Department
Concordia University
Montréal, Québec, Canada
kamthan@encs.concordia.ca

***Abstract*—In assessing test process maturity, one of the goals is to manage disciplinary issues. Managing corrective actions to closure is known to aid software quality assurance, in general, and testing process activities, in particular. In this paper, a framework for software testing assessment, namely OSS-TPA, that aims to evaluate corrective actions in OSS test process, is proposed. The OSS-TPA framework is based on earlier studies and relies on a conceptual model for test process activities in OSS development. Using success factors in OSS development, the relationship between the maturity of managing corrective actions and the adoption of OSS is investigated.**

***Index Terms*—Open Source Software; Software Engineering; Software Quality; Software Testing; Test Process Improvement.**

I. Introduction

In the past couple of decades, there has been a notable growth in the adoption of open source software (OSS), both by organizations and by people. The increasing commitment to OSS places ever more moral and ethical responsibility on the developers to produce better software. This, in turn, impacts the OSS development process, and calls for attention to OSS quality, in general, and OSS testing, in particular.

In this paper, the interest is in the improvement of OSS test process [1]. Indeed, process improvement (along with automation and standardization) is regarded as one of the major research directions in software testing [2]. To that end, this paper proposes an OSS test process assessment framework, henceforth abbreviated as OSS-TPA, that provides guidelines, procedures, and metrics with the aim of evaluating OSS projects.

In recent years, a number of maturity models have been proposed for evaluating OSS projects. However, these models do not focus on the underlying OSS development process, and do not adequately address issues related to testing or the maturity of the underlying testing process. This motivates the need for evaluating the OSS testing process systematically, and the OSS-TPA framework is a step in that direction. OSS-TPA is based on the Test Maturity Model Integration (TMM*i*) [3]. Furthermore, OSS-TPA relies on a conceptual framework that identifies OSS test process activities, such as Test Design and Implementation, Test Execution, and Test Incident Reporting, and aligns these activities with the ISO/IEC Standard for test process [4].

The rest of the paper is organized as follows. Section 2 analyzes existing approaches for assessing and measuring OSS projects, as well as examines related studies on the assessment of OSS test process. Section 3 provides a description of the OSS-TPA framework and its parts. Section 4 suggests avenues for future research. Finally, Section 5 provides concluding remarks.

II. Background and Related Work

The approaches for investigating the OSS test process can be classified into two categories, namely assessment and measurement [5]. These are discussed in some detail in the next two sections.

A. Assessment Approaches

An assessment approach is concerned with qualitative evaluation. In this approach, reasoning or subjective judgment is taken to conclude whether the OSS or one of its software components meet specified requirements.

A number of assessment models have been introduced over the years to provide the basis for evaluating the test process of software projects, as summarized in Table I [6]. TMM*i* is a successor of these initiatives. It provides guidelines and a reference model for test process improvement, and has proven useful in practice [3] [7] [8]. The TMM*i* reference model has sixteen process areas that include practices, ranging from general to specific, related to test process improvement. Furthermore, each process area is subdivided into a number of goals to be achieved in order to reach a specific level of maturity.

B. Measurement Approaches

A measurement approach is concerned with quantitative evaluation. In this approach, direct measures are recorded and compared to pre-established values to decide whether the OSS or one of its software components meet numerical thresholds.

The metrics for test process allow managers to track, understand, and control (and thereby improve) testing. For example, the number of test cases, defect density, and other similar metrics, provide an insight into different aspects of a test process [9] [10].

Features	TMM	TIM	TPI	TMM*i*
Model Type	Maturity	Maturity	Maturity	Maturity
Year of Development	1996	1996	1997	2008
Approach	Theoretical	Practical	Practical	Theoretical
Number of Levels	5	5	14	5
Number of Key Process Areas	13	5	20	16
Assessment Type	Questionnaire	Questionnaire	Checklist	N/A
Assessment Foundation	CMM, ISO, SPICE	Practical Experience	Practical Experience	CMMI
Information about Model	Articles, Thesis, Books	Articles	Articles, Books	Articles, Books

TABLE I: An Overview of the Main Features of Existing Test-Process Improvement Models

In recent years, a number of maturity models have been proposed for evaluating OSS projects, including OpenBQR [11], OpenBRR [12], SQO-OSS [13], and FOCSE [14]. They aim to help prospective adopters understand the features of an OSS, and to assess the advantages and drawbacks of its selection and use [15] [1].

However, these models are rather limited in their consideration of process maturity, in general, and test process maturity, in particular. For example, out of twenty-eight evaluation criteria in OpenBRR, only two criteria (namely, the average volume of the mailing list in the last six months and the number of unique contributors in the last six months) are relevant to process maturity [12]. In some maturity models, criteria for test process maturity (such as, the criterion of the availability of testing and benchmark reports in OpenBQR [16]) are mentioned, but not considered in any detail. Finally, these maturity models lack a standard basis [12], and the process of capturing data that these models are derived from is usually subjective [17]. The purpose of OSS-TPA is to overcome some of these drawbacks.

III. The OSS-TPA Framework

The OSS-TPA framework consists of four modules, namely Quality Model, Data Collection, Data Analysis, and Data Interpretation, as shown in Figure 1.

A. OSS-TPA Quality Model

The OSS-TPA quality model is based on two complementary approaches to satisfy the definition of test process evaluation [18], and thereby evaluate the OSS test process.

The first approach is the use of the Goal-Question-Metric (GQM) framework [19]. It is known that GQM provides a systematic approach towards software measurement via organization of relevant goals, questions, and metrics. The second approach is the use of the TMM*i* framework. The TMM*i* reference model specifies the test process area and provides means for controlling the scope of the goals in the OSS-TPA quality model. The attention in this paper is specifically on the aspects related to the Test Monitoring and Control process area of the TMM*i* framework.

The lack of time is among the frequently-cited reasons for organizations to not adopt Capability Maturity Model Integration (CMMI) [20].

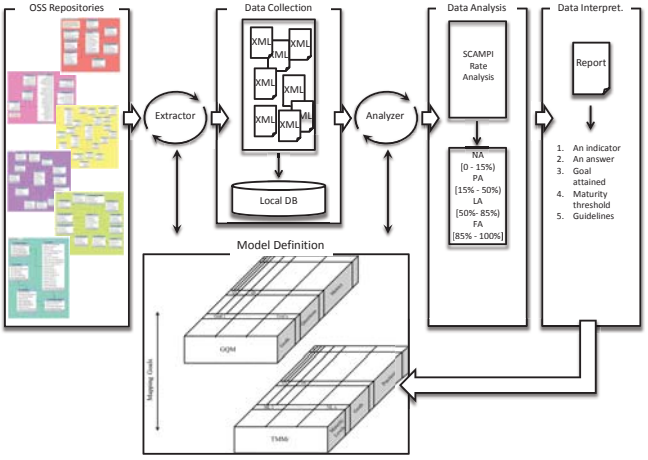


Fig. 1. A High-Level View of OSS-TPA Architecture

1) OSS-TPA Quality Model: Definition: The combination of GQM and TMM*i* contributes to decreasing the time for adopting a CMMI-based approach in an OSS project.
The measures for answering questions in the OSS-TPA quality model were computed manually, as well as, automatically, using a variety of tools.

The model definition module of OSS-TPA, as shown in Figure 2, consists of three abstract phases:

1. The conceptual phase, which derives the goal of managing the corrective actions to closure from the TMM*i* reference model.
2. The operational phase, which specifies a set of questions concerning the achievement of the goal stated in the conceptual phase. These questions are based on the relevant practices in the TMM*i* reference model.
3. The quantitative phase, which identifies a set of metrics for each specified question. These metrics are based on the work products associated with the test practices in the TMM*i* reference model.

It is known that appropriate corrective actions should be taken when test progress deviates significantly from the test plan, or product quality deviates from expectations [3].

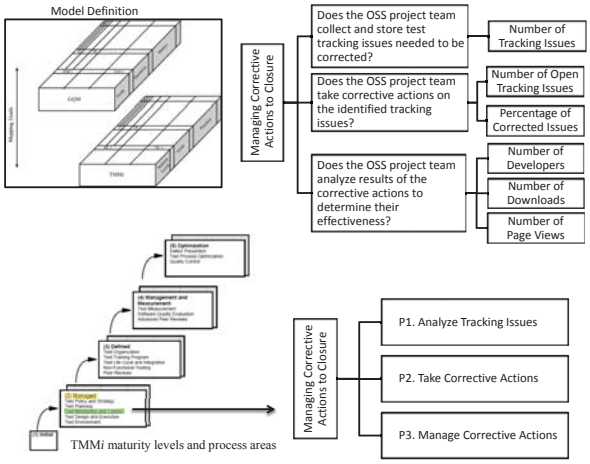


Fig. 2. A High-Level View of the OSS-TPA Quality Model

Metrics	Q_1	Q_2	Q_3
Number of Bugs	x		
Number of Patches	x		
Number of Feature Requests	x		
Number of Support Requests	x		
Number of Open Bugs		x	
Number of Closed Bugs		x	
Percentage of Corrected Bugs		x	x
Number of Open Feature Requests		x	
Number of Closed Feature Requests		x	
Percentage of Corrected Feature Requests		x	x
Number of Open Support Requests		x	
Number of Closed Support Requests		x	
Percentage of Corrected Support Requests		x	x
Number of Open Patches		x	
Number of Closed Patches		x	
Percentage of Corrected Patches		x	x
Number of Downloads			x
Number of Developers			x
Number of Page Views			x

TABLE II: Metrics of interest

Indeed, managing these actions to closure is one of the goals to be achieved in the Test Monitoring and Control process area that, in turn, belongs to Level 2 Managed in the TMM*i* framework, as shown in Figure 2.

2) OSS-TPA Quality Model: Measurement: The OSS-TPA quality model has a single goal:

Goal Manage corrective actions to closure with the aim of evaluating its maturity from a software manager's point of view.

To satisfy the aforementioned goal, the following questions are derived and formulated:

- Q_1 Does the OSS project team collect and store test tracking issues needed to be corrected?
- Q_2 Does the OSS project team take corrective actions on the identified tracking issues?
- Q_3 Does the OSS project team analyze results of the corrective actions to determine their effectiveness?

The OSS-TPA quality model includes three main quality attributes for managing corrective actions to closure, namely Analyze Issues, Take Corrective Action, and Manage Corrective Action, as shown in Figure 2. Each quality attribute is associated with a number of metrics. The selected metrics are based on the TMM*i* guidelines, and help in obtaining objective answers to the aforementioned questions.

The result, as shown in Table II, is a collection of 19 metrics, of which 4 correspond to the first quality attribute, 12 to the second quality attribute, and 3 to the third quality attribute. It can be noted that the mapping between the set of quality attributes and the set of metrics is not one-to-one.

The tracking issues in Table II are based on the classification scheme from SourceForge.net, and are categorized accordingly into four groups, namely Bugs, Feature Requests, Support Requests, and Patches.

B. Data Collection

The source of data for this study is the SourceForge Research Data Archive (SRDA) [21]. The SRDA is a repository of SourceForge OSS research data and allows the execution of SQL queries on tables exported from SourceForge.

The SQL query that follows has been used in this research to extract information on one of the tracking issues, namely bugs. For example, the following SQL query extracts the total number of bugs and the number of open ones for each project hosted on SourceForge in July 2010 for "sf0710" scheme:

```
SELECT g.group_id, ag.name, ac.count, ac.open_count
FROM sf0710.artifact_counts_agg ac, sf0710.artifact_group_list ag, sf0710.groups g
WHERE ac.group_artifact_id = ag.group_artifact_id AND g.group_id = ag.group_id
AND ag.name = 'Bugs'
```

The OSS projects with total issues (of the type bugs, feature requests, support requests, or patches) of zero have been excluded to get a valid number for the percentage of corrective issues (that is, total number of corrected issues divided by the total number of issues). Moreover, OSS projects that have assigned a NULL value to an issue have been excluded, as NULL cannot be considered as a valid number.

The data analysis module, as shown in Figure 1, deals with interpreting the collected data involving the percentage of issues for bugs, feature requests, and patches. This follows the ISO/IEC 15939 methodology for specifying indicators [22], and the ISO/IEC 15504 policy for rating indicators [23]. Each corrective action is measured on a four-point rating scale as follows:

NA:	Not Achieved	[0% - 15%)
PA:	Partially Achieved	[15% - 50%)
LA:	Largely Achieved	[50% - 85%)
FA:	Fully Achieved	[85% - 100%]

OSS Factor	Description	Indicator	Concept
Downloads	Total number of downloads of the software package	Moving from alpha to beta to stable; Achieved identified goals	Physical attribute; Community attribute
Developers	Total number of developers on the project	Activity level; User contribution; Knowledge sharing	Community attribute
Page Views	Total number of views of any of the project's website	User acceptance	Physical attribute; Community attribute

TABLE III: OSS Success/Abandonment Factors

C. Data Analysis and Data Interpretation

This section aims to answer the questions Q_1, Q_2, and Q_3 from Section III-A2. Figure 3 shows data related to the four tracking issues, and whether the data related to those issues was collected in an OSS project. For example, bugs were collected and stored in 30,029 out of 335,562 OSS projects.

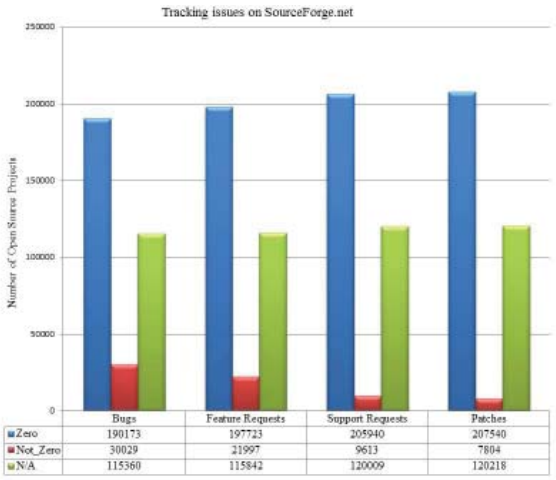


Fig. 3. Analyzing Tracking Issues

To answer the question Q_2 from Section III-A2, the following was carried out. Figure 4 shows the SCAMPI rating of OSS projects hosted on SourceForge.net. For example, 140 out of 1253 OSS projects failed to take corrective actions towards fixing bugs, while 411 out of 1253 OSS projects took corrective actions towards fixing bugs.

To answer the question Q_3 from Section III-A2, the following was carried out. The dataset consisting of 1253 OSS projects was categorized into four quartiles of 313 to 314 projects each. These were subsequently arranged in an ascending order of the applied factor of success, namely Number of Downloads, Number of Developers, and Page Views, as shown in Table III. These independent factors apply to the OSS development process [24], and allow distinguising between successful and abandoned OSS projects (for the majority of those) in the SourceForge repository [25].

Next, assuming that the data collected is randomly distributed and is not normal, chi-square test [26] has been applied to determine the effectiveness of the corrective actions by investigating the dependency between these actions and the success factors (specified later in Table V).

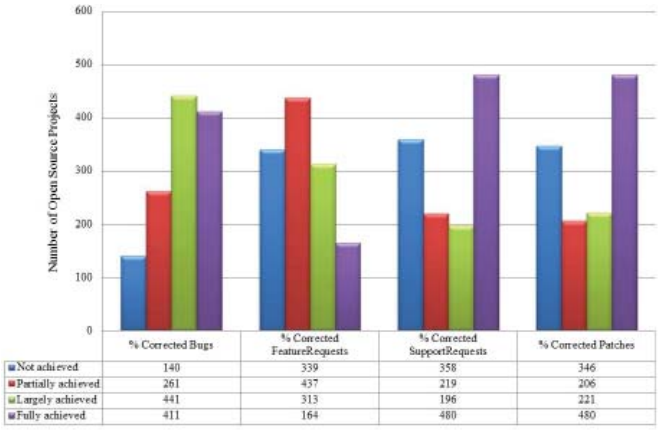


Fig. 4. Taking Corrective Actions

OSS Quart.	Frequencies	Not achieved	Partially achieved	Largely archived	Fully achieved	Totals
First	Observed	19	53	128	114	314
	Expected	35.08	65.41	110.51	103.00	
Second	Observed	42	72	103	96	313
	Expected	34.97	65.20	110.16	102.67	
Third	Observed	31	73	112	97	313
	Expected	34.97	65.20	110.16	102.67	
Fourth	Observed	48	63	98	104	313
	Expected	34.97	65.20	110.16	102.67	
Totals		140	261	441	411	1253

TABLE IV: Results of Chi-Square Test of Significance

The following conclusions can be drawn from the chi-square test results, as shown in Table IV, at a 5% level of significance.

There is insufficient evidence to conclude that the maturity of corrective actions towards fixing bugs, p-value > 0.05, is dependent on the success of OSS, using the number of downloads and the number of page views as success factors.

There is sufficient evidence to conclude that the maturity of corrective actions towards fixing bugs, p-value < 0.05, and the success of OSS, using the number of developers as a success factor, is interdependent.

There is sufficient evidence to conclude that the maturity of corrective actions towards fixing feature requests, support requests, and patches, p-value < 0.05, are dependent on the success of OSS, using the number of downloads, the number of developers, and the number of page views as success factors.

Tracking issues	Success factor	Chi-Square	P-Value	Dependency
Bugs	Number of downloads	13.938	0.125	No
Feature Requests	Number of downloads	20.170	0.017	Yes
Support Requests	Number of downloads	24.310	0.004	Yes
Patches	Number of downloads	31.607	0.000	Yes
Bugs	Number of developers	24.705	0.003	Yes
Feature Requests	Number of developers	25.969	0.002	Yes
Support Requests	Number of developers	35.291	0.000	Yes
Patches	Number of developers	49.881	0.000	Yes
Bugs	Number of page views	04.925	0.841	No
Feature Requests	Number of page views	20.469	0.015	Yes
Support Requests	Number of page views	29.262	0.001	Yes
Patches	Number of page views	43.866	0.000	Yes

TABLE V: Results of Chi-Square Test of Significance

IV. Support for OSS-TPA

The results of the previous section are supported by another empirical study [27]. In this study, six major OSS are studied to set up a reliability model using the general Weibull distribution. It is shown that widely-used measures, such as page views and downloads, are not highly correlated with the monthly bug arrival rate.

The results of the previous section also confirm the observations made in a study that has been applied on two major OSS projects, namely the Apache Web server and the Mozilla Firefox Web browser [28], namely that most bugs were reported by a relatively small developer community and not end-users. This signifies that, for most OSS projects, the number of bugs is not highly dependent on the number of page views or on the number of downloads.

V. Directions for Future Research

The work presented in this paper can be extended in a few different directions.

For example, the OSS-TPA framework can benefit from the support of further empirical studies. In particular, investigating the maturity of the OSS test processes in repositories other than SourceForge.net and/or with different process areas of the TMM*i* framework, is of research interest.

The list of success factors for OSS projects stated in this paper is not fixed, and can evolve. Indeed, other success factors might result in a different perspective on the OSS test process, and thereby constitutes another possible avenue of research interest.

VI. Conclusion

The growing number of competing software systems pose a challenge for their prospective adopters, and OSS are no different. The visibility of steps taken towards assuring the quality of an OSS is one of the most important factors towards its selection as a potential candidate.

This paper builds a foundation for evaluating and improving the OSS test process. In doing so, it presents a practical, customizable, and extensible framework, namely OSS-TPA, for understanding the management of corrective actions to closure in OSS.

The OSS-TPA framework supports the evaluation of a number of activities inherent in OSS test process, such as analyzing tracking issues, taking corrective actions, and managing corrective actions to closure, as shown by an empirical study presented in this paper. Using variations of GQM, the OSS-TPA framework can be customized to analyze the OSS test process in different contexts. Finally, OSS-TPA can be extended and applied to different maturity levels and relevant process areas of the TMM*i* framework.

VII. Acknowledgment

The authors would like to thank Olga Ormandjieva for discussions and comments on an earlier version of the paper.

References

[1] S. Morasca, D. Taibi, and D. Tosi, "Towards Certifying the Testing Process of Open-source Software: New challenges or Old Methodologies?" in *The 31st International Conference on Software Engineering (ICSE09) - The 2nd Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development (FLOSS09)*. Vancouver, Canada: IEEE, 2009, pp. 25–30.

[2] O. Taipale, K. Smolander, and H. Kalviainen, "Finding and Ranking Research Directions for Software Testing," in *Software Process Improvement*, ser. Lecture Notes in Computer Science, I. Richardson, P. Abrahamsson, and R. Messnarz, Eds. Springer-Verlag Berlin Heidelberg, 2005, vol. 3792, pp. 39–48.

[3] E. van Veenendaal and J. Jaap, "Testing Maturity - Where Are We Today: Results of the first TMMi benchmark," *Testing Experience*, vol. 3, no. 3, pp. 72–74, 2012.

[4] T. Abdou, P. Grogono, and P. Kamthan, "A Conceptual Framework for Open Source Software Test Process," in *The 36th Annual Computer Software and Applications Conference (COMPSAC12) - The 4th IEEE International Workshop on Software Test Automation (STA12)*. Izmir, Turkey: IEEE, 2012, pp. 458–463.

[5] R. S. Kenett and E. R. Baker, *Software Process Quality: Management and Control*. Marcel Dekker, 1999.

[6] R. Swinkels, "A Comparison of TMM and Other Test Process Improvement Models." Technical Report, Frits Philips Institute, Technische Universiteit Eindhoven, Netherlands, 2000.

[7] M. Rasking, "Experiences Developing TMMi as a Public Model," in *Communications in Computer and Information Science*, ser. Communications in Computer and Information Science, R. V. O'Connor, T. Rout, F. McCaffery, and A. Dorling, Eds. Springer-Verlag Berlin Heidelberg, 2011, vol. 155, pp. 190–193.

[8] International Organization For Standardization, "ISO/IEC WD 29119-2:2010 - Software and Systems Engineering - Software Testing - Test Process," 2010.

[9] N. Nagappan, L. Williams, M. Vouk, and J. Osborne, "Using In-Process Testing Metrics to Estimate Post-Release Field Quality," in *The 18th IEEE International Symposium on Software Reliability (ISSRE07)*. Trollhattan, Sweden: IEEE, 2007, pp. 209–214.

[10] I. Burnstein, *Practical Software Testing: A Process-oriented Approach*. Springer New York, 2003.
[11] D. Taibi, L. Lavazza, and S. Morasca, "OpenBQR: A Framework for the Assessment of OSS," in *Open Source Development, Adoption and Innovation*, ser. IFIP The International Federation for Information Processing, J. Feller, B. Fitzgerald, W. Scacchi, and A. Sillitti, Eds. Springer Boston, 2007, vol. 234, pp. 173–186.
[12] J.-C. Deprez and S. Alexandre, "Comparing Assessment Methodologies for Free/Open Source Software: OpenBRR and QSOS," in *Product-Focused Software Process Improvement*, ser. Lecture Notes in Computer Science, A. Jedlitschka and O. Salo, Eds. Springer-Verlag Berlin Heidelberg, 2008, vol. 5089, pp. 189–203.
[13] I. Samoladas, G. Gousios, D. Spinellis, and I. Stamelos, "The SQO-OSS Quality Model: Measurement Based Open Source Software Evaluation," in *Open Source Development, Communities and Quality*, ser. IFIP International Federation for Information Processing, B. Russo, E. Damiani, S. Hissam, B. Lundell, and G. Succi, Eds. Springer Boston, 2008, vol. 275, pp. 237–248.
[14] C. Ardagna, E. Damiani, and F. Frati, "FOCSE: An OWA-based Evaluation Framework for OS Adoption in Critical Environments," in *Open Source Development, Adoption and Innovation*, ser. IFIP International Federation for Information Processing, J. Feller, B. Fitzgerald, W. Scacchi, and A. Sillitti, Eds. Springer Boston, 2007, vol. 234, pp. 3–16.
[15] M. Michlmayr, "Software Process Maturity and the Success of Free Software Projects," in *Software Engineering: Evolution and Emerging Technologies*, K. Zieliski and T. Szmuc, Eds. IOS Press Amsterdam, The Netherlands, 2005, pp. 3–14.
[16] G. Gousios, V. Karakoidas, K. Stroggylos, P. Louridas, V. Vlachos, and D. Spinellis, "Software Quality Assessment of Open Source Software," in *The 11th Panhellenic Conference on Informatics (PCI07)*, Patras, Greece, 2007, pp. 303–315.
[17] M. Cabano, C. Monti, and G. Piancastelli, "Context-Dependent Evaluation Methodology for Open Source Software," in *Open Source Development, Adoption and Innovation*, ser. IFIP The International Federation for Information Processing, J. Feller, B. Fitzgerald, W. Scacchi, and A. Sillitti, Eds. Springer New York, 2007, vol. 234, pp. 301–306.
[18] Y. Wang and G. A. King, *Software Engineering Processes: Principles and Applications*. CRC Press, 2000.
[19] V. Basili, G. Caldiera, and D. H. Rombach, "The Goal Question Metric Approach," in *Encyclopedia of Software Engineering*, J. Marciniak, Ed. Wiley, 1994.
[20] M. Staples, M. Niazi, R. Jeffery, A. Abrahams, P. Byatt, and R. Murphy, "An Exploratory Study of Why Organizations Do Not Adopt CMMI," *Journal of Systems and Software*, vol. 80, no. 6, pp. 883–895, 2007.
[21] G. Madey, "The SourceForge Research Data Archive, SRDA, University of Notre Dame," 2010.
[22] International Organization For Standardization, "ISO/IEC 15939:2007-Systems and Software Engineering - Measurement Process," 2007.
[23] H. V. Loon, *Process Assessment and ISO/IEC 15504: A Reference Book*. Springer New York, 2007.
[24] K. Crowston, H. Annabi, J. Howison, and C. Masango, "Towards a Portfolio of FLOSS Project Success Measures," in *The 26th International Conference on Software Engineering (ICSE04) - Collaboration, Conflict and Control: The 4th Workshop on Open Source Software Engineering*, 2004, pp. 29–33.
[25] C. M. Schweik, R. C. English, and S. Haire, "Factors Leading to Success or Abandonment of Open Source Commons: An Empirical Analysis of Sourceforge.net Projects," *South African Computer Journal*, vol. 43, pp. 58–65, 2009.
[26] R. R. Johnson and P. J. Kuby, *Elementary Statistics*. Brooks/Cole, 2007.
[27] Y. Zhou and J. Davis, "Open Source Software Reliability Model: An Empirical Approach," *The 27th International Conference on Software Engineering (ICSE05) - Open Source Application Spaces: The 5th Workshop on Open Source Software Engineering*, vol. 30, no. 4, pp. 1–6, 2005.
[28] A. Mockus, R. T. Fielding, and J. D. Herbsleb, "Two Case Studies of Open Source Software Development: Apache and Mozilla," *The ACM Transactions on Software Engineering and Methodology*, vol. 11, no. 3, pp. 309–346, 2002.

Comparing Collaborative Filtering Methods Based on User-Topic Ratings

Tieke He, Xingzhong Du, Weiqing Wang, Zhenyu Chen, Jia Liu*
State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China
Software Institute, Nanjing University, Nanjing, China
*liujia@software.nju.edu.cn

Abstract

User based collaborative filtering (CF) has been successfully applied into recommender system for years. The main idea of user based CF is to discover communities of users sharing similar interests. However, existing user based CF methods may be inaccurate due to the problem of data sparsity. One possible way to improve it is to append new data sources into user based CF. Tags which are added and generated by users is one of the new sources. In order to utilize tags effectively, user-topic based CF is proposed to extract features behind tags, assign them to topics, and measure users' preferences on these topics. In this paper, we conduct comparisons between two user-topic based CF methods based on different tag-topic relations. Both methods calculate user-topic preferences according to ratings of items and topic weights. Experiments are conducted on the data set of MovieLens. The results show that user-topic based CF method is better than user based CF both in computational efficiency and recommendation effect. The effects are significant especially when each tag belongs to multiple topics.

Keywords: Recommender Systems, Collaborative Filtering, Topic Model, Tag

1. Introduction

Recommender systems [1] play an important role in E-commerce. Amazon[1], one of the most famous online retailers, recommends products to customers. And Netflix[2], one of the biggest online movie renting service providers, recommends movies to users. As one of the most important parts, recommendation algorithms have also achieved noticeable progresses.

Collaborative filtering (CF) [14] is one of the most common approaches of recommender systems. The main idea of CF is that similar users may share similar user preference patterns [6]. In CF method, user preference is represented by a vector, in which each entry indicates user's rating on a specific item. Similarities between users could be defined by the distances between user rating vectors. Such similarities could be inaccurate due to the data sparsity problem. And the rapid growth of the amount of items may lead to low computational efficiency.

With the rapid development of Web2.0 techniques, social annotation systems are showing their effects both for users and recommender systems. Researchers have introduced many approaches on using tags in recommender systems [10, 8]. Certain tags are always related to a group of relatively limited topics, and are related to item features. Such as a movie about intelligent robots may relate to topics on science fiction or high technology. Based on these facts, we assume that a user likes some topics before he or she likes some certain items. How much a user likes an item could be measured by the related tag ratings and the correlation of the topic and the item. Based on this assumption, user-topic ratings could enhance CF method.

In user-topic approach, each item is treated as a document, and collection of items as corpus. Tags are regraded as the words for the documents. There are different ways to improve CF by user-topic ratings: one tag to one topic – hierarchical clustering and one tag to multiple topics – Latent Dirichlet Allocation (LDA) [2, 16].

LDA could discover topics of items. It generates a series of analysis results including document-topic proportions, topic-words proportions and so on. LDA could regard the document-topic proportions as the correlations of topics and movies. With these data, user-topic ratings are inferred, and then user similarity could be computed.

In this paper, we conduct experiments on two different user-topic based CF, and compare them to user-based CF. The sparsity of data set is controlled carefully, and results are evaluated by classical metrics. Results show that keeping other factors unchanged, either user-topic based CF is better than user-based CF. In these two user-topic based CF methods, LDA could improve CF better.

[1] http://www.amazon.com
[2] http://www.netflix.com

The main contributions of this article are summarized as follows.

1. Based on inferred tag ratings, we choose two different topic model – hierarchical clustering and LDA to implement two different user-topic based CF.

2. We compare user-topic based CF to user based CF, and show the effectiveness of user-topic based CF. We also illustrate that LDA is better than hierarchical clustering in user-topic based CF.

This paper is organized as follows. Section 2 gives a brief introduction to user-based collaborative filtering method. Section 3 introduces two methods to do topic extraction, tag clustering and LDA model. Section 4 shows our experiments and the analysis of experimental results. A conclusion of our work is showed in Section 5.

2. User-Based Collaborative Filtering

Collaborative filtering (CF) has been widely used in business situations. CF methods consist of User-Based CF, Item-Based CF and other variations. The main idea of User-Based CF is that similar users may share similar preferences. It calls for calculating similarities between users by their ratings on items. A higher similarity between two users means they are much more similar.

Given user list $U = \{u_1, u_2, \ldots, u_n\}$ and item list $\{i_1, i_2, \ldots, i_m\}$, a user u can be represented by his rating vector $r_u = (r_{u,1}, r_{u,2}, \ldots, r_{u,m})$. $r_{u,i} (i \in (1, m))$ stands for user u's rating on the ith item. The similarity between user u and v can be measured by the distance between r_u and r_v, known as the Pearson's Correlation Coefficient. Equation 1, which states the similarity between user u and user v, denotes the computation of Pearson's Correlation Coefficient.

$$sim_{u,v} = \frac{\sum_{g \in G} (r_{u,g} - \overline{r}_{u,G})(r_{v,g} - \overline{r}_{v,G})}{\sqrt{\sum_{g \in G} (r_{u,g} - \overline{r}_{u,G})^2} \sqrt{\sum_{g \in G} (r_{v,g} - \overline{r}_{v,G})^2}} \quad (1)$$

To note the items rated by both of u and v (known as the co-rated items), G is used to stand for the set of co-rated items of u and v. $\overline{r}_{u,G}$ and $\overline{r}_{v,G}$ represents u's and v's average rating of G. A user may have personal bias on rating, which means a user may always tend to give high or low ratings. To alleviate this situation, each user's average rating is subtracted from his or her ratings.

$$pred(u, i) = \overline{r}_{u,G} + \frac{\sum_{v \in N_{sim}} sim_{u,v} \cdot (r_{v,i} - \overline{r}_{v,G})}{\sum_{v \in N_{sim}} |sim_{u,v}|} \quad (2)$$

Equation 2 demonstrates how to predict a user $u's$ rating on an item i. When user u's rating $r_{u,i}$ on an item i is predicted, only a set of similar users to user u which are denoted by N_{sim} are taken into prediction. Items can be recommended to user u, if their predicting ratings are higher than user u's average rating.

However, there always exist a lot of items which user u has rated but v has not, or vice verse. This could lead to inaccuracy due to too few co-rated items. A new user has no co-rated items with other users which makes it even worse.

3. User-Topic Based Collaborative Filtering

User-topic based CF is an improvement to the user-based CF method. Instead of using the sparse user-item ratings, we use inferred user-topic ratings to compute user similarity. Traditional prediction methods are used to generate recommendations for users. This section presents a detailed explanation of user-topic based CF.

3.1. Topic Extraction

The basic motivation behind user-topic based CF is to measure to what extent a user likes a specific topic. The most important is to extract the abstract topics, also known as latent semantics inside the items. Social annotation systems provide a chance. In social annotation systems, users use tags to express their personal viewpoints on items. This makes it possible for us to extract topics from these tags.

There are two methods to carry out this work – tag clustering and topic models. More specifically, hierarchy clustering [13] and the Latent Dirichlet Allocation (LDA) model [2]. Both methods are state-of-art algorithms. This section gives the process and formulation of these two methods, including mathematical notations.

3.2. Hierarchical Clustering on Tags

Hierarchical Clustering is a simple while useful clustering algorithm [13, 15]. In detail, there are two main types of Hierarchical Clustering which are top-down approach and bottom-up approach. We adopt the bottom-up approach in this paper. Given a set of tags $T = \{t_1, t_2, \ldots, t_n\}$, t_i denotes a certain tag. At first, each tag is placed in a single topic, so the initial set of clusters is

$$C = \{c_1 = \{t_1\}, c_2 = \{t_2\}, \ldots, c_n = \{t_n\}\}$$

In each iteration, two nearest clusters are picked out and aggregated together. So the distance between two clusters should be defined. In this paper, the distance between clusters is computed by tags. [3] proposed a method named co-occurrence probability. A tag z is denoted as an vector p_z,

which stands for its co-occurrence probabilities distribution. Co-occurrence probability $p_z(t)$ means the probability that tag t is tagged if tag z is tagged on one item. It is defined in Equation 3.

$$p_z(t) = \sum_{m \in I} q(t|m)Q(m|z) \tag{3}$$

$$q(t|m) = \frac{number\ of\ times\ tag\ t\ on\ item\ m}{overall\ number\ of\ tags\ on\ item\ m} \tag{4}$$

$$Q(m|z) = \frac{number\ of\ times\ tag\ z\ on\ item\ m}{number\ of\ times\ tag\ z\ on\ all\ items} \tag{5}$$

$z's$ co-occurrence probabilities can be expressed in a vector p_z and its feature is described in Equation 6.

$$\sum_{t=1}^{n} p_z(t) = 1 \tag{6}$$

Since tags could be represented as co-occurrence probability distribution, the distance between tags could be computed by Jensen-Shannon divergence (JSD). A correct JSD result is a finite value ranging from zero to one. Equation 7 is the computation of JSD.

$$JSD(A||B) = \frac{1}{2}D(A||M) + \frac{1}{2}D(B||M) \tag{7}$$

$$M = \frac{1}{2}(A + B) \tag{8}$$

$$D(A||B) = \sum_{i=1}^{n} A(i) \ln \frac{A(i)}{B(i)} \tag{9}$$

A and B are the co-occurrence distributions of two tags. n is the dimension of A or B. In Equation 9, if $A(i)$ and $B(i)$ are both 0 then we define $\frac{A(i)}{B(i)}$ as 1. We now can define the distance between clusters based on the distance between tags. Given cluster c_i and c_j with N_i and N_j tags respectively. The distance between c_i and c_j is defined in Equation 10.

$$Dis(c_i, c_j) = \frac{\sum_{t_1 \in c_i} \sum_{t_2 \in c_j} JSD(t_1||t_2)}{N_i * N_j} \tag{10}$$

A set of clusters are generated after all the iterations accomplished. Next step is to compute each topic's weight on each item. Equation 11 describes the computation of tag z's weight on item m. Equation 12 describes the computation of topic c's weight on item m.

$$w_m(t) = \frac{\sum_{z \in T_m} n(m, z) * p_z(t)}{\sum_{z \in T_m} n(m, z)} \tag{11}$$

$$w_m(c) = \sum_{t \in c} w_m(t) \tag{12}$$

In Equation 11, T_m stands for all tags applied to item m, and $n(m, z)$ denotes the times that tag z has been tagged on item m. In Equation 12, topic weight is the sum of all included tag weights.

3.3. Latent Dirichlet Allocation

This section introduces how to get the item-topic weights by topic model. Topic models have been widely used in many areas, especially in the area of NLP (Natural Language Processing). [4] proposed the Latent Semantic Index (LSI). They used the SVD method to do dimension reduction. Actually, LSI is not a style of topic model but the basis of probabilistic latent semantic analysis(pLSI). pLSI [7] does similar work with LSI, but pLSI is a generative probability model.

Latent Dirichlet Allocation (LDA) is a popular topic model. It performs well in many research works. In LDA, each document is drawn from a distribution over a specific group of topics, and these topics are shared by all the documents in the corpus. Each topic is a distribution over a vocabulary which contains all the unique words in the corpus. Given a set of items as a corpus, each item in the set as a document and tags of the items as words, it is natural to use the LDA model in our scenario. A basic assumption for the LDA model is "bag-of-words", which means the order of a certain document's words can be neglected. So the tags are disordered in our scenario.

Original LDA is proposed by Blei based on the EM algorithm. [5, 9] proposed a simple parameters estimation method called the Gibbs Sampling. And [11] proposed an implementation of the above method. It is applied in our experiment for its usability. Although the LDA model can infer both document-topic distribution and topic-word distribution at the same time, the former is essential for the following computation. The LDA model assumes that document-topic distribution is drawn from a Dirichlet Distribution. The sum of all topic probabilities in a certain document is 1 according to the definition of the Dirichlet Distribution. With this definition, it is unnecessary to normalize the item-topic probabilities, they can be directly used as item-topic weights.

In this paper, an implementation of LDA – GibbsLDA++ is applied. The input of the GibbsLDA++ is the corpus in a text format, including each tag's count in each item. It also needs several input parameters: the number of topics, hyper-parameter $alpha$ and $beta$. We find that 100 is an appropriate number of topics according to the experimental results, and $alpha$ and $beta$ are set as default value of $\frac{50}{count\ of\ topics}$ and 0.1 respectively. 2000 Gibbs Sampling iterations were conducted. After that, GibbsLDA++ gives

the document-topic proportions and they could be transformed into item-topic weights.

3.4. Inferred user-topic Ratings

Item-topic weights could be generated from the above methods . Next step is to infer the user-topic ratings, which is the foundation of user similarity computation. Each item relates to the universal set of topics, but it has some outstanding topics or known as high-weighted topics. If a user rates high for an item means that he or she likes that item, he would prefer higher-weighted topics to lower-weighted topics of that item, which indicates that the user is attracted by the item's outstanding topics with a high probability. Therefore, inferring user $u's$ rating on topic t could be calculated by equation 13.

$$r_u(t) = \frac{\sum\limits_{m \in I_u} w(m,t) * r_{u,m}}{\sum\limits_{m \in I_u} w(m,t)} \tag{13}$$

I_u stands for all the items that user u has rated. $w(m,t)$ denotes the weight of tag t on item m. $r_{u,m}$ represents u's rating on m. The user-topic ratings are denser than user-item ratings, and applying user-topic ratings can well improve recommendation accuracy. Finally, a vector r_u is generated for each user to express his ratings on topics, which will be used in the following steps.

3.5. Generate Recommendation

After the preivous steps, each user is denoted by a rating vector r_u consisting of his user-topic ratings. In our experiments we use the Pearson Correlation Coefficient to compute user similarity. A modified edition is denoted in Equation 14. Then Equation 15 is used to predict user ratings on movies.

$$sim_{u,v} = \frac{\sum\limits_{t \in T} (r_{u,t} - \overline{r}_u)(r_{v,t} - \overline{r}_v)}{\sqrt{\sum\limits_{t \in T} (r_{u,t} - \overline{r}_u)^2 \sum\limits_{t \in T} (r_{v,t} - \overline{r}_v)^2}} \tag{14}$$

$$pred_{u,m} = \overline{r}_u + \frac{\sum\limits_{v \in U_m} sim_{u,v} \cdot (r_{v,m} - \overline{r}_v)}{\sum\limits_{v \in U_m} |sim_{u,v}|} \tag{15}$$

Each user u has a vector v which denotes his ratings on topics. In practical, user u has ratings on all topics. In Equation 14, T stands for the set of all the topics. $r_{u,t}$ is u's rating on topic t, while $r_{v,t}$ is for user v. The average ratings of u and v on topics are denoted by $\overline{r}_u$ and $\overline{r}_v$. In Equation 15, U_m denotes the set of all the users that have rated item m.

After getting the predicted ratings for u, it is easy to generate a recommendation list for u. By sorting the predicted ratings and choosing k highest ratings, we can recommend top k items to u.

4. Experimental Evaluation

4.1. Evaluation Metric

In this paper, we choose MAE(Mean Absolute Error) and RMSE(Root Mean Squared Error) to evaluate the experiments. These two metrics are often used to evaluate recommendation accuracy, especially for those recommender systems which produce predicted ratings. They reveal the average errors between real ratings and predicted ratings. Equation 16 describes MAE's computation, and Equation 17 describes RMSE's.

$$MAE = \frac{\sum\limits_{i=1}^{N_{pr}} |r_i - \hat{r}_i|}{N_{pr}} \tag{16}$$

$$RMSE = \sqrt{\frac{\sum\limits_{i=1}^{N_{pr}} (r_i - \hat{r}_i)^2}{N_{pr} - 1}} \tag{17}$$

In Equation 16 and 17, N_{pr} stands for the count of predicted ratings. r_i and $\hat{r}_i$ are real rating and predicted rating. Given the same recommendation results, RMSE will always be larger than or equal to the MAE. And the difference between them reflects the variance of the individual errors in the recommendation results. In both metrics, a low value means a high accuracy.

Table 1. Final Data Sets

	T(r)	n(u)	n(r)
	[75,79)	826	64022
T(t):[15,∞)	[100,105)	749	77138
n(m)=484	[155,165)	624	99985
n(t)=1154	[215,240)	595	134916
	[280,∞)	543	174350

4.2. Data Set and Preprocessing

We choose the MovieLens data set which contains both the user-movie ratings and tags. This data set contains 10,000,054 ratings and 95,580 tags applied to 10,681 movies by 71,567 users of the online movie recommender service MovieLens.

The original data set contains too many noisy records which will interfere the experimental results. In that case,

Table 2. Experimental Results

T(t)	[15,∞)									
T(r)	[75,79)		[100,105)		[155,165)		[215,240)		[280,300)	
Sparsity	0.90		0.85		0.73		0.59		0.40	
Metrics	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE
CF	0.6672	0.7512	0.6402	0.7010	0.5947	0.6227	0.5809	0.5933	0.5734	0.5734
TC	0.6485	0.7153	0.6344	0.6883	0.5967	0.6235	0.5834	0.5989	0.5756	0.5769
LDA	0.6441	0.7068	0.6284	0.6762	0.5923	0.6134	0.5776	0.5877	0.5701	0.5667

we adopt certain rules to filter the noises. We will discuss it in the following paragraphs. After preprocessing, we divide the data set into samller parts by sparsity. This allows us to compare these state-of-art algorithms in different sparsity.

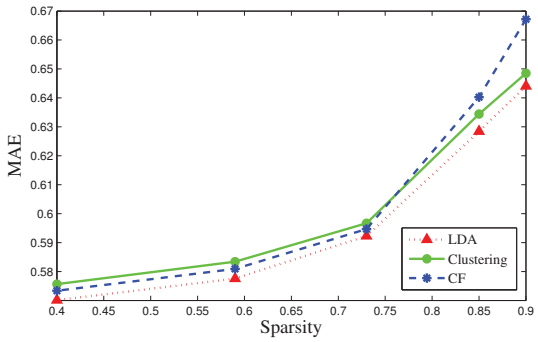


Figure 1. Comparisons in terms of MAE

We can divide tags into various kinds. Some of them express user personal preference, such as amazing, wonderful or nice. Other express item features, such as red, cubic and so on. Tags that used by users to express their likes or dislikes on movies are useless and will influence the experimental results. Therefore, we have to remove this kind of tags.

[12] proposed a method to do this job. The idea of their method is that tags expressing user preference can be divided into two types, the like tags and the dislike tags. If a user u gives a rating higher than his average rating on a movie m, then we say that movie m is a like movie for user u. We can use the same way to define dislike movies for users. So if a tag t only occurs in like movies, tag t is a like tag. Now we can remove this kind of tags. We also need to remove unpopular tags. Therefore, these tags used by less than 2 users or have been assigned to less than 5 movies are removed.

In order to compare experimental results under different data sparsity, we have set several thresholds for count of ratings and tags. Finally we got 10 different sub data sets. Full description of each sub data set is shown in Table 1.

In Table 1, $T(t)$ means count of each movie's tags ranges in that interval. $T(r)$ means the amount of each user's ratings ranges in that interval. $n(m)$ denotes the amount of movies, while $n(t)$ denotes the amount of tags. $n(u)$ and $n(r)$ stand for the amount of users and the amount of ratings respectively.

4.3. Experimental Results and Analysis

After preprocessing, we finally get ten data sets. Each data set will be divided into two parts, the training set and the test set. For each user, we randomly select 30 ratings to construct the test set. And the remaining part is the training set. For each data set we have tried all the three methods – CF method, tag clustering and LDA. We also compare the experimental results in MAE and RMSE.

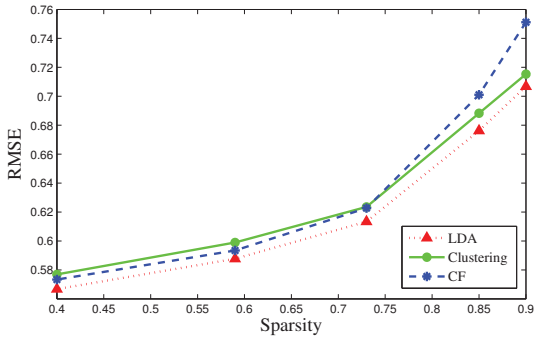


Figure 2. Comparisons in terms of RMSE

In Table 2, the computation of Sparsity S of the training set is denoted by Equation 18.

$$S = 1 - \frac{Count(Ratings)}{Count(Users) \cdot Count(Movies)} \tag{18}$$

Figure 1 and 2 reveal that the LDA method always performs better than CF method and tag clustering method. But when the ratings become denser, its advantage over CF method won't be obvious. Actually this is a phenomenon that could be explained – CF method suffers from data sparsity problem, but CF method could produce more accurate recommendation results with denser ratings. The tag clustering method only performs better than CF method with sparse ratings.

Overall these two figures suggest that user-topic based CF method are better than user based CF method. In user-topic based CF, LDA is better than tag clustering. It is because a tag is assigned to multiple topics in LDA, but is assigned to only one topic in tag clustering. In practice, a tag could carry different meanings in different perspectives. LDA extracts each topic from the distributions over the set of tags, so its implementation is closer to reality.

Nevertheless, when ratings become denser, the dimension of user ratings vector increases. CF method becomes inefficient due to computation complexity of user similarity. But the LDA method will still be effective if the amount of topics is defined. From the perspective of recommendation effect and computation efficiency, LDA outperforms the other two methods.

5. Conclusions

This paper presents comparisons between user-based and user-topic based CF along with different sparsity. We use tag clustering and the LDA model to do topic extraction from items, then we compute user preferences on the latent topics. As we did in user-based collaborative filtering, we predict user ratings by user-topic preferences.

The results show that, when ratings are relatively sparse, LDA model performs much better than user-based CF method. When ratings become relatively denser, LDA model still outperforms CF method and achieves better efficiency. We believe that each tag belonging to multiple topics makes the user-topic based CF more accurate.

There still exist some insufficiency in our experiments. Such as tags which represent users' preference are directly removed from the data set or the LDA model can also be used to model users. We will try to relieve these issues in the future.

6. Acknowledgments

The work described in this article was partially supported by the National Natural Science Foundation of China (11171148, 61003024, 61170067).

References

[1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *TKDE*, 17(6):734–749, 2005.

[2] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *JMLR*, 3:993–1022, 2003.

[3] W. Christian, B. Rogier, and W. Martin. Using tag co-occurrence for recommendation. In *ISDA*, pages 273–278, 2009.

[4] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *JASIS*, 41:391–407, 1990.

[5] T. Griffiths. Gibbs sampling in the generative model of latent dirichlet allocation. Technical report, Stanford University, 2002.

[6] S. Hoeffler and D. Ariely. Constructing stable preferences: A look into dimensions of experience and their impact on preference stability. *Journal Of Consumer Psychology*, 8:113–139, 1999.

[7] T. Hofmann. Probabilistic latent semantic indexing. In *SIGIR*, pages 50–57, 1999.

[8] R. Krestel and P. Fankhauser. Language models and topic models for personalizing tag recommendation. In *Web Intelligence*, pages 82–89, 2010.

[9] G. T. L. and S. Mark. Finding scientific topics. *PNAS*, 101(Suppl 1):5228–5235, 2004.

[10] A. K. Milicevic, A. Nanopoulos, and M. Ivanovic. Social tagging in recommender systems: a survey of the state-of-the-art and possible extensions. *Artificial Intelligence Review*, 33(3):187–209, 2010.

[11] X.-H. Phan and C.-T. Nguyen. Gibbslda++: A c/c++ implementation of latent dirichlet allocation (lda). Technical report, 2007.

[12] Q. Qi, Z. Chen, J. Liu, C. Hui, and Q. Wu. Using inferred tag ratings to improve user-based collaborative filtering. In *SAC*, pages 2008–2013, 2012.

[13] A. Shepitsen, J. Gemmell, B. Mobasher, and R. Burke. Personalized recommendation in social tagging systems using hierarchical clustering. In *Recsys*, pages 259–266, 2008.

[14] X. Su and T. M. Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in Artificial Intelligence*, 2009:1–20, 2009.

[15] W. Wang, Z. Chen, J. Liu, Q. Qi, and Z. Zhao. User-based collaborative filtering on cross domain by tag transfer learning. In *Proceedings of the 1st International Workshop on Cross Domain Knowledge Discovery in Web and Social Network Mining*, pages 10–17, 2012.

[16] S. Xiance and S. Maosong. Tag-lda for scalable real-time tag recommendation. *JCIS*, 6(1):23–31, 2009.

ABEY: an Incremental Personalized Method Based on Attribute Entropy for Recommender Systems

Xingzhong Du, Tieke He, Zhenyu Chen, Jia Liu*, Chengfeng Hui
State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China
Software Institute, Nanjing University, Nanjing, China
*liujia@software.nju.edu.cn

Abstract

Recording attribute frequencies and calculating user preferences on attributes are generally used in current personalized method – user profile for recommender system. In this paper, we propose a new personalized method, namely ABEY, to do the personalization in another way. ABEY firstly uses attribute entropies to calculates user preferences on attribute classes. ABEY refines these preferences by wiping off some interferences, and allocates a group of such preferences for each user at the end of measurement. ABEY could re-rank given recommendations, according to aggregate attribute class preferences for them. Experiments are conducted on data sets collected from an E-commerce site in real world. The effectiveness of ABEY, including accuracy, coverage and novelty, has been evaluated and the results indicate that ABEY can be a superior alternative to user profile on finding related items for individuals.

Keywords: Personalized Method, User Profile, User Preference, Attribute Entropy, Recommender Systems

1. Introduction

User profile, built by the content-based filtering, contains item attributes and user preferences. It could personalize recommendations for individuals [6]. As a consequence, user profile has been widely incorporated with other recommending methods in recommender systems. In particular, it is often combined with collaborative filtering (CF), which provides accurate recommendations by calculating similar users or items [1, 10, 2, 4, 3, 5], to help CF obtain more personalized results.

Nonetheless, some new attributes belonging to recommended items are not included in individual user profile [8, 7]. In this situation, the measurement of user preferences is incomplete, which causes a invalidation in personalized recommending. For an instance, if a user has not met any item of black color, the user profile could not reflect his or her preference on black. When some black items are recommended to the user, they would be filtered by the user profile. To accommodate this situation, an available way is to find substitutes for these unknown preferences.

In this paper, we propose a new personalized method ABEY, which measures user preferences on attribute classes by attribute entropies. ABEY transforms the attribute entropies into user preferences after wiping off the interferences of global attribute entropies. It allocates a group of such preferences for each user. ABEY could be incrementally implemented on the existing recommender systems. It calculates user preferences for recommendations and re-ranks the results by considering their original rankings and preferences simultaneously.

The main contributions of this article are summarized as follows.

1. We firstly introduce attribute entropy to measure user preferences on attribute classes. It is believed to have a positive impact on personalized recommending.
2. We propose a new personalized method based on AttriBute EntropY (ABEY). ABEY is implemented as an increment to CF, which operates similarly as user profile. Therefore, it could improve the performance of CF on finding related items for individuals without any changes to original algorithm.

The rest of the paper is organized as follows: It begins with introducing the details of ABEY in Section 2. Experiments and their results are illustrated in Section 3. Finally, Section 4 concludes our work.

2. ABEY Method

Similar to user profile, ABEY calls for the attributes of all items collected and classified before user preferences on attribute classes are calculated. With these detailed attribute information, a user's browsed item attributes could

be aligned by attribute classes as the example in Table 1. For each attribute class, attributes and their frequencies construct an attribute distribution. Then the attribute entropies are derived from these distributions, which is calculated by Equation 1. In Equation 1, e is the entropy of the specific attribute class, n is the number of different attributes in this class, f_j is the jth attribute's occurrence, f_t is the sum of all the occurrences. ABEY connects attribute entropies and user preferences on attribute classes in this way: low attribute entropies means high user preferences on attribute classes and vice versa.

Table 1. A user's item browsing history aligned by attribute classes

Browsing history			
Item	Brand	Color	Price
$item_1$	A	red	\$3
$item_2$	B	blue	\$2
$item_3$	C	green	\$20
$item_4$	D	yellow	\$10
⋮	⋮	⋮	⋮

$$e = -\sum_{j=1}^{n} \frac{f_j}{f_t} log \frac{f_j}{f_t} \ (f_t = \sum_{j=1}^{n} f_j) \tag{1}$$

ABEY calculates a group of attribute entropies for a user according to the amount of attribute classes after calculation. Nevertheless, these attribute entropies are limited to reflect user preferences on attribute classes to some extent. This is because users are regarded free on attribute choosing while they are influenced by external factors actually. For an extreme example in Table 2, if all the items are of the same color, the user preference on color is 0 without exception. In this situation, user attribute entropy is forced to be low by the offered monotonous attributes.

Table 2. Global attribute diversity will influence user entropy of colors

Item attributes			
Items	Brand	Color	Price
$item_1$	A	red	\$3
$item_2$	B	red	\$2
$item_3$	C	red	\$20
$item_4$	D	red	\$10
⋮	⋮	⋮	⋮
A user who has already visited A, B, C			
Entropy of color			0

Thus, user preferences on attribute classes are also influenced by the attributes the web sites offered, which we call it global attribute diversity. The global attribute diversity could be measured by Equation 1 as well. Such entropy is called as global attribute entropy, and a high global attribute diversity means a high entropy.

ABEY applies the global attribute entropy into the calculation of preference. It uses the differences between global entropies and user entropies to represent user preferences on attribute classes finally. The final preferences are calculated by Equation 2, in which d is the difference, E is the global entropy, e is the user entropy, and c is an attribute class. A high d_c means the user has high preference on attribute class c.

$$d_c = E_c - e_c \tag{2}$$

Given an item g and a set of attribute classes C, ABEY could calculate user preference for item i by Equation 3. In Equation 3, $M(i, c, g)$ is a variant of $M(i, a)$. It returns 1 if i and g have the same attribute in class c, while returns 0 for other cases.

$$p_u(i, g) = \sum_{c \in C} d_c M(i, c, g) \tag{3}$$

3. Experiments

We conducted temporal experiments on a data set to examine the effect of ABEY, comparing with user profile (UP). These data were collected from an E-commerce site[1]. Three metrics about accuracy, coverage and novelty were used to evaluate the experiments respectively.

3.1. Comparisons

We set item-based CF as the base line, and compare ABEY with user profile (UP):

CF: It is the item-based CF that was used in experiments. It uses the co-browsing usages to compute the item-item similarities and the most similar ones as recommendations. Given an item, the detailed operations of item-based CF are divided into following steps:

1. Searching a group of users who have visited this item.
2. For each user found in step 1, constructing a usage vector which includes all distinct visited items and their usages.
3. For each distinct visited item, adding it to a recommending list and counting its total amount of usages over all usage vectors constructed in step 2.
4. Returning the recommending list ranked by items' total amount of usages from high to low.

[1]To avoid business conflicts, we do not announce the URLs in this paper.

UP: The incremental method is user profile. For each individual, recommending list is re-ranked by the preferences calculated by user profile.

ABEY: The incremental method is ABEY, and the recommending list is re-ranked by the preferences calculated by ABEY for individuals.

3.2. Data Set

The data set was collected from an E-commerce site which sells eyeglasses. The data set consists of two parts. One is the user browsing history where each record contains a timestamp, an item id and a user id. The other part is item attribute information. It consists of the item attributes which are shown on the item pages. The collected attribute classes are shown in Table 3.

Table 3. The attribute classes of the glasses

Name	Attribute Classes
glasses	material, price, style, brand, color, gender

After the data collecting was finished, a user-item pair is allocated the value 1 if the user had visited the item or 0 in the contrary situation, which constructs the usage matrix in the end. The basic information of glasses site is displayed in Table 4.

Table 4. The basic information of the data set

Type	# Users	# Items
glasses	277781	3308

Those users who had visited at least 10 distinct items were picked out to ensure that user profile could store more user preferences. For each selected user, his or her distinct visited items were ordered by the timestamps from early to late. After that, the fourth latest timestamp was picked out, and according to this timestamp these visited items are divided into three sets:

Test set is used to test the recommending effect. It contains the last three ones among the user's distinct visited items. We limit the size to 3, because it is enough to differentiate the effects of different recommending methods.

Context set is used as the item which the user is visiting. It contains the latest item which is browsed before test set.

Training set is used by item-based CF to generate recommendations. It contains all item visits of all users which occur before the context set.

3.3. Evaluation Metrics

In this part, we introduce the metrics of accuracy, coverage and novelty, which were used in our experiments.

3.3.1 Accuracy

Accuracy describes the capacity of recommender systems predicting users' opinions over items or the probability of usage. The metric of accuracy used in our experiments is *precision* which has been widely applied by information retrieval [3, 9]. If the universal set of users is *U*, the size of *U* is represented by —*U*—, as well as hit_u and rec_u belong to u respectively, the *precision* P of a recommending method will be calculated by Equation 4.

$$P = \frac{\sum_{u \in U} |hit_u|}{\sum_{u \in U} |rec_u|} \tag{4}$$

The sets of hitting users – $U(H)$, which contains the users who are recommended the items included in their test sets, are also collected in experiments. It reveals that how many users are benefited from different methods.

3.3.2 Coverage

In our experiments, the catalog coverage was applied. We divided the coverage into two aspects: one is recommending coverage $C(R)$, and the other one is hitting coverage $C(H)$. Recommending coverage is used to express the coverage of the recommender method, while hitting coverage is a verification that the goal of increasing coverage is achieved. If the universal set of items is *I*, these two kinds of coverage to a recommender method are calculated by Equation 5.

$$C(R) = \frac{|\bigcup_{u \in U} rec_u|}{|I|} \qquad C(H) = \frac{|\bigcup_{u \in U} hit_u|}{|I|} \tag{5}$$

3.3.3 Novelty

Novelty is a capacity to recommend items which user may be not aware of but will finally find on their own [5, 11]. In order to calculate an item's novelty value, its popularity value is calculated first by Equation 6 with the visited frequency of the item v. A high popularity means a low novelty, so an item novelty value n is calculated by Equation 7, in which pop_{max} is the maximum popularity value among all items.

$$pop = \frac{v}{\sum_{k \in I} v_k} \tag{6}$$

$$n = 1 - \frac{pop}{pop_{max}} \tag{7}$$

And with all items' novelties, recommending novelty and hitting novelty of a recommending method are calculated by Equation 8 and 9.

$$N(R) = \frac{\sum_{u \in U} \sum_{i \in rec_u} n_i}{\sum_{u \in U} |rec_u|} \tag{8}$$

$$N(H) = \frac{\sum_{u \in U} \sum_{i \in hit_u} n_i}{\sum_{u \in U} |hit_u|} \tag{9}$$

3.4. Experimental Results

The results show that ABEY achieves an amazing *precision* of 0.0923. Its recommending and hitting coverage are both 5% higher than others at the same time. It is also interesting to see that UP fails to improve the amount of hitting users at the moment, which leads a noticeable negative influence (around -2%).

Table 5. Performance on glasses site. The bold number indicates a maximum improvement

Glasses Recommendation @ 5						
Method	N(R)	C(R)	P	N(H)	C(H)	U(H)
CF	0.781	60.7%	0.082	0.837	38.5%	4889
UP	0.775	59.7%	0.080	0.834	37.5%	4809
ABEY	**0.813**	**67.8%**	**0.092**	**0.862**	**45.4%**	**5409**

3.5. Results Analysis

The results of glasses site reveal that ABEY improves recommending effect in terms of accuracy, coverage and novelty than UP. And since ABEY keeps a higher coverage and accuracy at the same time, it is also reasonable that ABEY makes more individually preferred items recommended to users.

Overall the results suggest that, ABEY achieves higher recommending coverage than other comparisons and consequently makes the hitting coverage a rational increment, which also occurs on novelty. What's more, the results show that all of these increments are not at the expense of accuracy. Therefore, we believe that, when recommendations are offered on item pages, ABEY could substitute UP.

4. Conclusions

In this paper, we propose an incremental personalized method – ABEY – to improve user profile on item pages using attribute entropy. Based on attribute entropy, ABEY generates users' multiple preferences on attribute classes, and calculates user preferences for recommendations by them. As a result, recommendations on item pages are re-ranked to meet what the users are concerning about.

We implemented ABEY and conducted temporal experiments on data a set from a glasses site. Given item-based CF as base line and compared to a classical incremental method – user profile (UP), ABEY's performances noticeably outperform it in most cases. As a consequence, we believe that ABEY is useful for many application scenarios, because item-based CF is widely used on item page recommendation in practice.

5. Acknowledgments

The work described in this article was partially supported by the National Natural Science Foundation of China (11171148, 61003024, 61170067).

References

[1] J. S. Breese, D. Heckerman, and C. M. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *AIED*, pages 43–52, 1998.

[2] J. L. Herlocker, J. A. Konstan, and J. Riedl. Explaining collaborative filtering recommendations. In *ACM CSCW*, pages 241–250, 2000.

[3] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. Riedl. Evaluating collaborative filtering recommender systems. *TOIS*, 22(1):5–53, 2004.

[4] G. Linden, B. Smith, and J. York. Amazon.com recommendations: Item-to-item collaborative filtering. *INTERNET*, 7(1):76–80, 2003.

[5] S. M. McNee, J. Riedl, and J. A. Konsta. Being accurate is not enough how accuracy metrics have hurt recommender systems. In *CHI Extended Abstracts*, pages 1097–1101, 2006.

[6] S. E. Middleton, D. D. Roure, and N. Shadbolt. Capturing knowledge of user preferences: ontologies in recommender systems. In *ICKC*, pages 100–107, 2001.

[7] J. Oh, S. Park, H. Yu, M. Song, and S.-T. Park. Recommendation based on personal popularity tendency. In *ICDM*, pages 507–516, 2011.

[8] F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor. *Recommender Systems Handbook*. Springer, 2011.

[9] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Riedl. Analysis of recommendation algorithms for e-commerce. In *ACM EC*, pages 158–167, 2000.

[10] J. B. Schafer, J. A. Konstan, and J. Riedl. Recommender systems in e-commerce. In *ACM EC*, pages 158–166, 1999.

[11] C.-N. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen. Improving recommendation lists through topic diversification. In *WWW*, pages 22–32, 2005.

STERS: A System for Service Trustworthiness Evaluation and Recommendation based on the Trust Network

Yasha Wang[1,2,3], Jiangtao Wang[1,2], Yuxing Teng[1,2], Junfeng Zhao[1,2,3]
1 School of Electronics Engineering and Computer Science, Peking University, Beijing 10087, China
2 National Engineering Research Center of Software Engineering, Peking University, China
3 Beijing Beida Software Engineering Development Co., Ltd., China
{wangys, wangjt10, yxteng, zhaojf}@sei.pku.edu.cn

***Abstract*—Along with the rapid development of the Internet, more and more web services can be found and used. However, service trustworthiness is one of the most critical factors to keep the confidence of users in using them. In this paper, we propose a novel approach to evaluate the trustworthiness of web services using the Trust Network and give a mechanism for service recommendation in consideration of the users' different preference and perspectives. We adopt the Trust Network to estimate the important degree of subjective evidence from different resources and filter the false or malicious evidence. Then we calculate the trustworthiness of web services according to both of subjective and objective evidence. In addition to this, we put forward a preference template to recommend the proper service to the users according to the users' requirements. Finally, we develop a system called STERS to evaluate the service trustworthiness and recommend the service. Experiments conducted on a large-scale real-world dataset show that our method can effectively evaluate the trustworthiness of web services, which helps users to select and use them.**

Keywords-Web Service trustworthiness; Trust Network; QoS

I. *INTRODUCTION*

Along with the rapid development of the Internet, abundant web services can be found and used. However, since most web services are provided and/or hosted over the unreliable Internet, a question arises that how to select web services to keep the users' satisfaction and confidence of the application in which the web services are invoked? Trustworthiness evaluation of web services and trustworthiness based web service selection is a feasible solution to the above question.

Many approaches have been proposed to evaluate the trustworthiness of web services [1]. For example, E.M. Maximilien and M.P. Singh [2] present an automatic web service trustworthiness evaluation method using decision theory and ontologies.

However, there are still several points can be improved. First, according to our literature review, there are two kinds of evidences of trustworthiness: objective evidences including QoS attributes like response time and availability, and subjective evidences like reputation [3] [4]. Most of previous approaches rely on only one kind of evidences, while it is more reasonable to take both kinds into consideration [10] [5]. Second, most of the existing works, such as [1] and [2], just consider the subjective evidence from different source equally important. Considering that the subjective evidences are given based on personal experiences and perspectives, we think that the importance of these evidences should be consider differently according to the trust relationship among users. Third, we should recommend the services to the users according to their different preferences of different service properties.

The Trust Network [6] is adopted to model the trust relationship among users in this paper. The trust between the users in the Trust Network means: 1) trust each other's evaluation (i.e. the evidence) of the service; 2) trust the individual that is trusted by the other users in the Trust Network. The Trust Network is a directed graph, in which a node represents a user, an edge represents the trust relationship between two connected users, and the weight of the edge represents the degree of the trust.

In this paper, we propose a novel approach to evaluate the trustworthiness of web services using the Trust Network and give a mechanism for service recommendation in consideration of the users' different preference. We adopt the Trust Network to estimate the important degree of subjective evidence provided by different users. In addition to this, we filter false and malicious subjective evidence using the Trust Network to make more rational use of the evidence. After dealing with the subjective evidence, we can calculate the trustworthiness of web services. Based on it, we put forward a preference template to recommend the proper service to the users according to the requirements of the users. Finally, we develop a system called STERS (System for Service Trustworthiness Evaluation and Recommendation) to evaluate the service trustworthiness and recommend the service.

To study the effectiveness of our approach, we conduct several experiments on a large-scale real-world dataset. The experimental results show that: (1) with the evidence, our method can effectively evaluate the trustworthiness of web services; (2) the evaluated trustworthiness results are useful to help users to select and use web services.

The rest of this paper is organized as follows: Section II analyzes the related works. In Section III, we illustrated the approach for service trustworthiness evaluation based on the Trust Network. The service recommendation mechanism is introduced in Section IV. The evaluation of the trustworthiness evaluation approach is described in Section V. Finally, Section VI concludes the paper.

II. RELATED WORK

Trustworthiness is of paramount importance for developers to discover, select, and compose Web services [4].

G. Jennifer [7] made a lot of research on the Trust Network via using the data from the web site for the movie recommendation.

Various approaches have been proposed to evaluate the trustworthiness of web services. E.M. Maximilien and M.P. Singh [2] present an automatic web service trustworthiness evaluation method using decision theory and ontologies. Both the objective evidence and subjective evidence are important in web services trustworthiness evaluation. However, most of previous approaches leverage only one kind of evidence while it is more reasonable to leverage both [5]. In this research, both kinds of evidence are collected and used to evaluate a service's trustworthiness.

III. SERVICE TRUSTWORTHINESS EVALUATION BASED ON THE TRUST NETWORK

According to the previous work in [3] [8], we can collect abundant evidence, and evaluate the trustworthiness of services based on them. In order to distinguish the importance of evidence, we adopt the Trust Network to manage different kinds of trust relationships among entities while evaluating trustworthiness. To differentiate the importance of evidences, we assign the higher weights to the more important ones.

A. The structure of the Trust Network

In this paper we regard each user as the node in the Trust Network and the users comprise $A = \{a_1, a_2 \ldots a_n\}$. The trust between users makes trust matrix *Trust* that represents as the Cartesian product of the users *A*.

$$A \times A = \{\langle a_i, a_j \rangle \mid a_i, a_j \in A\}$$

The value of $\langle a_i, a_j \rangle$ in the trust matrix *Trust* is an edge with weight $trust_{i,j}$ between user *i* and user *j*. The $trust_{i,j}$ is called trust value from i to j, it represents the degree of trust and has the value of an integer of 0 to 100 ($0 \le trust_{i,j} \le 100$).

We connect all the services $O = \{o_1, o_2 \ldots o_n\}$ and their trustworthy attributes $P = \{p_1, p_2 \ldots p_n\}$ with users. Another matrix Score is used to represent the user's subjective evaluation of the trustworthy attribute of a certain service.

$$A \times O \times P = \{\langle a_i, o_j, p_k \rangle \mid a_i \in A, o_j \in O, p_k \in P\}$$

We use integer $Score_{i,o,k}$ ($0 \le Score_{i,o,k} \le 100$) to represent the value of $\langle a_i, o_j, p_k \rangle$, which is the value of subjective evaluation for the trustworthy attribute p_k that the user a_i assigned to the service o_j. The $Score_{i,o,k}$ will has a value greater than 0 only if the user a_i has used the service o_j, otherwise the $Score_{i,o,k} = 0$.

B. The initialization the Trust Network

There are two steps to initialize the Trust Network. In step one, the trust values of already registered users are initialized. In step two, the initial trust value of each new registering user is calculated and assigned.

The initialization in the step one is very simple, we just assign all the users' trust value in trust matrix to 50 ($0 \le trust_{ij} \le 100$).

The initialization in the step two is more complicated. Whenever a new user registers, a new node is added in the Trust Network. We assume that the trust value between those old nodes in the Trust Network has been completely evolved, which means the convergence condition of the iteration for evolving the trust values between old nodes has been met. We adopt two strategies to assign the initial trust value of the new node to the other old nodes take advantage of those completely evolved old trust values.

1) The Average Strategy.

The trust value of the new user *a* to the old user *b* is assigned as the following formula.

$$Trust_{a,b} = \frac{\sum Trust_{i,b}}{N}$$

In which $Trust_{i,b}$ is the trust value of the user *i* to the user *b*, *N* is the total number of users.

2) The Masses Strategy

For a given service *s*, many users provide evidences about its trustworthiness. An evidence is an evaluation score of the trustworthiness of *s* in the perspective of a given user. The average of all scores is considered as the public opinion on the trustworthiness of *s*.

$$Trust_{a,b} = \frac{w_{i,b} \sum Trust_{i,b}}{N} \qquad (1)$$

In which $w_{i,b}$ is the weight of the trust value of user *i* to user b, N is the total number of users.

C. The evolution of the Trust Network

With the increasing amount of evidence, we should adjust the trust values in the Trust Network continuously. We call this adjustment as the evolution of the Trust Network.

The developing of the Trust Network can be calculated by the single-step Markov model as below:

$$t_{(n)} = k \cdot f(t_{(n-1)} / k, tag, weight) \qquad (2)$$

$t_{(n-1)}$ is the trustworthy value calculated by the last time. *k* is the constant equal to the maximum value of the range of trustworthiness value. *tag* is a variable parameter and the value is *{-1,1}*, in which *-1* represents the trust reduced and *+1* is the increase of the trust. *weight* ($0 \le weight \le 1$) is the changing value of the trustworthiness. *f* can be computed by the

following formula, in which x represent $t_{(n-1)}/k$, t represent tag, w represent weight in formula (2) :

$$f(x,t,w)=\begin{cases} x(1+t\cdot w) & \text{If } t<0 \\ x+(1-x)t\cdot w & Else \end{cases} \quad (3)$$

From the formula we can know that under the same amplitude of variation *w*, *t* has different value means:

If $f_{t<0}(x,t,w) > f_{t>0}(x,t,w)$, this shows that the user's trust value is reduced due to a malicious behavior. The value needs more evolvement iterations to restore to the original value. Thus it can monitor the truth of user's evaluation behavior.

In order to calculate the variation *w*, we proposed two policies, namely "Mass Trust Policy (MTP)" and "Similar Trust Policy (STP)".

For a given user j, $w=R(Score_{j,o}, Score\_strategy)$,in which $Score_{j,o}$ is the latest score of user *j* to the service *o*, and $Score\_strategy$ has different means according to the two strategies. In the MTP $Score\_strategy$ is the average score of all the users that evaluate service *o*. In the STP, $Score\_strategy$ is the score of the user's evaluation of the service *o*. *R* is the degree of correlation of the two scores. As the evaluation of the user *a* to service *o* is a vector $(s_{a,1}, s_{a,2}, \ldots, s_{a,|p|})$, in which *p* is the set of the service's trustworthy attributes, *R* can be gained by calculating the cosine of the angle between the two vectors. The formula is as follows:

$$R(S_a,S_b)=\frac{\sum_{j=1}^{|p|} s_{a,j} * s_{b,j}}{\sqrt{\sum_{h=1}^{|p|}(s_{a,h})^2 * \sum_{k=1}^{|p|}(s_{b,k})^2}} \quad (4)$$

The MTP is to identify those specific users whose evaluations are different with what the most users had. The STP is to find identical users by analyzing different users' evaluations on the same service and build higher trust level among them.

D. Turstworthness Evaluation

We can calculate the trustworthiness of user *i* to the service *o* based on the evaluation of the user *i* to the service *o*, the trust among users and their similarity. The trustworthiness is also a multi-dimension vector where each dimension is an attribute in the service trustworthiness model that given in [3].

$$Belief_{i,o,k}=\frac{\sum_{1\le j\le n} w_{i,j}\cdot T_{j,o,k}\cdot K}{\sum_{1\le j\le n} w_{i,j}\cdot K} \quad (5)$$

$Belief_{i,o,k}$ represents the trustworthiness value of user *i* to the service *o* on the dimension *k*. $w_{i,j}$ varies according to the evaluation of dimension *k* which is either the subjective evidence or the objective evidence. *n* is the number of total users who used service *o*. As we know from section C, subjective evidence can make the Trust Network derive to a more accurate estimated value. Instead the objective evidence is not generated by the users' subjective evaluation, thus $w_{i,j}$ can be any arbitrary constant. As we can see from the following formula, $w_{i,j}$ will be the degree of trust between the user *i* and *j* when the *k* is subjective evidence, otherwise $w_{i,j}$ takes the constant 1. $T_{j,o,k}$ is the QoS value of user *j* to service *o* on the dimension *k*. *K* is the number of that user *j* has used service *o*.

$$w_{i,j}=\begin{cases} Trust_{i,j} & k \text{ is a subjective evidence} \\ 1 & k \text{ is an objective evidence} \end{cases} \quad (6)$$

We can adopt following computing policies when some special situations occur.

1) Believe in yourself.

Because the user uses certain services often and is familiar with them. In such circumstance, w_{ij} is degrading to 0

2) Believe in the most.

Because the user is unfamiliar with certain services, the historical majority evaluations must be used. The majority's $w_{i,j1}$ is effective and the rest $w_{i,j2}$ is 0.

$$\Diamond_{i,o}^{majority}=\max\left(\frac{T_{i,o,k}(\{v_k\})}{T_{i,o,k}(SVB)}\right) \quad (7)$$

SVB is integer and the range is 0≤SVB≤100. The formula can calculate the value V_k that is majority's average score on the dimension *K*. Thus we can find who scores V_k on the dimension *K* and set these users' trust value $w_{i,j}$ effectively.

3) Believe in the experienced.

In the formula of $Belief_{i,o,k}$, *K* represents the times that the user had used the service. Under certain conditions, such as the user believe other experienced users, the evaluation of the user *B* is more convinced than *A's* if *B* used the services for *N (N>1)* times while *A* is just once.

IV. User Preference and Service Recommendation Mechanism

There are many attributes to describe the trustworthiness of service in the service trustworthiness model.

The model of the user preference template is the same as the service QoS model in [3]. It contains the satisfaction degree, executing time, performance compliance, function compliance, etc. On each dimension has a user defined score from 1 to 100. In the template, the users can assign the weight to every trustworthiness attributes and calculate the Final value of service trustworthiness via the trust value $Belief_{i,o,k}$. The formula is as follows:

$$Final_{i,o}^{Trust} = \sum_{i=1}^{K} Belief_{i,o,k} \times pref_{i,k} \qquad (8)$$

In which $Belief_{i,o,k}$ is the trust value that the user *i* to service *o* on the dimension *k,* it is calculated by the formula in section IV . $pref_{i,k}$ is the weight on the dimension *k*. *K* is the number of service trustworthy attributes.

Based on the approach we put forward, we develop the STERS system to recommend the services to the user according to the result of the value of $Final_{i,o}$ in a sequence of high to low. The system will evaluate the service after the user finished using it and feedback the results to the Trust Network. Thus the process of finding a required service is finished.

V. EVALUATION

In order to verify the validity of the approach we proposed for evaluating services trustworthiness, we compare 3 approaches.

Approach 1: Service trustworthiness is gained by simply calculating the mathematical average of all evidences without considering their importance degree.

Approach 2: First trusted users are selected by leveraging the Collaborative Filtering (CF) method, and then the service trustworthiness is gained by calculating the mathematical average of evidences given by the selected users.

Approach 3: Service trustworthiness is gained by using the Trust Networks based approach proposed in this paper.

We have collected experiment data from August 1st, 2011 to April 1th, 2012 and the amount of data is over 900 sets per service. We select the data of CDYNE IP2Geo[1] web service. The curves of the Estimated Accuracy of the three strategies are shown in Figure 2. In which the AV represents the approach 1, CF is approach 2 and TN is approach 3.

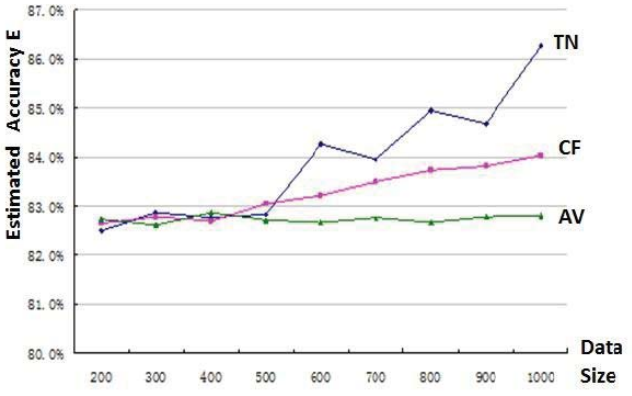


Figure 2. The evaluation results and comparison

[1] http://ws.cdyne.com/ip2geo/ip2geo.asmx?wsdl

From the figure 2, there is a little difference in the Estimated Accuracy E among the three approaches from August 1st, 2011 to November 1st, 2011. However, as time goes by, the Estimated Accuracy E of the Trust Networks and CF method is gradually improved. But the improvement of CF approach is not as significant as that of TN. To April 1st 2012, the Estimated Accuracy E of the Trust Networks is 2.5% higher than the CF's and 3% higher than the average method. The Estimated Accuracy E of the TN and the CF is continually increasing, while that of AV was stably stayed in the range of 82% to 83%.

VI. CONCLUSION

Trustworthiness is of paramount importance to help users to discover, select, and compose Web services in the computing paradigm of Service-Oriented Computing. In this paper, we proposed a novel approach to evaluate the trustworthiness of web services using the Trust Network, give a mechanism for service recommendation in consideration of the users' different preference. We also develop a system called STERS to collect evidence, evaluate the service trustworthiness and recommend services.

ACKNOWLEDGMENT

This work is supported by the Key National Science & Technology Specific Projects under Grant NO. 2011ZX01043-001-002, and High-Tech Research and Development Program of China under Grant No. 2013AA01A605.

REFERENCES

[1] Y. Wang, J. Vassileva, "A review on trust and reputation for web service selection," 27th International Conference on Distributed Computing Systems Workshops (ICDCSW'07), pp. 25-32. 2007.

[2] E.M. Maximilien, M.P. Singh. "Toward autonomic web services trust and selection." The 2nd international conference on Service oriented computing, pp.212-221. 2004.(15)

[3] M. Li, J. F. Zhao, L. J. Wang, S. B. Cai, and B. Xie. "CoWS: AnInternet-enriched and quality-aware Web services search engine".IEEE International Conference on Web Services (ICWS) , pp.419-427, 2011.

[4] L. Li, Y. Wang. "Trust evaluation in composite services selection and discovery." IEEE International Conference on Service Computing (SCC). pp. 482-485. 2009.

[5] Y. Zhang, Z. Zheng, and MR. Lyu, "WSExpress: a QoS-aware search engine for Web services," IEEE International Conference on Web Services, pp. 91-98.2010.(11)

[6] Matt Blaze, Joan Feigenbaum, Jack Lacy, "Decentralized Trust Management", Proceedings of the 1996 IEEE Symposium on Security and Privacy, p.164, May 06-08, 1996

[7] Jennifer Golbeck, Matthew Rothstein." Linking Social Networks on the Web with FOAF: A Semantic Web Case Study". Proceedings of the Twenty-Third Conference on Artificial Intelligence (AAAI'08)

[8] M. Li, J. Jin, and J. F. Zhao. A propositional logic-based and evidence-rich Web services trustworthiness evaluation framework. Software Trustworthiness (SoTrust 2011), Workshop of International Conference on Software Reuse (ICSR 2011), pp.26-33, 2011

Towards a Network Ecology of Software Ecosystems: an Analysis of two OSGi Ecosystems

Klaus Marius Hansen and Konstantinos Manikas
University of Copenhagen
Department of Computer Science (DIKU)
Copenhagen, Denmark
{klausmh,kmanikas}@diku.dk

***Abstract*—"Software ecosystems" are gaining importance in commercial software development; the iPhone iOS and Salesforce.com ecosystems are examples of this. In contrast to traditional forms of software reuse, such as common platforms or product lines, software ecosystems have a heterogeneous set of actors sharing and collaborating over one or more technological platforms and business model(s) that serve the actors. However, little research has investigated the properties of actual software ecosystems.**

In this paper, we present an exploratory study of software ecosystems using the formalizations and metrics of the "network ecology" approach to the analysis of natural ecosystems. In doing so, we mine the Maven central Java repository and analyze two OSGi ecosystems: Apache Felix and Eclipse Equinox. In particular, we define the concept of an ecosystem "neighborhood", apply network ecology metrics to these neighborhoods (including a keystone index that identifies the importance of elements in the ecosystem), and compare the ecosystems.

***Index Terms*—software ecosystems; dependency structure; dependency graphs; network ecology**

I. Introduction

Software ecosystems (SECOs) arguably present an effective way of constructing large software systems on top of a software platform by composing components developed by actors internal and external to the organization developing the platform [1]. In this setting, software engineering also takes place outside the traditional borders of software companies to a group of companies, private persons, or other legal entities.

This differs from traditional outsourcing techniques in that the organization developing the platform does not necessarily own the software produced by contributing actors and does not hire the contributing actors. All actors, however, coexist in an interdependent way, an example being the iOS ecosystem in which Apple provides a platform for applications and review of applications in return for a yearly fee and 30% of revenues of application sale[1]. This is a parallel to the natural ecosystems where the different members of the ecosystems (e.g., the plants, animals, or insects) are part of a food chain where the existence of one species depends on the rest.

While examples of successful software ecosystems exist, little empirical research in real software ecosystems exists [2]. In this paper, we study the software ecosystem created around Apache Maven[2], a build automation tool. A characteristic of Maven is that it automatically downloads project dependencies from online repositories. One of the main repositories that Maven consults is the Maven central repository[3], Maven Central, from which we have obtained the repository artifacts' metadata. In particular, we investigate the subset of Maven central that relates to these Apache Felix and Eclipse Equinox OSGi software ecosystems [3].

In the analysis of the ecosystems, we take inspiration from the analysis of natural ecosystems as graphs through "network ecology" [4]. In particular, we investigate software ecosystems using metrics from network ecology. Furthermore, we apply the PageRank [5] algorithm to the structure of software ecosystems.

Our research in this paper is exploratory; the underlying research questions being:

- How can metrics and formalizations of the "network ecology" approach to natural ecosystem analysis provide insight into the structure of software ecosystems?
- How can we analyze and compare different software ecosystems that may be subsets of a wider software ecosystem?

While we do not answer the questions fully, this paper provides initial insight into the answers.

II. Background

A. Apache Maven

Apache Maven [6] is a project management tool, primarily used for Java projects. A central concept in Maven is the "Project Object Model" (POM) that models a project among others allows Maven to build, package, upload to a remote repository, and generate web pages. The main uses of POMs are thus i) to enable building projects and ii) to enable resolution of dependencies in projects. POM files define *artifacts* (i.e., the software components of a project), *versions* of artifacts, and *groups* containing artifacts.

Maven Central[4] is a web service that collects artifacts (e.g., JAR files) and their description in form of POMs. We consider Maven Central as a software ecosystem (referenced to as the Maven ecosystem) with Apache Maven as the common technological platform. Because of the explicitness

[1] http://developer.apple.com/programs/ios/distribute.html
[2] http://maven.apache.org/
[3] http://search.maven.org
[4] http://search.maven.org/

of the dependencies, the Maven ecosystem allows the study of artifacts as a graph. In this paper, we study two ecosystems that are subsets of the Maven ecosystem: Apache Felix and Eclipse Equinox. Both ecosystems, as much as their artifacts, form part of the Maven ecosystem with potential interdependencies. In order to study each of the two software ecosystems, it is necessary to define their size and borders. For this reason we define and calculate the software ecosystem *neighborhood* for each sub-ecosystem.

Given the artifacts in the groups of the OSGi frameworks ("org.apache.felix" and "org.eclipse.equinox"), we follow their dependencies and find artifacts that depend on these artifacts to calculate an ecosystem neighborhood of increasing depth, where for each depth level we add the artifacts depending on artifacts already added or that artifacts already added depend on.

We calculate the neighborhood, A_d = NEIGHBORHOOD(S, d), of a set of artifacts, S, at distance d as follows based on a seed, $S := \{s_1, s_2, ..., s_n\}$, and distance, d:

1) $A_0 := S$
2) $A_{i+1} := A_i \cup \{a \mid \exists b \in A_i : a \text{ depends on } b\} \cup \{b \mid \exists a \in A_i : a \text{ depends on } b\}$

Concretely, we calculated NEIGHBORHOOD(F, i), where F is the set of artifacts in the Apache Felix group (`org.apache.felix`), $i = 0, 1, ...$ and NEIGHBORHOOD(E, i), where E is the set of artifacts in the Eclipse Equinox group (`org.eclipse.equinox`) , $i = 0, 1, ...$.

B. Network ecology

A common theme in ecology of natural ecosystems is a focus on interactions and relationships among living individuals, between predator and prey. "Network ecology" [4], models food webs as *networks* or *graphs* of species and interactions. The networks are often directed to indicate, e.g., energy flow between individuals (e.g., from prey to predator) and are sometimes weighted. Figure 1 shows a simplified example of a food web with one predator (A) and two prey (B and C). Network ecology applies both classical graph/network metrics (e.g., connectance) and novel metrics to food networks (e.g., keystone index); see Section III.

Of particular interest in ecosystems are *keystone species* that have a large effect on their environment compared to the number of members of the species in the environment. In the example, species A may be a keystone since it has effect on both B and C.

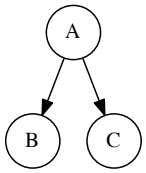


Fig. 1. Simple food web example

In our application of network ecology to the study of software ecosystems, we draw a parallel between species in a natural ecosystem and actors and software artifacts in software ecosystems. Species prey on other species and in this way energy flows from prey to predator. Actors produce software artifacts and software artifacts consume other software artifacts via usage dependencies. And just as the co-existence of species in a natural ecosystem determines the health of the natural ecosystem, the co-existence of actors and software artifacts in a software ecosystem determins the health of the software ecosystem. We thus apply energy flow analysis of natural ecosystems to dependency analysis in software ecosystems. In this our study, we analyzed software ecosystem actors only in terms of their technical products, i.e., the software artifacts, and thus our study is of software dependencies.

In our case, the species relationships from network ecology are usage dependencies among software artifacts and, thus, at runtime, control and data may flow from depender to dependee. Furthermore, our networks are always directed (in the direction of dependencies) and without weight (or equivalently with equal weight).

III. CHARACTERIZING NETWORK ECOLOGIES

In this section, we present the characteristics of network ecologies that we apply in our software ecosystem analysis. We divide the characteristics into *global* characteristics that we calculate for a complete network (Section III-A), and *local* characteristics that we calculate for individual nodes. Furthermore, we apply a page rank algorithm (Section III-C) to the network to compare to network ecology characteristics.

A. Global network characteristics

Traditional graph metrics (e.g., number of nodes and edges) can be applied to networks of ecologies. In particular, we consider connectance, $C = \frac{L}{N \cdot (N-1)}$, where L is the number of relationships in the network, dependencies in our case, and N is the number of nodes in the network, artifacts in our case. Connectance measures the proportion of all potential relationships that are present in the network. Since our network is directed and has no self-relationships, the maximum number of relationships is $N \cdot (N-1)$.

Applying the connectance metric to the Felix and Equinox ecosystems would provide a view on the connected the whole ecosystem is. This, apart from assisting in describing the ecosystem, can measure similarity of the ecosystems as far as connectance goes.

B. Local network characteristics

One of the basic properties of a node is the number of connections this node has to its neighboring nodes, the connection degree of the node. The *connection degree* D of a node i is $D_i = D_{in} + D_{out}$ where D_{in} is the number or nodes depending on node i and D_{out} the number of nodes that node i depends on. The degree of a node is an elementary property that shows how much a node is interacting with its neighboring nodes. Similarly, if we take into consideration the dependency

direction, the *orientation* of a node is $D'_i = D_{in} - D_{out}$. This metric may be used to measure the interaction a node has with its neighboring nodes in directed graphs. If an artifact has a negative orientation, the number of artifacts that it depends is larger than the number of artifacts that depends on it.

The *clustering coefficient* gives further insight into how nodes connect in the graph. The clustering coefficient of a node shows how close this node is to becoming a complete graph with its neighbors. The clustering coefficient of a node i is as follows:

$$CC_i = \frac{L_i}{N_i \cdot (N_i - 1)}$$

where L_i is the number of links between the neighboring nodes N_i of node i. The *global clustering coefficient* for the whole graph is calculated from the clustering coefficient of each node:

$$\bar{C} = \frac{1}{n} \sum_{i=1}^{n} C_i$$

We use the clustering coefficient calculation in our study to analyze the dependencies of the artifacts in each ecosystem. Additionally, we use the calculation to of compare the two ecosystems.

Jordán et al. introduced a reliability-theoretic approach to characterizing keystone indices [7]. They base their keystone index on the concept of a *bottom-up* keystone index and a *top-down* keystone index. The bottom-up keystone index of a species i, $K_b(i)$, is a measure of secondary extinctions because of bottom-up effects if the species i is removed. Here, "bottom-up effects" refer to effects going from prey to predator if the species is removed.

The bottom-up index is defined as:

$$K_b(i) = \sum_{j \in P(i)} \frac{1}{m_i(j)} \cdot (1 + K_b(j))$$

where $P(i)$ are the direct predators of i, $m_i(j)$ the number of direct prey of $P(i)$, and $K_b(j)$ is the bottom-up keystone index for j. The top-down index, $K_t(i)$ is calculated by reversing the species relationships and calculating an index similar to above. Finally, the keystone index is calculated as:

$$K(i) = K_b(i) + K_t(i)$$

Naturally, if species i have no direct predators, their bottom-up keystone index $K_b(i)$ is zero and the keystone index is calculated only by the top-down approach. For the simple example in Figure 1, the keystones indices are $K(A) = 2$ and $K(B) = K(C) = 0.5$.

We adapt the keystone index in the study of the two software ecosystems where instead of a network of preys and predators we have an artifact dependency graph. In this case, the keystone index indicates the importance of an artifact, i.e. what effect the removal of that artifact would have to the software ecosystem.

C. PageRank

The PageRank algorithm [5] calculates the weight of a web page p_i by summing the weights of each page pointing to p_i normalized by the number of links pointing out of that page and adjusted by a dumping factor constant d. PageRank can be explained using the behavior of a hypothetical web surfer who either i) chooses to follow a link at random from a web page with probability d, or, ii) at random chooses any page to go to with probability $1 - d$. The PageRank of a web page p_i, $P(p_i)$ is then the probability of being on that web page that this iterative procedure converges to. The PageRank has the following property:

$$P(p_i) = \frac{(1-d)}{N} + d \cdot \left(\frac{P(p_1)}{C(p_1)} + ... + \frac{P(p_n)}{C(p_n)} \right)$$

where p_1 to p_n are the pages pointing at page p_i, $C(p_i)$ the number of links going out of page p_i and N the total number of pages.

Apart from ranking webpages, PageRank, can be applied to rank nodes in directed graphs. In this study we apply PageRank to estimate weights of the artifacts in the studied ecosystems, with the page links implying artifact dependency. In that sense, an artifact with high PageRank would either have a high number of artifacts or artifacts with high PageRanks depending on it. Such an artifact, will arguably have a large effect on the ecosystem compared with the total number of artifacts. In doing so, we follow an approach close to Bhattacharya et al. [8] who define a "NodeRank" for graph-based representations of software based on PageRank.

Our hypothesis is, that the artifacts with high PageRank would also be "keystone species" in the software ecosystem. For the PageRank ranking of both Apache Felix and Eclipse Equinox software ecosystems, we used a fixed damping factor, d, of 0.85.

IV. Data gathering and processing

We processed the set of 286,922 POM XML files of Maven Central as of 2012-01-24. In the processing, we identified an artifact by group id and artifact id, i.e., we disregarded version numbers of artifacts for reasons of scalability. For each POM file, we output the group id and artifact id and for each dependency declared, we output the relationship between the processed artifact and its dependency.

In total, we extracted 155,127 unique artifacts in 5,676 groups and 495,020 unique dependencies among these artifacts. A number of POM files (36) were empty or contained errors and a number of dependencies (93,455) were not found in the repository.

For the complete set of artifacts and dependencies, we calculated the neighborhoods of the groups org.apache.felix and org.equinox.felix. Table I shows this together with summary statistics of the neighborhoods.

V. Results

In the following, we present the results of applying the metrics of Section III to the neighborhoods of Felix and Equinox. Table I shows the global metrics.

	Felix			Equinox		
Distance	# artifacts	# dependencies	connectance	# artifacts	# dependencies	connectance
0	25	29	0.048333	19	7	0.020468
1	110	138	0.011510	80	69	0.010918
2	7772	8036	0.000133	635	679	0.001687
3	13638	25414	0.000137	9481	10037	0.000112
4	15629	28222	0.000116	14625	25082	0.000117
5	15923	28649	0.000113	15797	26778	0.000107
6	15983	28725	0.000112	15967	26983	0.000106
7	15984	28726	0.000112	15992	27008	0.000106
8	15984	28726	0.000112	15992	27008	0.000106

TABLE I
SUMMARY STATISTICS OF CALCULATED NEIGHBORHOODS

Analyzing the neighborhoods for both ecosystems, we notice that the number of artifacts and dependencies remain the same for distances greater than seven. This implies that the minimum distance of any two indirectly connected artifacts in the Felix or Equinox ecosystems is not more than seven. Therefore, the neighborhood distance of seven is the border of both ecosystems within the whole Maven ecosystem.

A. Network ecology metrics

In the figure 2, we show a log-log histogram for Felix for connection degree $D < 100$, which is the majority of the artifacts (for Felix 15,956 out of 15,984 and Equinox 15,965 out of 15,992). For the Felix degree distribution, the maximum count appears in degree 1 with 7,950 artifacts while the highest degree is 3932 with count of 1 artifact. The artifacts with degrees 1, 2 and 3 account for 80% of the total number of artifacts. For the Equinox group, degree 1 has the maximum count of artifacts, 8,235, while the highest degree is 3,519. Similarly, degrees 1, 2 and 3 account for 82% of the total artifacts. In both groups, the artifact with the highest degree is *junit*. The degree for the artifacts of Felix has an average of 3.59 and a standard deviation of 1,097.74, while Equinox has an average of 3.38 and a standard deviation of 904.63. From the degree distributions, we can verify the similarity in the dependency relations noted for the connectance results of the high distance neighborhoods, with Felix having slightly more dependencies.

Tables II and III display the artifact that have top orientation rankings for Felix and Equinox. In both groups, the top orientation ranking is almost identical (the order for "testing" and "maven-plugin-api" changes) for the top connection degrees. Taking into consideration that the average orientation in both groups is zero, the orientation metrics reveal a tendency in the high ranked connection degree artifacts to have close to zero dependencies to other artifacts, i.e. equally high orientation. This supports the idea of certain artifacts being the center of a cluster or the ecosystem with all the rest of the artifacts depending on them. This enhances the interest of studying the keystone and PageRank metrics.

The clustering coefficient analysis shows that there is a low number of nodes that had a non-zero clustering coefficient (20 out of the 15,984 artifacts for Felix and 4 artifacts from the total 15,992 for Equinox). Out of the 20 non-zero clustering

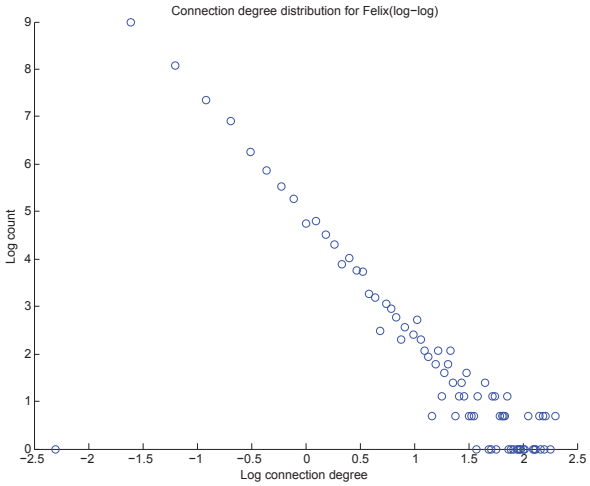


Fig. 2. Connection degree distribution for Felix (log-log;$D < 100$)

org.apache.felix			
Group id	artifact id	degree	orientation
junit	junit	3932	3930
javax.servlet	servlet-api	770	770
org.slf4j	slf4j-api	489	489
org.easymock	easymock	368	368
org.mockito	mockito-all	320	320
commons-io	commons-io	305	305
org.scala-lang	scala-library	235	235
stax	stax-api	234	234
xerces	xercesImpl	202	198
org.apache.maven	maven-plugin-api	196	196
org.testng	testng	198	190

TABLE II
TOP NODES ACCORDING TO THEIR CONNECTION DEGREE AND ORIENTATION FOR ORG.APACHE.FELIX

coefficients for Felix, only 4 artifacts do not belong to the Felix group, while all 4 of the resulting artifacts for Equinox belong to the Equinox group. We may interpret this as Felix having a higher level of reciprocity and thus, according to [9], being a more *open* software ecosystem than Equinox. However, the findings are not adequate to fully support this statement. Table IV shows the top clustering coefficient artifacts for the Felix group and Table V shows the same for the Equinox group . The global clustering coefficient is 0.0002047 for Felix and

org.eclipse.equinox			
Group id	artifact id	degree	orientation
junit	junit	3519	3517
javax.servlet	servlet-api	646	646
log4j	log4j	568	558
org.slf4j	slf4j-api	364	364
org.easymock	easymock	354	354
commons-logging	commons-logging	335	331
org.eclipse.core	runtime	272	268
org.scala-lang	scala-library	263	263
stax	stax-api	262	262
commons-lang	commons-lang	231	231

TABLE III
TOP NODES ACCORDING TO THEIR CONNECTION DEGREE AND ORIENTATION FOR ORG.ECLIPSE.EQUINOX

0.0000377 for Equinox.

The fact that both ecosystems have little clustering is in accordance with the numbers from the previous section where the majority of the artifacts (80% for Felix, 82% for Equinox) had a sum of 3 or less incoming and outgoing dependencies.

org.apache.felix		
Group id	artifact id	CC
org.apache.felix	org.apache.felix.ipojo.handler.white.board.pattern	0.5
org.apache.felix	org.apache.felix.ipojo.handler.extender.pattern	0.5
org.apache.felix	org.apache.felix.ipojo.handler.extender	0.5
org.apache.felix	org.apache.felix.ipojo.arch	0.5
org.apache.felix	org.apache.felix.ipojo.handler.temporal	0.25
org.apache.felix	org.apache.felix.ipojo.composite	0.25
org.apache.felix	org.apache.felix.ipojo.handler.whiteboard	0.167
org.apache.felix	org.apache.felix.scr.generator	0.119
org.apache.felix	org.apache.felix.ipojo.manipulator	0.119
org.apache.felix	org.apache.felix.ipojo.arch.gogo	0.1

TABLE IV
TOP CLUSTERING COEFFICIENTS FOR ORG.APACHE.FELIX

org.eclipse.equinox		
Group id	artifact id	CC
org.eclipse.equinox	preferences	0.5
org.eclipse.equinox	app	0.0833
org.eclipse.equinox	registry	0.01818
org.eclipse.equinox	common	0.00215

TABLE V
TOP CLUSTERING COEFFICIENTS FOR ORG.ECLIPSE.EQUINOX

Table VI and Table VII show the top 10 keystone indices (cf. Section III-A). For both Eclipse and Equinox, JUnit and Hamcrest (upon which JUnit depends) are outliers. For Felix, 3,931 artifacts depend on JUnit (6 on Hamcrest) and for Equinox, 3,518 artifacts depend on JUnit (and 6 on Hamcrest). Further down the list, and of interest, Eclipse-related artifacts appear for both Felix and Equinox, indicating that these are important in both ecosystems.

Scatter plotting of the keystone indices on a log-log scale (not shown), indicates visually that the keystone indices follow a power law distribution.

org.apache.felix		
Group id	artifact id	K
org.hamcrest	hamcrest-core	4285.85
junit	junit	4262.06
javax.servlet	servlet-api	662.97
org.slf4j	slf4j-api	483.07
xml-apis	xml-apis	396.62
org.easymock	easymock	317.17
commons-io	commons-io	288.47
org.eclipse.equinox	common	264.29
org.testng	testng	245.33
org.eclipse.core	runtime	227.88

TABLE VI
TOP KEYSTONE INDICES FOR ORG.APACHE.FELIX

org.eclipse.equinox		
Group id	artifact id	K
org.hamcrest	hamcrest-core	4085.94
junit	junit	4057.05
javax.servlet	servlet-api	616.57
log4j	log4j	445.58
org.slf4j	slf4j-api	430.02
org.easymock	easymock	336.70
org.eclipse.core	runtime	305.49
org.eclipse.equinox	common	273.20
commons-logging	commons-logging	259.43
xml-apis	xml-apis	251.65

TABLE VII
TOP KEYSTONE INDICES FOR ORG.ECLIPSE.EQUINOX

B. PageRank

Table VIII and Table IX show the top 10 PageRanks (cf. Section III-C). The Jaccard index (i.e., the size of the set intersection divided by size of the set union) of the sets of the top 10 artifacts for Eclipse Equinox with respect to keystone index and pagerank is high, 0.67, meaning that the sets are similar. For Apache Felix the Jaccard index is even higher, 0,82.

org.apache.felix		
Group id	Artifact id	$PageRank$
junit	junit	0.0904794430569
org.hamcrest	hamcrest-core	0.0773937500317
javax.servlet	servlet-api	0.0143641067466
org.slf4j	slf4j-api	0.0103344297076
xml-apis	xml-apis	0.00711548141087
org.easymock	easymock	0.00684332751987
commons-io	commons-io	0.00616812512782
org.testng	testng	0.00598292281005
org.mockito	mockito-all	0.00474775877824
org.eclipse.core	runtime	0.00460514855267

TABLE VIII
TOP PAGERANKS FOR ORG.APACHE.FELIX

VI. FUTURE AND RELATED WORK

Researchers have previously analysed networks and graphs in the context of software engineering, in particular on social networks [10]. One problem in this context has been incomplete data, something that we circumvent by obtaining a full copy of the Maven Central POMs.

org.eclipse.equinox		
Group id	Artifact id	$PageRank$
junit	junit	0.0862679466078
org.hamcrest	hamcrest-core	0.0739051502318
javax.servlet	servlet-api	0.0132943226829
log4j	log4j	0.00996111702211
org.slf4j	slf4j-api	0.00894363970498
org.easymock	easymock	0.00735198854011
org.eclipse.core	runtime	0.00676022114354
commons-logging	commons-logging	0.00574973950937
org.testng	testng	0.00536452952305
commons-io	commons-io	0.00486674036983

TABLE IX
TOP PAGERANKS FOR ORG.ECLIPSE.EQUINOX

We recently completed a systematic literature review of research of software ecosystems [2] in which we categorized research output according to the taxonomy of Shaw [11] and found little research on empirical models of software ecosystems. Yu et al. [12] studied symbiosis in software ecosystem as a parallel to symbiosis in natural ecosystems, but no research to our knowledge applied concepts from network ecology to software ecosystems.

Future work includes the analysis of all of the Maven POM data as a network rather than looking at subsets such as Apache Felix and Eclipse Equinox. Similarly to adding additional artifacts in the analysis, an analysis that takes versions and time series into account should also be performed. We made the simplifying assumption of abstracting away version numbers for reasons of scalability of the analysis which may impact the validity of our results. Further, a goal of our research is to find measures of health for software ecosystems, something for which the identification of keystones is only one step towards. In the future, we will also evaluate the use of social network analysis to analyze the artifact dependencies.

Additionally, in this paper, we compared the finding from a keystone index algorithm to the PageRank ranking using a damping factor d of 0.85 for the PageRank algorithm. Different values of the damping factor could enhance the precision of the results.

Another interesting extension is to consider series and dynamics of Maven POM data over time. In particular, Jordán and Scheuring [4] also consider dynamics in their work on network ecology. Extending the network ecology analysis to other ecosystems such as the Eclipse Equinox ecosystem with source code or other open source repositories [13] would also be of interest.

VII. SUMMARY

In this paper, we presented an exploratory study of the Apache Felix and Eclipse Equinox software ecosystems. The Maven central repository stores artifacts belonging to the ecosystems and we extracted descriptions and dependencies based on Project Object Model (POM) files. In total, we analyzed 286,922 POM files, extracting 155,127 artifacts with 495,020 dependencies for further analysis. We defined the neighborhoods for both Felix and Equinox identifying the neighborhood distance of seven as the fix point of both software ecosystems in the Maven ecosystem in terms of number of artifacts included.

For these neighborhoods of Felix and Equinox, we applied concepts from "network ecology" that examines natural ecosystems as networks of prey and predators. We calculated the clustering coefficient for each artifact and the global clustering coefficient for each neighborhoods and concluded that both of the neighborhoods exhibit little clustering. We furthermore defined and calculated a keystone index for the neighborhoods and demonstrated that this index appears to follow a power law distribution. Furthermore, we compared the keystone index to the PageRank algorithm and found that for high-ranked artifacts, there was a large overlap according to a Jaccard index.

We have thus tentatively pointed to the use of the keystone index as a way to identify keystone artifacts in a software ecosystem, artifacts that are of particular importance in the health of the ecosystem. Furthermore, by comparing the Felix and Equinox ecosystems, we demonstrated that it is possible to compare two software ecosystems in terms of size and dependency characteristics.

ACKNOWLEDGMENTS

This work was funded by the Net4Care project. We thank Sonatype for providing access to Maven Central POMs.

REFERENCES

[1] J. Bosch, "From software product lines to software ecosystems," in *Proceedings of the 13th International Software Product Line Conference*, ser. SPLC '09. Pittsburgh, PA, USA: Carnegie Mellon University, 2009, pp. 111–119.

[2] K. Manikas and K. M. Hansen, "Software ecosystems – a systematic literature review," *Journal of Systems and Software*, 2013, in press.

[3] The OSGi Alliance, "OSGi service platform core specification, release 5," http://www.osgi.org/Specifications, June 2012.

[4] F. Jordán and I. Scheuring, "Network ecology: topological constraints on ecosystem dynamics," *Physics of Life Reviews*, vol. 1, no. 3, pp. 139–172, 2004.

[5] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," *Computer networks and ISDN systems*, vol. 30, no. 1-7, pp. 107–117, 1998.

[6] T. O'Brien, M. Moser, J. Casey, B. Fox, J. V. Zyl, E. Redmond, and L. Shatzer, *Maven: The Complete Reference*. Sonatype, 2011. [Online]. Available: http://www.sonatype.com/books/mvnref-book/reference/

[7] F. Jordán, A. Takács-Sánta, and I. Molnár, "A reliability theoretical quest for keystones," *Oikos*, pp. 453–462, 1999.

[8] P. Bhattacharya, M. Iliofotou, I. Neamtiu, and M. Faloutsos, "Graph-based analysis and prediction for software evolution," in *ICSE 2012*. IEEE Press, Jun. 2012, pp. 419–429. [Online]. Available: http://dl.acm.org/citation.cfm?id=2337223.2337273

[9] J. te Molder, B. van Lier, and S. Jansen, "Clopenness of systems: The interwoven nature of ecosystems," in *Third International Workshop on Software Ecosystems (IWSECO-2011)*. CEUR-WS, 2011, pp. 52–64.

[10] R. Nia, C. Bird, P. Devanbu, and V. Filkov, "Validity of network analyses in open source projects," in *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*. IEEE, 2010, pp. 201–209.

[11] M. Shaw, "Writing good software engineering research papers," *Mini-tutorial for Proc ICSE" 03*, 2003.

[12] L. Yu, S. Ramaswamy, and J. Bush, "Symbiosis and software evolvability," *IT Professional*, vol. 10, no. 4, pp. 56 –62, july-aug. 2008.

[13] G. Robles, J. Gonzalez-Barahona, M. Michlmayr, and J. Amor, "Mining large software compilations over time: another perspective of software evolution," in *Proceedings of the 2006 international workshop on Mining software repositories*. ACM, 2006, pp. 3–9.

Revisiting the Performance of Weighted k-Nearest Centroid Neighbor Classifiers

Muhammad Rezaul Karim and Malek Mouhoub
Department of Computer Science
University of Regina, Canada
{karim20m,malek.mouhoub}@uregina.ca

Abstract

k-*Nearest Neighbor (KNN) is one of the most fundamental classification techniques. KNN relies on the distances of the samples to select* k *neighbors.* k*-Nearest Centroid Neighbor classification (KNCN) scheme, on the other hand, takes into account both the distances and the distribution of samples to improve the performance of KNN. In the past studies, with the help of two kernel functions, it was shown that assigning weights to the neighbors in KNCN further improve the performances of KNN based algorithms. In this study, we revisit the performance of Weighted* k-*Nearest Centroid Neighbor (WKNCN) method with various voting schemes and perform extensive comparison with other state-of-the-art KNN based algorithms. Unlike the previous studies, our experimental results show that weighted voting does not have any significant impact on the performance of KNCN method. To validate our claim, we design a new kernel for the WKNCN and perform statistical test on the experimental results. Our analysis with the various kernels also show that only well-designed distance based kernels like Inverse-distance kernel can exhibit comparable performance as the existing rank based kernels.*

1. Introduction

k-Nearest Neighbor (KNN) [1] is a simple non-parametric classification method. KNN is the primary choice when there is little or no prior knowledge about the distribution of the data. In KNN, classification is performed by a majority vote of the neighbors of the test data object, with the object being assigned to the class most common amongst its k nearest neighbors. KNN works very well when large numbers of training samples are available, but its performance is not retained when the training data is small. Over the years, researchers proposed various techniques to improve the performance of KNN [2, 5, 6, 7, 8, 10]. The *k*-Nearest Centroid Neighbor (KNCN) algorithm introduced by Sanchez et al. [2] is an extension to the KNN algorithm and is very effective when the training data is small. KNCN is based on the nearest centroid neighborhood concept proposed by Chaudhuri et al. [3]. KNN takes into account the nearness of the neighbors only; while KNCN takes into account both nearness and spatial distribution of the neighbors.

KNCN assigns equal weight to each centroid neighbor which is not always appropriate in classification. Weighted *k*-Nearest Centroid Neighbor (WKNCN) classification method is a recently proposed extension to the KNCN method and assigns different weights to different neighbors [4]. It was shown that WKNCN improves the performance of KNCN on six data sets. In WKNCN, kernel functions are used to assign weight to each neighbor. In [4], two kernel functions were proposed for this weighted version of KNCN algorithm. One kernel function uses the distance of a neighbor to compute its weight, while the other kernel assigns a weight based on the rank of the neighbor, in other words, the order in which the neighbor was selected. Unlike distance weighted KNN [5], both distance and rank based kernels were shown to be equally effective for WKNCN, which needs more investigation.

In this paper, we perform analysis on the existing distance and rank based kernels to shed more light on the importance of weighted voting for KNCN. To determine the appropriate kernel type, we also design and investigate the performance of a new distance based kernel. Our results and analysis show that assigning weights to the neighbors do not significantly improve the performance of KNCN. We also notice that a well-designed distance based kernel can exhibit comparable performance as the existing rank based kernels.

The rest of the paper is organized as follows: Section 2 provides an overview of KNCN. WKNCN method with existing kernels are summarized in section 3, while a new kernel function for the WKNCN is proposed in section 4. In section 5, the experimental settings are detailed, results are reported and analyzed. The paper concludes in section 6 with a summary of the work done.

2. KNCN

KNCN [2] is a good alternative to KNN algorithm and improves the performance of KNN. Classification in KNCN is a two step process. In the first step, the algorithm finds out the k nearest centroid neighbors of a test data object. In the next step, the object is being assigned to the class most common amongst its k neighbors. The basic idea of KNCN is to use the neighbors that are as close to a query pattern as possible, as well as, the neighbors that are distributed as geometrically around that pattern as possible. KNN takes into account the nearness of the neighbors only; while KNCN takes into account both nearness and spatial distribution of the neighbors.

Let $T = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$ be a set of points where $x_n \in \mathbf{R}^m$ is a m-dimensional feature space and $y_n \in \{c_1, c_2, c_3, ..., c_M\}$ is the class label for the training point x_n. Given a set of points T and a test pattern p, the KNCN classification scheme is defined as follows:

1. Find the k nearest centroid neighbors of the test pattern p.
2. Assign the test pattern p the class with a majority of votes among its k nearest centroid neighbors (resolve ties randomly).

Chaudhuri et al. proposed an iterative way of searching k centroid neighbors of p, which is as follows [3]:

1. The first centroid neighbor of p is its nearest neighbor, N_1.
2. The i^{th} centroid neighbor N_i where $i \geq 2$, is such that the centroid of this point and previously selected neighbors, N_1, N_2,..., N_{i-1}, is the closest to p.

The centroid of a set points $S = \{N_1, N_2, ..., N_q\}$ can be computed as follows:

$$S_c = \frac{1}{q} \sum_{i=1}^{q} N_i \quad (1)$$

3. WKNCN

KNCN classifier assigns equal weight to each of its k centroid neighbors. In classification, the more reliable neighbors of a test pattern should be assigned more weight than the unreliable neighbors. A training sample that has the same class as the test pattern is a reliable neighbor of the test pattern, while a noisy sample is an example of unreliable neighbor. KNCN algorithm equally treats the unreliable and reliable centroid neighbors. This results in poor classification accuracy and it can be improved by assigning different weights to different centroid neighbors [4].

WKNCN

S : a set of training samples
NCN : a set of k nearest centroid neighbors

Step 1: Compute distance to the test pattern p
  For each training sample $x_i \in S$, compute $dist(x_p, x_i)$, the distance to the test pattern p;
  $x_r = min(dist(x_p, x_i))$ {the training sample x_r with the minimum distance is the first centroid neighbor};

Step 2: Determine centroid neighbors
  $NCN = \emptyset$;
  $NCN = NCN \cup \{x_r\}$;
  $S = S - \{x_r\}$;
repeat
  For each sample $x_i \in S$, find the centroid C_i of samples in the set $NCN \cup \{x_i\}$;
  Choose the sample x_c such that C_c is the closest to p;
  $NCN = NCN \cup \{x_c\}$;
  $S = S - \{x_c\}$;
until k-1 centroid neighbors have been generated

Step 3: Compute weights for the centroid neighbors
  Compute the weight w_i of each centroid neighbor $x_i \in NCN$ using a kernel function;

Step 4: Predict the class
  Predict the class represented by the majority weighted vote of centroid neighbors;

Figure 1. Weighted k-Nearest Centroid Neighbor Classification

Gou et al. proposed WKNCN method to improve the performance of KNCN [4]. Unlike KNCN, in WKNCN, the class of a test pattern or data object is represented by the majority weighted vote of the k nearest centroid neighbors. Two kernel functions were proposed in [4] to assign weights to the centroid neighbors. The main idea was to assign more weight to the highly reliable centroid neighbors, while less weight to the less reliable centroid neighbors. WKNCN classification scheme is given in Figure 1.

In [4], one of the kernel functions was referred to as the *Heat kernel* and the other was referred to as the *Uniform kernel*. These kernels can be classified into two groups based on how weights are assigned to the neighbors: *distance based* and *rank based*. Kernels belonging to the first category assign weight based on the dissimilarity or the distance of a neighbor to the test pattern, while the kernels in the second category assign weight based on the rank or the order in which a neighbor is selected.

The Heat kernel is a distance based kernel and is given by:

$$w_i = exp(-\frac{||x_p - x_i||^2}{width}), \quad i = 1, 2, ..., k \qquad (2)$$

where $width = \frac{1}{k^2}\sum_{i=1}^{k} ||x_p - x_i||^2$ and i refers to the rank of each centroid neighbor. This kernel assigns a weight to a centroid neighbor that is related to its distance from the test pattern. The more the distance from the test pattern, the less the weight assigned. The Uniform kernel, on the other hand, is a rank based kernel and is given by:

$$w_i = \frac{1}{i}, \quad i = 1, 2, ..., k \qquad (3)$$

Here i refers to the rank of each centroid neighbor. The Uniform kernel assigns weights to the centroid neighbors in the order neighbors are computed, that is more weight to the first centroid neighbor, least weight to the last centroid neighbor. It does not take the into account the distance of a centroid neighbor to the test pattern. For this kernel, weight lies in the interval (0,1].

Gou et al. used six real data sets to perform experiments with both Heat and Uniform Kernels [4]. They reported that UHWKNCN and HWKNCN, the weighted versions of KNCN using Uniform and Heat kernel, respectively, outperform KNCN, KNN and distance weighted KNN [5] across all values of the neighborhood size, k. Improvement shown in that paper was significant for large values of k. Their results also show that both kernel functions have equal performances, which needs more investigation with diverse data sets with different distributions.

4. Inverse-distance Kernel for WKNCN

In this paper, we design a new kernel function for the WKNCN method and investigate its performance. This new kernel function is given by:

$$w_i = \frac{1}{1 + (dist(x_p, N_i))^2} \quad i = 1, 2, ..., k \qquad (4)$$

where i refers to the rank of each centroid neighbor and $dist(x_p, N_i)$ refers to the distance of the i^{th} centroid neighbor to the test pattern p. We refer to this kernel as *Inverse-distance* kernel.

Inspired by the inversion kernel for KNN described in [10], we investigate this kernel for the WKNCN. It is worth noting that the popular distance weighted kernel for KNN proposed in [5] cannot be applied to KNCN, as the distance of a closer nearest centroid neighbor may be larger than that of the farther one. Like Heat kernel, Inverse-distance is a distance based kernel. In this kernel, weight is based on the dissimilarity between the test pattern and a neighbor. The more the dissimilarity or distance, the less the weight assigned to a neighbor. More dissimilarity means that a centroid neighbor might not be of the same class of the test pattern. In this case, the centroid neighbor is unreliable and is given less weight. In this kernel, distance is squared to reduce the effect of the neighbor in the voting process.

Unlike Heat kernel, Inverse-distance does not take into account other neighbors distance. Heat kernel needs other neighbors distance to compute kernel width, which is used in the computation of weight of each centroid neighbor. For Inverse-distance kernel, weight lies in the interval (0,1]. If a centroid neighbor is exactly similar to the test pattern i.e. $dist(x_p, N_i) = 0$, weight assigned to the centroid neighbor is 1, the highest possible weight.

Table 1. Data Sets

Data Set	Instances	Classes	Features
Wine	178	3	13
Sonar	208	2	60
Glass	214	6	9
Spect	267	2	22
Spectf	267	2	44
Liver	345	2	6
Ionosphere	351	2	34
Libras	360	15	90
Wdbc	569	2	30
Pima	768	2	8
Vehicle	846	4	18
Vowel	990	11	12
Cardio	2126	3	21
Image	2310	7	19
Optdigits	5620	3	21

5. Experimentation

In this section, we report and analyze the results of our experiments for various algorithms. First, we introduce experimental settings and compare the performance of three kernels for the WKNCN method in terms of classification error. Then, we report the results for KNN and KNCN algorithms and compare with the WKNCN.

5.1. Experimental Settings

In order to investigate the classification behavior of the WKNCN and other methods, we performed experiments on 15 real data sets from the UCI Machine Learning repository [9]. The summary of the data sets are given in Table 1. The experiments for all data sets have been executed on a cluster with 72 CPU cores and 144 Gb RAM.

We used $Matlab$ [11] numerical computing software to implement various algorithms. Our written code employs the features in *Matlab Parallel Computing Toolbox* so that a block of code, whenever possible, is executed in parallel on a cluster of workers. We do pre-processing on a single data set: *Pima Indians Diabetes*, as this data set has missing values for a single attribute. As a pre-processing step, we replace the missing attribute value with the average of the existing values for that attribute.

We compare the performance of various methods in terms of classification error. For each data set, we use the first 80% of the data as training set, while the remaining 20% data as test set. We run the experiments 10 times to obtain reliable estimate for a data set and the data is reshuffled before each round to get different training and test set. Knuth shuffle [12] algorithm is used to shuffle the data before it is used in a trial. For all the compared methods, we use Euclidean distance as the distance measure.

For all data sets, each experiment is done with cross-validation. We use leave-one-out cross-validation for data sets with less than 750 instances, while for the others we use 10-fold cross-validation. For each experiment, the value of the neighborhood size k is varied from 1 to 15. The value of k for which a method obtains lowest validation error, is later tested on the test set to get the generalization error of the method. The final performance is achieved by averaging classification errors over 10 runs. To compare two methods, we determine the statistical significance of the differences in classification errors using t-test requiring $p < 0.05$.

5.2. Comparison of Various Kernels

Table 2 presents pairwise comparison of the kernels for WKNCN, while Table 3 shows the detail experimental results achieved for each kernel. In Table 2 and all the subsequent tables, *Inverse* is the abbreviated name for the Inverse-distance kernel.

The mean classification error computed from 10 runs along with standard deviations are presented in Table 3. As it can be seen in this table, when averaged over all data sets, the performances of WKNCN with Inverse-distance and Uniform kernel are comparable. For Inverse-distance kernel the average classification accuracy is 17.28%, while for Uniform kernel the average classification accuracy is 17.43%. Among all the tested kernels, WKNCN achieves worst classification error when the Heat kernel is used, yielding 19.06% average classification error.

In Table 2, the entry in the i^{th} row and j^{th} column gives the number of data sets on which the difference between kernel i and kernel j is statistically significant and also the performance of kernel i is better than kernel j. The last column denoted by $\sum_{col}$ provides the number of data sets for which the corresponding kernel performs better than one or more of the other kernels. A larger value in the last column corresponds to a better system.

Table 2. Pairwise comparison of various kernels

Kernel	Inverse	Heat	Uniform	$\sum_{col}$
Inverse	-	4	0	4
Heat	0	-	0	0
Uniform	0	4	-	4

i^{th} row and j^{th} column give the number of data sets on which method i performs statistically significantly better compared with method j. $\sum_{col}$ denotes the number of data sets for which the corresponding scheme performs better than other schemes

From Table 2, it is clear that WKNCN performs worse when the Heat kernel, a distance based kernel, is used to assign weight to the centroid neighbors. It can be argued that Heat kernel is the weakest among all the kernels as it performs worse than any other kernel on several data sets. This contradicts the finding in [4], where it was shown using 6 data sets that both Heat and Uniform kernels have same performances. Our experimental results are based on 15 data sets where we have found that, for four data sets, Heat kernel performs worse than any other kernel. Both Inverse-distance and Uniform kernel perform better than Heat kernel, but none of them surpass one another on any of the data sets. It is evident that both kernel has comparable performances, even though Inverse-distance is a distance based kernel, while Uniform is a rank based kernel.

It is worth noting that both Heat and Inverse-distance kernels are distance based kernels but the latter outperforms the former. Our careful observation of the Heat kernel reveals that, for two centroid neighbors, if there is a small difference between distances from the test pattern, it assigns unusually higher value to one neighbor than to the other. Thus, Heat kernel may overemphasize the importance of a neighbor which might degrade classification accuracy. Inverse-distance kernel does not have this problem because of having better weighting scheme.

5.3. Comparison of WKNCN and Other Methods

In Table 4, we show the classification error for WKNCN, KNCN and KNN, while in Table 5, we present a pairwise comparison of those methods. When we compare WKNCN with other methods, we only use the results of Inverse-distance kernel. It is evident from our analysis in previous section that Inverse-distance kernel is one of the best kernels for WKNCN. As it can be seen in Table 4, when averaged over all data sets, the best average performance is provided by KNCN, while the worst result is achieved by KNN.

It is evident from the last column of Table 5 that KNCN achieves best results among all the compared neighborhood based algorithms. KNCN not only performs better than

Table 3. Comparison of kernels for WKNCN. A number shown before $\pm$ is the mean classification error over 10 trials, while the number after $\pm$ is the corresponding standard deviation. *Inverse* is the abbreviated name for the Inverse-distance kernel. '*' indicates that the difference between the two specified methods as indicated by a column is statistically significant. '-' indicates that the difference is not statistically significant

Data Set	WKNCN Kernels			Statistical Significance		
	Inverse	Heat	Uniform	Inverse Vs. Heat	Inverse Vs. Uniform	Heat Vs. Uniform
Wine	25.88± 4.55	28.52± 6.94	25.00 ± 5.76	-	-	-
Sonar	16.56± 5.09	15.85± 5.54	15.61 ± 5.77	-	-	-
Glass	26.25± 3.77	26.25± 4.28	25.25 ± 4.63	-	-	-
Spect	18.49± 2.32	20.75 ± 3.3	20.38 ± 3.75	-	-	-
Spectf	23.02± 5.68	24.75± 5.5	24.34 ± 5.06	-	-	-
Liver	34.18± 4.99	40.60± 6.13	34.93 ± 5.32	*	-	*
Ionosphere	9.57± 1.79	12.43± 3.02	9.71 ± 2.59	*	-	*
Libras	13.83± 3.69	13.5± 2.88	13.16± 2.88	-	-	-
Wdbc	6.73± 1.97	8.41± 2.14	7.08± 2.00	-	-	-
Pima	27.39± 3.52	29.80 ± 3.35	27.12± 4.57	-	-	-
Vehicle	30.6± 4.08	35.45± 4.35	31.01 ± 4.58	*	-	*
Vowel	0.86± 0.48	0.86± 0.46	0.86 ± 0.48	-	-	-
Cardio	21.55± 1.30	24.14± 1.21	22.48 ± 1.2	*	-	*
Image	3.48± 0.93	3.63± 0.79	3.50 ± 0.82	-	-	-
Optdigits	0.84± 0.04	1.05± 0.27	1.034 ± 0.30	-	-	-
Average	17.28	19.06	17.43			

Table 4. Comparison of WKNCN and non-weighted KNN based methods. A number shown before $\pm$ is the mean classification error over 10 trials, while the number after $\pm$ is the corresponding standard deviation. '*' indicates that the difference between the two specified methods as indicated by a column is statistically significant. '-' indicates that the difference is not statistically significant

Data Set	Methods			Statistical Significance		
	Inverse	KNN	KNCN	Inverse Vs. KNN	Inverse Vs. KNCN	KNN Vs. KNCN
Wine	25.88± 4.55	28.53 ± 8.32	24.71 ± 9.01	-	-	-
Sonar	16.56± 5.09	17.80± 4.01	18.04 ± 5.29	-	-	-
Glass	26.25± 3.77	25.5± 4.53	26.75± 4.57	-	-	-
Spect	18.49± 2.32	23.21 ± 5.27	17.74 ± 3.23	*	-	*
Spectf	23.02± 5.68	23.21 ± 5.26	21.89 ± 4.28	-	-	-
Liver	34.18± 4.99	34.03± 3.84	29.70 ± 4.53	-	*	*
Ionosphere	9.57± 1.79	15.86± 3.89	6.86 ± 2.92	*	*	*
Libras	13.83± 3.69	13.66± 2.70	14.17± 3.62	-	-	-
Wdbc	6.73± 1.97	6.81± 2.32	5.75 ± 1.78	-	-	-
Pima	27.39± 3.52	27.32 ± 4.12	26.21 ± 4.72	-	-	-
Vehicle	30.6± 4.08	35.75± 4.06	26.77 ± 2.62	*	*	*
Vowel	0.86± 0.48	0.86± 0.48	0.86 ± 0.48	-	-	-
Cardio	21.55± 1.30	24.45± 1.39	23.67 ± 2.12	*	*	-
Image	3.48± 0.93	3.61 ± 0.68	3.79 ± 0.36	-	-	-
Optdigits	0.84± 0.04	1.35± 0.38	1.16 ± 0.51	*	-	-
Average	17.28	18.80	16.54			

its predecessor KNN, it also surpasses its weighted version. KNCN performs statistically significantly better than WKNCN on three data sets, whereas the latter surpasses the former only on a single data set. It can also be seen that KNN is the weakest method among all the methods compared. WKNCN achieves significantly better results than

Table 5. Pairwise comparison of WKNCN, KNN and KNCN

Method	WKNCN (Inverse)	KNN	KNCN	$\sum_{col}$
WKNCN (Inverse)	-	5	1	6
KNN	0	-	0	0
KNCN	3	4	-	7

i^{th} row and j^{th} column give the number of data sets on which method i performs statistically significantly better compared with method j. $\sum_{col}$ denotes the number of data sets for which the corresponding scheme performs better than other schemes

KNN on five data sets, while KNCN performs better than KNN on four data sets. Unlike [4] where weighted voting schemes were shown to improve KNCN performance, from our experimental results, we can conclude that assigning weights to the neighbors in KNCN do not improve its performance.

It is important to figure out the reasons for the differences in classification errors among various methods. The differences might be due to the distribution of the selected neighbors or due to the selection of suboptimal neighborhood size k. In our experiments, for each method, optimal k value for a data set is selected using cross-validation, before the test set error is computed. Thus, we can argue that if a method fails to achieve good result even after the selection of the appropriate k value, it may be due to the distribution of the selected k neighbors that fails to address a particular type of data distribution. The effect of the distribution of the selected neighbors on the performance of each method, especially on the WKNCN, needs to be investigated.

6. Conclusion

In this paper, we have performed extensive analysis on the KNN, KNCN and WKNCN methods. We have investigated the impact of weighted voting schemes on the KNCN as well as compared the various kernel functions for the WKNCN. Apart from this, we have investigated the Inverse-distance kernel, a new distance based kernel function for the WKNCN.

From our analysis, based on classification error rate of each method, it is evident that not all distance based kernels are weaker than rank based kernels. We observe that Heat kernel, the only existing distance based kernel has worse performance than Uniform kernel, the existing rank based kernel. This contradicts the results reported in the previous studies conducted with small number of data sets. However, well-designed distance based kernel like Inverse-distance kernel, can exhibit comparable performance as the Uniform kernel.

Our results and analysis also show that, even though WKNCN outperforms KNN algorithm, weighted voting schemes do not significantly improve the performance of KNCN, as reported in the previous studies. In future, we will perform more analysis to understand the reason why weighted voting schemes do not work well for KNCN. In addition to that, we will investigate the effect of the distribution of centroid neighbors in WKNCN using diverse data sets.

References

[1] Cover, T., Hart, P.: Nearest neighbor pattern classification. IEEE Transactions on Information Theory 13(1), 21– 27 (1967)

[2] Sánchez, J., Pla, F., Ferri, F.: On the use of neighbourhood-based non-parametric classifiers. Pattern Recognition Letters 18(11–13), 1179 – 1186 (1997)

[3] Chaudhuri, B.: A new definition of neighborhood of a point in multi-dimensional space. Pattern Recognition Letters 17(1), 11 – 17 (1996)

[4] Gou, J., Du, L., Xiong, T.: Weighted k-nearest centroid neighbor classification. Journal of Computational Information Systems 8(2), 851–860 (2012)

[5] Dudani, S.A.: The distance-weighted k-nearest-neighbor rule. IEEE Transactions on Systems, Man and Cybernetics SMC-6(4), 325 –327 (april 1976)

[6] Ahn, H., Kim, K.j., Han, I.: Global optimization of feature weights and the number of neighbors that combine in a case-based reasoning system. Expert Systems 23(5), 290–301 (2006)

[7] Altincay, H.: Improving the k-nearest neighbour rule: using geometrical neighbourhoods and manifold-based metrics. Expert Systems 28(4), 391–406 (2011)

[8] Denoeux, T.: A k-nearest neighbor classification rule based on dempster-shafer theory. IEEE Transactions on Systems, Man and Cybernetics 25(5), 804–813 (1995)

[9] Frank, A., Asuncion, A.: Uci machine learning repository (2012), `http://archive.ics.uci.edu/ml`

[10] Hechenbichler, K., Schliep, K.: Weighted k-nearest-neighbor techniques and ordinal classification. In: Discussion Paper 399, SFB 386 (2006)

[11] MATLAB: version 7.10.0 (R2010a). The MathWorks Inc., Massachusetts, USA (2010)

[12] Knuth, D.E.: The Art of Computer Programming, Vol. 2. Addison-Wesley, Reading, MA, second edn. (1981)

Mining Software Repository to Identify Crosscutting Concerns Using Combined Techniques

Ingrid Marçal, Rogério Eduardo Garcia, Ronaldo C. M. Correia, Celso Olivete Junior
Departamento de Matemática e Computação – Faculdade de Ciências e Tecnologia
Universidade Estadual Paulista "Júlio de Mesquita Filho" – UNESP
in.marcal@gmail.com, {rogerio,ronaldo,olivete}@fct.unesp.br

Abstract

Modularization is a goal difficult to achieve in software development. Aspect mining aims to identify crosscutting concerns in non aspect oriented software allowing organizing them into aspects and, thus, improve the modularization. Several techniques have been proposed to identify crosscutting concerns from software repositories, usually by analyzing static data or execution traces. In this paper we present an approach on mining crosscutting concerns using combined results from two techniques: Frequent Closed Itemset Mining and Commit Frequency Analysis, which is presented in this paper as well. We evaluated our approach and observed that combining both techniques lead to better sets of crosscutting concerns candidates.

Keywords: code evolution, aspect mining, mining software repositories, software maintenance.

1 Introduction

Modularization is a goal difficult to achieve in software development. Some requirements, named crosscutting concerns (CC) [10], cannot be clearly mapped to isolated units of code, and their implementation tend to be scattered. Thus, modifying one piece of functionality might require changing scattered code leading to maintenance problems (e.g., entanglement, scattering and lack of legibility) [10]. Aspect Oriented Programming [10] offers mechanisms to mitigate problems caused by CC in software systems [13], allowing refactoring these CC into aspects. However, before the refactoring process, it is necessary to identify existing CC in software artifacts (aspect mining).

Software analysis can be *static*, *dynamic* and *historical* [4]. In this paper we focus on historical analysis of data extracted from software repositories, which allows us to investigate how an artifact was modified over time, when it was changed, why it was changed, who changed it, and what change had another as a side effect (changed together). Canfora et al. [4] pointed out that mining software repositories opens challenging research directions. Breu and Zimmermann [2] argue that CC might emerge along the software development, which means that some of them are no longer identified by static or dynamic analysis. Mulder and Zaidman [12] focused on mining items that are frequently changed together using a technique named *frequent closed itemset mining* [17], but they pointed out that it is necessary to improve the results, i.e., by combining with other.

Our main interest is to obtain results from different mining aspect techniques using the historical perspective of software development and investigate whether their combination improves their individual results. For this work, each source file (file name) in software repository is considered an item and each revision (version committed) a transaction. Our approach is based on how frequently source files are committed (*Commit Frequency Analysis*) and how frequently they are committed together (*Frequent Closed Itemset Mining*). We suggest filters to refine the results and evaluate the proposed approach using the *JHotDraw* project repository, which allow comparing results with other approaches in literature. Finally, we combine results from both mentioned techniques and observed an improvement on their individual results.

To present our approach, this paper is organized as follow: Section 2 presents a review of related works and the background required to analyze results; Section 3 presents the CF Analysis; Section 4 shows the used methodology; we present a discussion about our results in Section 5; at last, Section 6 presents final remarks about strengths and weaknesses of our approach, and future works.

2 Background and Related Work

Object-Oriented Programming (*OOP*) is a paradigm that aids Software Engineering, since the underlying object model provides a better fit to real domain problems, but

OOP techniques are not sufficient to clearly capture important design decisions about different concerns implemented in a software system [10, 13]. Aspect-Oriented Program (*AOP*) tackles this so-called *crosscutting concerns* (CC) problem supporting the definition of implementation units (*aspects*) that cut across the system units, composing the expected behavior [10].

Aspect mining aims at creating and studying new methods and tools to identify CCs present in non aspect oriented software, thus they can be reorganized into aspects. Breu et al. [2] research opened a new perspective for aspect mining exploring software repositories to search for CCs. When applied to software repositories, aspect mining can be done basically in three steps: data acquisition, processing and presentation [12]. Data acquisition and presentation aim at collecting data from software repositories and presenting results, respectively. The core consists in applying a mining algorithm to process acquired data in order to extract and derive information. There are techniques and algorithms available in literature to process data from software repositories [2, 5, 14, 15]. Software repositories can be used for aspect mining purpose using different approaches, and in this paper we explore *Frequent Closed Itemset Mining* [15].

Mulder and Zaidman [12] applied *Frequent Closed Itemset Mining* in software repositories. *Frequent Closed Itemset Mining* is a constrained *Frequent Itemset Mining*, which is usually applied to discover association rules [3]. *Frequent Itemset Mining* describes the frequency of simultaneous appearance of related elements, using the following definitions summarized from [12].

2.1 Definitions for Frequent Itemset Mining

Let $I = \{I_1, I_2, ..., I_m\}$ a set of items, $X \subseteq I$ an *itemset*, D a data base with a set $D = \{t_1, t_2, ..., t_n\}$ of transactions, where $t_i = \{I_{i1}, I_{i2}, ..., I_{ik}\} \mid I_{ij} \in I$. Let also, $t(X)$ a set of transactions that contains an *itemset* X, so $t(X) = \{Y \in D \mid Y \supseteq X\}$.

The $support$ of an *itemset* X is defined as the ratio between the number of transactions on the data base that contains X and the total number of transactions on the data base, so

$$support(X) = \frac{\mid t(X) \mid}{\mid D \mid}$$

An *itemset* X is $frequent$ if it's *support* is greater than or equals to a given minimum *support*: $support(X) \geq minsupport$. The set of all frequent *itemsets* is denoted by FI and is a subset of all possible *itemsets* generated by I: **FI** $\subseteq 2^I$ [1][8].

We are not interested in subsets with the same *support* of their superset – they are considered irrelevant due to redundant information. In order to ignore irrelevant *itemsets* we have to look for frequent closed *itemsets*. A frequent *itemset* is considered *closed* if there is no supersets with the same *support* [15, 12]. Formally, an *itemset* X is closed if: $I(t(X)) = X$, where $I(S) = \bigcap_{T \in S}$ and T,$S \subseteq D$ [15].

Frequent closed itemsets integrates a larger set of frequent items. Let FCI the set of all frequent closed *itemsets*, so: $FCI \subseteq FI$

The most common metrics to measure results in data extraction algorithms are *precision* and *recall* (and we used the same metrics to compare results [12]). *Precision* (P) is defined as the ratio of items predicted correctly and the total number of items predicted. *Recall* is defined as the ratio of items predicted correctly and the total number of items that can be selected. The *F1-measure* is a combination of *Precision* and *Recall*, defined as:

$$F1 = \frac{2(P \times R)}{P + R}$$

$F1$ assumes values ranging from zero to one, where 0 is the worst case. For aspect mining purposes, on *Precision* metric *items predicted correctly* refers to the number of files confirmed as a CC, and *the total number of items predicted* is the number of items that composes the final result. Finally, on *Recall* metric the *total number of items that can be selected* refers to total number of known CCs.

Mulder [12] applied *Frequent Closed Itemset Mining* in two granularity levels: file-level and method-level. For file-level only file names were considered and for method level the author performed syntactic analysis on each file, to identify added and modified methods. It was observed that applying *Frequent Closed Itemset Mining* results in low precision, indicated by low F1-value on final results. The F1-value is provided by the F1-measure.

3 Commit Frequency Analysis

We have made some assumptions to propose the *Commit Frequency Analysis*. Kiczales et al. [10] have stated that: "In general, whenever two properties being programmed must compose differently and yet be coordinated, we say that they cross-cut each other. Because GP[1] languages provide only one composition mechanism, the programmer must do the co-composition manually, leading to complexity and tangling in the code". Such "co-composition manually" is done using the mechanisms offered by languages, that is, methods and their callings or spread code across multiple source files.

Zimmermann et al. [17] pointed out that "every change in a method implies a change in the enclosing class", what supports the rationale for searching changes in the enclosing files or packages to identify patterns like 'this change

[1]Generalized-Procedure languages for the purposes of the discussion, which is simpler to focus on what is common across several programming languages, including Object Oriented ones.

implies another change in this package'. Instead of method level, such search at file level might be less precise in locating modifications, but provide higher confidence [17]. Also, they pointed out that "the revision history can also tell us which parts of the system are coupled by common changes or co-changes". Other evolutionary couplings and patterns are presented and discussed by Kagdi et al. [9]. Additionally, Van Rysselberghe and Demeyer [16] have introduced the concept of *Frequently Applied Changes* (FACs), defined as changes occurring multiple times in the system version history. They analyzed source code changes and suggested that FACs may be used to identify recurring change patterns and, in turn, identification of refactorings.

Based on that, we propose the *Commit Frequency Analysis* technique and its metric to analyze potential CCs. We proposed a top-down approach focusing on files (classes). The rationale considers each commit as a group of files with some coupling degree, and coupled files (classes) might be considered as CC candidates for further analysis. So, we do not focus on creating groups of related files (classes), we already have grouped files and focus on eliminated files (classes) that can be discard as candidates to aspect. As metric, *Commit Frequency* represents the number of commits across multiples versions.

The approach proposed consists in a sequence of steps: 1) Compute the *Commit Frequency* (CF) for each file; 2) Discard files with CF equals to one, that are isolated commits (only groups of files must be considered as potential candidates); 3) Perform statistical analysis to obtain the shape of distribution and descriptive data; 4) Define an interval of interest for analyzing by establishing minimal and maximal thresholds; 5) Apply those thresholds and obtain a list of files with CC candidates; 6) Obtain the list of versions where each file was updated; 7) Each file versions list is manually analyzed to verify whether there is a concern that matches; 8) Compute the F1-value for each group and, then, the average of F1-value to obtain an overall score. We have used a heuristic to define the interval of interest that is presented in the following, along with the application and the results obtained.

4 Methodology

The framework proposed by Marin et al. [11] was used to state our approach on applying individually both techniques and combining them. Thus, our approach fits in the following: **Search goal**: CCs represented by files that are frequently and simultaneously modified – we do not restrict our investigation to unique sorts of concerns (e.g, *Consistent Behavior* and *Contract Enforcement* [11]) because our intent is to investigate all possible resultant sort of concerns; **Results Format**: A list with file names that are frequently and simultaneously modified – this behavior is a characteristic of identifying CCs candidates; **Results Interpretation**: The resultant file names are entities names containing methods that are CC candidates; **Validation Metrics**: we manually compare our results to known sets of CCs for the subject analyzed, then, the *F1-measure* is used to precise how the results match with these known sets of CCs.

The approach proposed is depicted in Figure 1. Each step is supported by modules implemented in our own mining tool. We used SVNKit library to develop a module for acquiring data from software repository (**Data Acquisition**). Hence, we collected all entries from the history of development database that consists in: revision identification number, date, log message and the names of entities modified, added or excluded. Data collected from software repository are stored using Apache Commons Collection (hash table) using a format accepted by the algorithms developed (**Pre-processing**). As **Processing**, the LCM algorithm [15] was chosen to apply *Frequent Closed Itemset Mining* due its high performance with large amount of data; also, we created an algorithm to perform the *Commit Frequency Analysis*. Regarding the **Filtering**, a filter to *Data Acquisition* was defined, and the data set obtained has been used on both techniques; also, other filters were used on each technique, and their parameters are discussed in the following. After the experiments with *Frequent Closed Itemset Mining* and *Commit Frequency Analysis*, we combine the results from both (**Combining**). Thus, it is possible to analyze whether combining the results from both techniques improves their individual results. Finally, the results are presented to the user in a list of entities names representing the CCs candidates identified (**Presentation**).

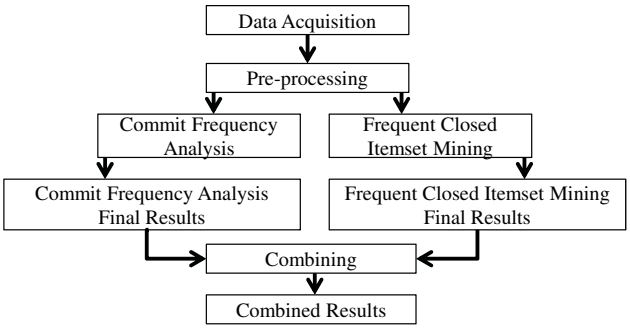


Figure 1: Schema proposed to combine results

5 Evaluating the approach

To validate the proposed approach and discuss our results, it is necessary to compare the obtained CCs candidates with known sets of CCs (oracle). However, there are few known crosscutting concerns documented and publicly available, which makes difficult to compare results. Marin

et al. [11] proposed a web forum[2] where aspect mining researchers can exchange and discuss aspect candidates found in (open source) software systems. Despite their effort, only their own results are available there, obtained mainly from fan-in analysis. Therefore, our validation and discussion are based on information found in aforementioned forum and on a list of known CCs from other works [2, 5, 14]. Also, Frank Mulder kindly provided us with his results (list of CCs). The *JHotDraw* – a java framework to support graphical applications development – has been used in aspect mining literature. We chosen it to enable the validation of results by comparing to aspect mining results from previous works. All concerns identified in literature applying *Dynamic Analysis*, *Identifier Analysis*, *Fan-in analysis* and *Frequent Closed Itemset Mining*were compiled and are presented in Table 1.

Table 1: Known CC compiled from literature [5, 11, 12, 14]

Add text	Add URL to figure
Bring to front	Command executability
Connect figures	Connect text
Manage figure changed event	Figure update
Get attribute	Move figure
Manage handles	Manage figures outside drawing
Set Attribute	Send to back
Undo	Manage view rectangle
Visitor	Adapter
Consistent Behavior	Composite
Decorator	Exception Handling
Command	Observer
Persistence	Activation
Command Execution	Producer-Consumer
Event Handling	File Handling
Handling Mouse Events	Area Tracking
Background Drawing	Managing Display Boxes
Chopping Figures	Clearing Figures
Finding Figures	Color Choosing
Constraining Points	Figure Inclusion
Menu Handling	Managing Views
Font Handling	Image Handling
Create Drawings	Desktop Management
Invocation	Manipulating Figure Handles
Finding Connection	Dealing With Selections
Inserting Figures	Layout Calculation
Moving Figures	Handle and Figure Enumeration
Resource Management	Performing Actions
Iterating Over Collections	Working With Maps

5.1 Experiments and Results Discussion using Frequent Closed Itemset Mining

The first step (*Data Acquisition*) was executed to extract data from repository. For that, we used a filter applied to establish the minimum interval between commits by author – we considered that developers usually commit related changes in different revisions in short period and, thus, in this work we decided consider such revisions as a single commit to reduce noise on final results. After analyzing meta-data about the commits, we empirically decided to use a threshold of 1700 seconds between commits performed by the same author. So, we obtained a data set of 15747 files across 660 versions committed.

We have run several experiments applying *Frequent Closed Itemset Mining* to acquired data, using two parameters to compare their results. The first parameter is the minimum changeset size: very large and very small changesets can be ignored, since they rarely indicate CC [12]. And *Support*, the second parameter, is a measure that indicates the frequency of an *itemset*: high frequencies means possible presence of CCs. In this paper we related the frequency of an *itemset* with its size.

As initial experiment, *Frequent Closed Itemset Mining* was applied to *JHotDraw* repository considering only java source code – file names with *.java* extension, using $Support = 2.0$. A set of 493 changesets was obtained, and its *F1-value* is 0.35. Next, the filters were applied to investigate their influences.

A set of experiments focused on investigating the relation between the precision and the changeset size. Thus, experiments were executed using changeset size as a constraint, and we established different changeset size empirically, keeping $Support = 2.0$. Initially, we used changeset size limited to 5, increasing this value at intervals of 5 for next 20 experiments running. We repeated a sequence of experiments using the initial changeset size limited to 10, and then increasing at intervals of 10. For experiments with changeset size less than 40, we observed low F1-values reaching the minimum value of 0.2. For experiments using changeset size equal to 40, the F1-value obtained is 0.449. Thus, we observed that the changeset size influences the results precision only for size values at the interval $[2, 40]$. Furthermore, experiments showed that few *itemsets* have high cardinality – usually *itemsets* with low cardinality are frequent – that is an important lesson learned used on evaluating *Commit Frequency Analysis*. It is important to note that the F1-value is not the same pointed out by Mulder and Zaidman, since they used JHotDraw version 5.4b1, and we considered all entries from JHotDraw repository.

Other experiments were conducted using different *Support* values ($Support = 3.0$ and $Support = 4.0$), but the results obtained were worse – even empty itemsets were obtained. Our best result (F1-value = 0.449) was better comparing to no filter result. However, we considered that it can be improved, especially taking into account that the F1-measure ranges at interval [0,1]. In order to explore alternatives to improve aspect mining precision, we investigate the application of *Commit Frequency Analysis* using software repository.

[2] http://swerl.tudelft.nl/amr/

5.2 Experiments and Results Discussion using COMMIT FREQUENCY

To apply Commit Frequency Analysis we used the same data set previously obtained in *Data Acquisition* – 660 versions with 15747 files committed. The file list consists in 3358 files with no repetition across 660 versions. The *Commit Frequency* was computed for each file (step 1). We applied the filter established as *step 2*, selecting only files with CF greater than 1. At this point, 2236 files were selected across 212 versions.

The statistical analysis was performed according [7] (step 3). The measurements of location and variability are presented in Table 2. Also, we analyzed the frequency distribution and the normal probability curve[3], and we observe a concentration of commits at low CF values. At first glance, we considered the minimal and maximal thresholds around the mean value – *mean* $\pm$ *standard deviation*. However, considering the distribution of normal probability plot (omitted due page restriction), we decided to be more restrictive, focus on interval $2 \leq CF < 10$ (step 4). It is important to note that there is no rule defined to choose the intervals, but statistical data description was used as parameter to make a decision. The heuristic is a greedy approach, trying to select more potential candidates, but we are aware that some CCs candidates can be out of the range, specially considering the Kurtosis value [6].

Table 2: Location and variability Measurements

Measure	Value
Mean	6,540698
Median	5
Standard Deviation	5,825551
Variance	33,93705
Kurtosis	17,3149

The filtering on *step 5* (considering interval $2 \leq x < 10$ to CF-values) resulted in 1838 files. So, we selected reviews where such files appear, and such reviews represent changesets with potential CCs candidates (step 6). We did not use any changeset filter. Each changeset was manually analyzed: we compared items implementation (file) to the concerns listed in Table 1 to verify whether the group items (files) can be refactored into aspect, or not (step 7). We considered the best matches to obtain the F1-value. To get an overall score for the groups, we take the average of these F1-values (step 8). We get F1-value = 0.448.

One may observe that the F1-values obtained using both techniques were quite similar, what lead us to evaluate not only the measures, but also the data set obtained. *Frequent Closed Itemset* has generated data sets with less elements and, consequently, the F1-value reach such result by avoiding false positives, which increase the measures of precision and recall, but several CCs are ignored as well. The *Commit Frequency Analysis*, on the other hand, has generated data sets with greater number of elements, but reach similar F1-value because it increases the number of false positive as well. The increasing of false positive requires (and wastes) time and effort on unnecessary analysis.

[3]Omitted due to page restriction

5.3 Combining Results

According to schema depicted in Figure 1, we aimed at improving individual results dealing with the limitations observed in *Commit Frequency Analysis*. We proposed to combine the results by taking the intersection of their individual results. For that, we used results from *Frequent Closed Itemset Mining* to compare to those files excluded from *Commit Frequency Analysis* due to filtering. We followed the steps: 1) Take the list of files excluded from previous *Commit Frequency Analysis*; 2) Compare the itemsets obtained from *Frequent Closed Itemset Mining*, creating a set of intersections S; 3) For $s \in S$, obtain the version which updated s; 4) Each file in version list must be manually analyzed to verify whether there is a concern that matches; 5) Compute the F-1-value for each group and, then, the average of F1-value to obtain an overall score.

As first step, we filtered CF-values ≥ 10 getting 398 files. We compared those files to itemsets from *Frequent Closed Itemset Mining* – 44 files matched, but 33 was previously considered related to files that have CF-value less than 10 – grouped in some version and, therefore, considered in a changeset (step 2). So, 11 different files with CF-value in $10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 31$ should be submitted to a new *Commit Frequency Analysis*, what helps on dealing with the effect of filtering values in a distribution with high Kurtosis value. For that, a list of versions where those files were updated was created (step 3).

Each file in version list was manually analyzed, trying to use the best matching (step 4), resulting in 65 different CCs. The F1-value was computed (step 5) and overall F1-value was 0.590. This result seems better, but it was reached due to low cardinality on changeset (version) analyzed: the first *Commit Frequency Analysis* had average of changeset cardinality equal to 59.57, and for this analysis 9.47. Even with smaller number of CCs identified, the reduction of false positives has influenced to increase F1-value.

6 Conclusion

In this paper we present an approach for mining crosscutting concerns using results from two techniques: *Frequent Closed Itemset Mining* and *Commit Frequency Analysis*, which is presented in this paper as well. Both tech-

niques deal with the same granularity level (files). *Commit Frequency Analysis* is described step by step, its applications was presented, and we showed how to combine the results. In order to evaluate our approach, we chose a known open source project used on several researches. The results found in literature [2, 14, 5] were used as parameter to evaluate which crosscutting concern candidate would be classified as aspect. Also, this decision was based on practical judgments rather than concepts and principles, as found in literature, as well. We considered the best matches to not introduce any bias. It was observed that combining results leads to better sets of crosscutting concerns candidates. The greater F1-values and the increased number of crosscutting concerns identified are evidences of improvement.

Important observations were made about aspect mining using software repository and the chosen techniques for this paper, they are: a) the F1-values from both techniques are quite similar, but obtained from sets with different characteristics; b) the rate of false positives obtained along *Frequent Closed Itemset Mining* experiments is lower than ones obtained from *Commit Frequency Analysis*; c) we identified more crosscutting concerns applying *Commit Frequency Analysis* than applying *Frequent Closed Itemset Mining*; d) applying filters on both techniques improves their final result significantly; e) results from *Commit Frequency Analysis* are improved by combining *Frequent Closed Itemset Mining* results.

Our approach was validated with only one project software repository, which is a known benchmark on aspect mining research. However, we intent to conduct other experiments using other software projects repositories, for example, Eclipse[4] and Tomcat[5]. Also, we intent to investigate how other techniques that deal with mining repositories at Method-Level Granularity can be used in a complementary way to our work at file-level analysis.

Acknowledgment

We would like to thank Frank Mulder for sending us a CSV file, sharing with us his results.

References

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *SIGMOD '93: Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, pages 207–216, New York, NY, USA, 1993. ACM Press.

[2] S. Breu and T. Zimmermann. Mining aspects from version history. In *Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering*, pages 221–230, Washington, DC, USA, Nov. 2006. IEEE Computer Society.

[3] D. Burdick, M. Calimlim, and J. Gehrke. Mafia: A maximal frequent itemset algorithm for transactional databases. In D. Georgakopoulos and A. Buchmann, editors, *Proceedings. 17th International Conference on Data Engineering*, pages 443–452. IEEE Computer Society, 2001.

[4] G. Canfora, M. Di Penta, and L. Cerulo. Achievements and challenges in software reverse engineering. *Commun. ACM*, 54(4):142–151, Apr. 2011.

[5] M. Ceccato, M. Marin, K. Mens, L. Moonen, P. Tonella, and T. Tourwe. A qualitative comparison of three aspect mining techniques. In *Proceedings of the 13th International Workshop on Program Comprehension*, IWPC '05, pages 13–22, Washington, DC, USA, 2005. IEEE Computer Society.

[6] L. T. DeCarlo. On the meaning and use of kurtosis. *Psychological Methods*, 2(3):292–307, 1997.

[7] J. L. Devore. *Probability & Statistics For Engineering And The Sciences*. Cenage Learning, Boston, MA, 2012.

[8] M. H. Dunham. *Data Mining: Introductory and Advanced Topics*. Prentice Hall, 2002.

[9] 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, March 2007.

[10] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. marc Loingtier, and J. Irwin. Aspect-oriented programming. In *Proceedings of the European Conference on Object-Oriented Programming*, pages 220–242. SpringerVerlag, 1997.

[11] M. Marin, L. Moonen, and A. van Deursen. A common framework for aspect mining based on crosscutting concern sorts. In *Proceedings of the 13th Working Conference on Reverse Engineering*, WCRE'06, pages 29–38, Washington, DC, USA, 2006. IEEE Computer Society.

[12] F. Mulder and A. Zaidman. Identifying cross-cutting concerns using software repository mining. In *Proc. of the Joint ERCIM Workshop on Software Evolution and Int. Workshop on Principles of Software Evolution*, IWPSE-EVOL '10, pages 23–32, New York, NY, USA, 2010. ACM.

[13] E. F. Robert, E. Tzilla, C. Siobhan, and A. Mehmet. *Aspect-Oriented Software Development*. John Wait, 2005.

[14] P. Tonella and M. Ceccato. Aspect mining through the formal concept analysis of execution traces. In *Proceedings of the 11th Working Conference on Reverse Engineering*, WCRE'04, pages 112–121, Washington, DC, USA, 2004. IEEE Computer Society.

[15] T. Uno, T. Asai, Y. Uchida, and H. Arimura. Lcm: An efficient algorithm for enumerating frequent closed item sets. In *In Proceedings of Workshop on Frequent itemset Mining Implementations*, 2003.

[16] F. Van Rysselberghe and S. Demeyer. Mining version control systems for facs (frequently applied changes). In *International Workshop on Mining Software Repositories*, pages 48 – 52, Edinburgh, Scotland, UK, 2004. IEE.

[17] T. Zimmermann, P. Weissgerber, S. Diehl, and A. Zeller. Mining version histories to guide software changes. *IEEE Trans. on Software Engineering*, 31(6):429–445, june 2005.

[4] www.eclipse.org

[5] http://tomcat.apache.org

The Layered Architecture revisited: Is it an Optimization Problem?

Alvine Boaye Belle[1], Ghizlane El Boussaidi[1],Christian Desrosiers[1], Hafedh Mili[2]
[1]Department of Software and IT engineering, École de technologie supérieure
[2] Department of Computer Science, Université du Québec à Montréal
Montréal, Canada

***Abstract*—In this paper we present an approach to reconstruct the layered architecture of software systems. We revisit the layered architectural style to extract a minimum set of fundamental principles. These principles are used to specify a set of constraints that a layered system must conform to. Recovering the layered architecture of a system is then translated to an optimization problem that we solve using a heuristic search algorithm. Preliminary experimentations with the approach yielded interesting results.**

Keywords-reverse-engineering; architecture reconstruction; layered architecture; layered principles; optimization

I. INTRODUCTION

The evolution process of a software system can span from maintenance, to modernization, to an entire replacement of the system [1]. Various other disciplines are considered as part of this process including assessment, restructuring and refactoring of the system's architecture. There are many definitions of software architecture in the literature (e.g., [3, 7, 8, 11]). However, there is a consensus on the following: 1) architecture represents a judicious partitioning of the system into parts with specific relations among these parts [7]; and 2) architecture aims at satisfying a set of functional requirements and quality attributes [3, 8, 11]. Software architecture is commonly defined as a set of components and connectors (i.e., interactions between components). When designing software architectures, an architect relies on a set of idiomatic patterns, commonly named architectural styles or patterns, which describe families of systems [3]. An architectural style determines the vocabulary of components and connectors that can be used in instances of that style, together with a set of constraints on how they can be combined. Many common architectural styles are described in [3, 7, 11]. Examples of such styles include layered, pipes and filters, client-server and service-oriented styles; each of these styles has its own vocabulary and constraints and promotes some specific quality attributes.

Software systems are practically built by combining and composing architectural styles. However many researchers observed that the as-built architecture does not conform to the initial style that guided its design (e.g., [4, 5]). This is mainly due to: 1) the conceptual gap between the abstract elements that define a style and the concrete source code constructs that implements the system [5]; 2) violations of the style constraints due to their misinterpretation; and 3) the continuous and cumulative changes undergone by the system, which increases its complexity and leads to a deviation from its initial design [1]. Furthermore, the as-built architecture is often insufficiently and inaccurately documented [4]. Hence to appropriately support the evolution and maintenance of an existing software system, a software architecture reconstruction process is required to reconstruct and document its architecture. The reconstructed architecture enables to understand the system, to restructure it as needed, and to constrain its future evolution.

Many approaches were proposed to support architecture recovery using various techniques and producing different tools that support them [4]. The technique used is generally dependent on the way the system's data is represented. In [14], techniques were classified into three automation levels: quasi-manual, semi-automatic and quasi-automatic. In the context of large and complex systems, we need a quasi-automatic technique that alleviates the burden of reconstructing these systems architectures. One such technique is clustering, which is a common used technique to reconstruct architecture (e.g., [9, 10, 12, 13]). However, these approaches target specific languages and systems and do not use a standard representation of the data of the system under analysis. As a consequence, resulting tools do not interoperate with each other [2]. To tackle this problem, the OMG introduced the Knowledge Discovery Metamodel (KDM) [6]. The KDM defines a meta-model for representing—at various levels of abstraction—all aspects of existing legacy systems. This meta-model provides a common interchange format to ensure interoperability between tools. Although the KDM specifies concepts and relations to describe software architecture of existing systems, it does not specify or suggest a way of inferring these high-level representations from low-level data that were extracted from these systems. Furthermore, most of existing approaches to reconstruct architecture do not enable to build architectural views as the ones commonly used in a forward design process (e.g., a layered style [11]).

To address the issues mentioned above, we propose an approach that makes use of the KDM standard to reconstruct and document software architectural views of existing systems. In this paper, we focus on systems built according to the layered style which is a widely used pattern to structure large software systems. We analyzed the layered style to establish a minimal set of principles that can guide the recovery of a layered architectural view of software systems. These principles were used to define a set of metrics and constraints that a layered software system should satisfy. The problem of recovering layers of software systems is then translated into an optimization problem that we solve using a heuristic search algorithm. The main contributions of this paper are: 1) a minimal set of principles and metrics that guide the layering

recovery process; 2) the translation of the layering recovery problem to an optimization problem; and 3) a layering recovery approach that is language and platform independent.

The paper is organized as follows. We first discuss the layered style in section 2. Section 3 introduces the layering principles that we retained from our analysis of this style. In section 4, these principles are used to define metrics and constraints that enable to formalize the layering recovery process as an optimization problem. We describe our layering reconstruction algorithm in section 5. Section 6 presents and discusses the experimentation results. Related works are discussed in section 7 and we conclude and outline some future works in section 8.

II. BACKGROUND: THE LAYERED STYLE

The layered style is one of the most used styles to depict a structural view of a software system. It was described in many reference books and papers (e.g., [3, 11, 7, 19]). The layered style is a technique for decomposing a software system into groups of subtasks where each group of subtasks is at a particular level of abstraction [11]. In other words, a layered architecture is an organized hierarchy where each layer is providing services to the layer above it and serves as a client to the layer below [3]. The OSI reference model [17] is one of the most known layered architecture system. In OSI, a layer uses services provided by lower layers and adds value to them to provide services needed by higher layers.

The layered style promotes a set of quality attributes which include reuse, portability, maintainability, understandability, and exchangeability ([11], [3], [7]). Different strategies can be used to partition a software system into layers while guaranteeing these quality attributes. The most common layering strategies are the responsibility-based and the reuse-based layering strategies [19]. The responsibility-based strategy aims at grouping components of the system according to their responsibility and assigning each responsibility group to a layer. The reuse-based layering strategy aims at grouping components of the system according to their level of reuse and assigning the most reusable components to the bottom layers. The OSI model [17] and the e-learning systems [21] are respective examples of these two strategies.

The various layered style descriptions given in the books and papers we analyzed imply that, in an ideal layered architecture, a layer may only use services of the next lower layer. This is referred to as strict layering in [11] and as closed layering in [18]. This strict ordering relation is often violated in practice; i.e., very often, layered systems allow a layer to use services provided by any lower layer. Not restricting the dependence of a layer to its lower adjacent layer is considered as a regular feature in the open layering [18] and the relaxed layering [11]. However, it is considered as a violation named a skip-call violation in [20] and layer bridging in [7].

Exceptionally, a layer may need to rely on a service offered by an upper layer. These dependencies are called back-calls in [20] and are discussed in [7] under the name "upward usage". However, the quality attributes promoted by the layered style are no longer supported when layers are allowed to use services of higher layers without restriction [7]. Accordingly, the structure of a layered architecture must be a directed acyclic graph or at least a directed graph with very few cycles connecting different layers.

Another important characteristic of a layered architecture is its number of layers. A layered system must be structured into an appropriate number of layers; this number depends on the abstraction criterion [11] used to order the systems' responsibilities. Different criterions may be used depending on the system under analysis and the layering strategy. While the OSI reference model has 7 layers [17], most of web-based applications have three layers. In [22], it is recommended to define some 4 to 6 layers of subsystems.

III. EXTRACTING PRINCIPLES FOR DISCOVERING SOFTWARE LAYERS

Based on the analysis of several definitions and descriptions of the layered style, we identified two important dimensions of reasoning when it comes to applying this style: responsibility and abstraction. Indeed, applying the layered style means partitioning the system into a set of components (i.e., responsibilities) and appropriately assigning these components to a set of abstraction levels. We hence retain two principles that will be used to specify the metrics and constraints that will guide the reconstruction process of software architectural layers. These two principles are discussed below.

A. *The Responsibility Principle*

This principle states that each layer of the system must be assigned a given responsibility [19], so that the topmost layer corresponds to the overall function of the system as perceived by the final user and the responsibilities of the lower layers contribute to those of the higher layers [11]. The concept of responsibility is defined in [8] as "*the functionality, data, or information that a software element provides*". Thus, the logic of a software system is divided into several responsibilities. Each responsibility is implemented by a set of interacting components that need to be cohesive and specific to a given domain [7]. In this context, each component of the system should be designed to implement a specific service and must belong to a single layer. Therefore, each component of the system contributes to the realization of the principle of responsibility.

This principle is related to the notion of modularity that has already been subject to a lot of studies. Most of work on architecture recovery using clustering techniques focused on decomposing systems into components while minimizing the coupling between resulting components and maximizing the cohesion of each component (e.g., [16, 13, 10, 23]). In the context of this paper, we analyze object oriented systems and we rely on existing decomposition of these systems into packages. Hence, we focus on the abstraction principle discussed in the following subsection.

B. *The Abstraction Principle*

This principle states that the layers of a system must be ordered according to the abstraction criterion that rules the flow of communication between components of the system. The abstraction principle encompasses two properties:

1) The Layer abstraction uniformity property: this property has two facets. The first facet is related to the fact that each layer must have a precise meaning [19]. This characteristic is closely related to the responsibility principle. The second facet corresponds to the fact that components of the same layer must be at the same abstraction level. The level of abstraction of a component often refers to its conceptual distance from the "physical" components of the system [11], i.e. hardware, database, files and network. Components at the highest levels are domain specific [22], they generally contain the visible functionalities provided by the system [11]. Somehow, this property led to several layering algorithms based on a depth traversal of dependency graphs built from existing software systems (e.g., [20, 23]).

2) Incremental layer dependency property: this property is related to the "ideal layering property" that states that a component in a layer (*j*) must only rely on services of the layer below (*j-1*). As discussed above, this principle is the one that is mostly violated, either through back-call or skip-call dependencies between layers. Our analysis of the various descriptions of the layered style and several open source projects led us to conclude that the incremental dependency property should be stated in a way that allows—to some extent—the skip-call and back call violations. Hence, we formulate this property as "*components of layer j-1 are mainly geared towards offering services to components of layer j*". This means that in the event when there are some skip-call and back-call dependencies between layers, the number of these dependencies must be insignificant compared to the number of downward dependencies between adjacent layers.

IV. THE LAYERED STYLE AS A COMBINATORIAL OPTIMIZATION PROBLEM

In the following, we define a set of metrics and constraints based on the abstraction principle as discussed above. These metrics and constraints are then used to formalize the layering principles as an optimization problem. In the context of our paper, we focus on object oriented systems and we work at the package level; i.e., we rely on existing decomposition of object oriented systems into packages. Thus, we assume that these packages have been designed in concordance with the responsibility principle.

A. Extracting Metrics and Constraints from the Layered Principles

To ensure the layer abstraction uniformity property, the packages of the same layer should be at the same distance from the "physical" or lowest layer packages. However, the existence of back-call and skip-call dependencies introduces a discrepancy between the packages' distances, even when they belong to the same layer. Hence, compliance with the layer abstraction uniformity property derives largely from compliance with the incremental layer dependency property. To formalize the later property using constraints, we introduce a number of metrics related to the dependencies between layers.

We define the relative index of use of a layer *j* by a layer *i* as the number of dependencies directed from layer *i* to layer *j*. This index is obtained by summing the weights of the dependencies directed from each package of layer *i* to each package of layer *j*. The dependency between two packages derives from the dependencies between their respective classes and it includes class references, inheritance, method invocation and parameters. In what follows, this relative index is labeled as:

- IndexOfUse(i,j) when j = i-1. IndexOfUse(i,j) denotes the number of dependencies directed from layer *i* to its adjacent lower layer *j*.
- SkipUse(i,j) when j < i-1. SkipUse(i,j) is the number of skip-call dependencies directed from layer *i* to layer *j*.
- BackUse(i,j) when i < j. BackUse(i,j) is the number of back-call dependencies directed from layer *i* to layer *j*.
- IntraUse(i) (when i=j). IntraUse(i) is the number of the dependencies inside layer *i*.

Figure 1 illustrates the calculation of the layer dependency metrics for a system made of three layers where all dependencies have the same weight (i.e., a weight of 1).

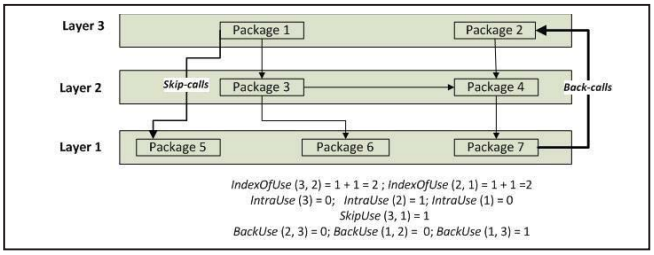


Figure 1. Example of the calculation of the 4 types of layer metrics.

In accordance with the incremental layer dependency property, we want to minimize the number of skip-call and back-call dependencies. This means that, apart from the upper layer adjacent to layer j, we must minimize the relative index of use relating other layers to layer j. However, we consider the skip-calls as a necessary evil (i.e., skip-calls are often used for performance reasons [11]) and tolerate them more than the back-calls which lead to a poorly structured system. These restrictions are formalized by the following constraint:

$$\text{For all } i, j, k \mid j<i \text{ and } k<j-1,\ \text{BackUse}(j,i) \le \text{SkipUse}(j,k) \le \text{IndexOfUse}(j, j-1) \quad (1)$$

Constraint (1) may be certainly satisfied when the number of the layers of the system is very small (i.e., when layers are merged). However, dependencies between packages of the same layer are not recommended unless otherwise stated [24] or when some concerns as portability need to be addressed [7]. Accordingly, we subjoined to constraint (1) the following constraint that limits the number of intra-dependencies of a layer:

$$\text{IntraUse}(j) \le \text{IndexOfUse}(j, j-1) \quad (2)$$

B. *Translating The Layering into an Optimization Problem*

The reconstruction process that we propose aim at rewarding the adjacency between layers while keeping their intra-connectivity quite low and minimizing the skip-calls and back-calls. Hence, we will use the metrics defined in the previous section to guide the process of assigning the packages of a given system to a set of layers. For this purpose, we define the individual layering cost (ILC) of a given layer *i* of the system as follows:

$$\mathrm{ILC(i)} = \alpha\mathrm{IndexOfUse(i, i-1)} + \beta\mathrm{IntraUse(i)} + \gamma\sum_{j=i-2}^{1}\mathrm{SkipfUse(i,j)} + \delta\sum_{j=i+1}^{N}\mathrm{BackUse(i,j)} \quad (3)$$

Where α is the cost of adjacent dependencies, β is the cost of intra-dependencies, γ is the cost of skip-call dependencies and δ is the cost of back-call dependencies. For instance, in figure 1, ILC(3) = 2α + γ.

In order to penalize the undesired dependencies and satisfy the two constraints defined before, the cost α must be lower than the other costs.The global layering cost LC of assigning the packages of a system to a set of *n* layers is then computed by summing the individual layering cost for each layer *i* of the system.

$$\mathrm{LC} = \sum_{i=1}^{n}\mathrm{ILC(i)}$$

The lower LC is, the better the assignment of packages to layers is. Attempting to reconstruct a layered architecture while minimizing its LC, is a problem that can be solved by adapting optimization algorithms that rely on heuristic search to reduce the search space.

V. RECONSTRUCTING THE LAYERED ARCHITECTURE USING A HEURISTIC SEARCH ALGORITHM

In order to build the optimal layering of software systems, we choose to adapt the hill-climbing technique [10] using our LC as a fitness function. We focused on the hill-climbing algorithm because it performs well in the context of large systems and it has been successfully used in several approaches. The algorithm works in an iterative way. It starts by an initial partition of the system's modules into a set of clusters; usually a randomly generated partition as in [10]. Modules are then moved between clusters to improve the partition according to some criterion. This criterion is based on maximizing or minimizing a fitness function.

A high-level view of our adaptation of hill-climbing to the layering problem is given in figure 2. It starts with an initial partition consisting of a set of n layers in which the uppermost one is constituted by the packages having no incident dependencies; the lowermost layer of this partition contains the packages having no outgoing dependencies; and the remaining packages are randomly assigned to other layers. The so-called initial system is then considered as the current solution of the algorithm (line 1). In the following iterations (lines 3 to 20), all the neighboring solutions are created (line 6) and evaluated using their cost (line 8). A neighbor solution is computed by moving a single package from a layer A to a layer B of the current solution, provided these two layers are different. The neighbor having the lowest value of LC is considered as the best neighbor of the iteration (lines 9 to 12) and accepted as the current solution if its cost is lower than the one of the current solution (lines 14 to 17). The algorithm stops if the current solution cannot be improved anymore (lines 18 and 19). In order to compute the cost of each neighbor, we set the values of α, β, γ and δ prior to the application of the algorithm.

```
LayeredSystem :: layeringBycost (LayeredSystem initialSystem , real MAX_VALUE)
1. currentSolution ← initialSystem
2. currentLC ← computeLC (initialSystem )
3. while (TRUE){
4.       bestLC  ← MAX_VALUE
5.       bestNeighbor  ← NULL
6.       neighborList ← computeAllNeighbors(currentSolution)
7.       for (neighbor in neighborList){
8.                neighborLC ← computeLC(neighbor)
9.                if (neighborLC  < bestLC) {
10.                       bestNeighbor ← neighbor
11.                       bestLC  ← neighborLC
12.               }//end if
13.      }//end for
14.      if (bestLC < currentLC){
15.               currentSolution ← bestNeighbor
16.               currentLC ←bestLC
17.       }//end if
18.      else
19.               return currentSolution
20. }//end while
21. return currentSolution
```

Figure 2. A high level view of our layering algorithm.

The algorithm generates the best layering solution which is generally made of 3 to 4 layers according to our experiments. So, in order to refine the resulting layering solution, the user can repeatedly re-execute the algorithm, selecting a given layer that should be further partitioned into layers. The algorithm will then be executed on a new partition constructed using only the packages of the selected layer.

VI. EXPERIMENTS WITH THE APPROACH

We implemented a tool supporting our approach within the Eclipse™ environment. This tool is made of two modules. The first module was built on top of the MoDisco open source tool which enables to analyze source code files of the system under study and to generate a KDM representation of the system. This KDM representation is then used by our module to extract the system's packages and their dependencies which are used to generate a module dependency graph. The latter allows the creation of the initial partition that is the input of the second module of our tool. The latter implements our layering algorithm.

We carried out preliminary experimentations of our layering algorithm on four open source projects. Some characteristics of these projects are shown in Table I. Table II summarizes the results of executing the algorithm on these systems using three different setups. We indicate for each setup the values of the cost parameters that were used and the number of layers (3 in this case). Notice that the adjacency cost (α) is set to 0 for all these setups; we reward downward adjacent dependencies. Table II displays for each setup the global layering cost (LC) for the solution returned by the algorithm. It also shows, for each returned solution, four values (Adjac, Skip, Intra, and Back) corresponding respectively to the sum of the four metrics (IndexOfUse, SkipUse, IntraUse and BackUse) of all layers of the given solution.

TABLE I. ANALYZED PROJECTS

Project	*Number of files*	*LOC*	*Number of packages*	*Package dependencies*
JReversePro 1.4.1	85	18 238	11	31
JHotDraw 7.0.6	310	51 801	24	89
JUnit 4.10	162	10 402	28	104
JFreeChart 1.0.14	596	209711	37	207

TABLE II. LAYERING RESULTS

		JReverseP.	*JHotDraw*	*JUnit*	*JFreeCh.*
Setup 1 α =0, β =1, γ = 2, δ = 4	LC	115	385	153	1069
	Adjac	72	632	229	995
	Skip	7	8	3	64
	Intra	101	353	111	629
	Back	0	4	9	78
Setup 2 α =0, β =2, γ = 1, δ = 4	LC	185	816	277	1508
	Adjac	91	523	218	1024
	Skip	29	152	61	164
	Intra	42	312	38	484
	Back	18	10	35	94
Setup 3 α =0, β =2, γ = 1, δ = 8	LC	209	600	297	1945
	Adjac	72	647	229	997
	Skip	7	106	3	67
	Intra	101	243	111	623
	Back	0	1	9	79

Interestingly, most executions of our algorithm on these projects yield the best results for the setups 1 and 3. This has a different explanation depending on the quality of the system's design. For projects including several cyclic dependencies (e.g., JReversePro), the best results are obtained when the packages involved in a cyclic dependency are assigned to the same layer; this is possible only when the intra-dependencies cost β is kept low (i.e., setup 1) or when the back-calls cost δ is set to a high value (e.g., setup 3). Presence of several cyclic dependencies reveals a poor structure of the system resulting from its poor design or evolution. For properly designed systems, we expected that our algorithm would yield the optimal layering using the third setup which largely penalizes back-calls and intra-dependencies with respect to skip-calls. This was the case of the JHotDraw project whose optimal layering is shown in Figure 3. In order to refine this layering, we decided to further partition its lowermost layer into 2 layers. The algorithm then generated a 4-layered system. This allowed us to move from a solution costing 600 (see LC of the third setup for JHotDraw in Table II) to a 4-layered solution costing 582. Provided that the layering cost does not increase, the user may decide to further decompose the resulting layers. The resulting layering for JHotDraw was corroborated by a manual analysis of its source code. Another interesting fact we found during experimentation is that beyond a given value of the back-calls cost δ, the layering returned by the algorithm is the same; i.e., for each of the four analyzed projects, the layering solution is the same for all the values of δ that are greater or equal to a given value (e.g., 8 for JHotDraw and JUnit) provided that the other cost parameters are not altered.

VII. RELATED WORKS

Several software architecture reconstruction approaches were proposed in the literature. Many of these approaches use clustering techniques. Various clustering-based approaches are discussed in [12] and [15]. Most of these approaches aim at finding a clustering of the system that optimizes the modularity of resulting components (e.g., [10, 13, 16]). Our work is more related to the approaches proposed to recover or analyze layered architectures (e.g., [20, 25, 26, 27]). In [25], a framework is proposed for analyzing layered systems to assess the coherence between the description of the architecture given in design documents and the structure of the source code. The framework relies on a set of questions for evaluating the properties of a layered system and a set of metrics that help answering these questions. This empirical study has shown that strict layering is not enforced in layered systems as skip-calls are made extensively however there are no back-calls. Though the framework does not support the recovery of the layered architecture, its results helped us adjust our skip-call cost parameter compared to the intra and back-call cost parameters.

Laval et al. [26] propose an approach, called oZone, which handles undesired cyclic dependencies and decomposes a system into layers. They proposed two heuristics to find dependencies that belong to cycles and impede the finding of layers of a system. These dependencies are tagged by the proposed algorithm as removable and they are ignored when building layers of the system. In [20], the authors proposed 3 layering principles (skip-call, back-call and cyclic dependency) and a set of metrics that measure the violation of these principles. Although these principles are focused on detecting violations, they are related to the abstraction uniformity and incremental layer dependency properties proposed in this paper. However, contrary to both [26] and [20], we do not rely

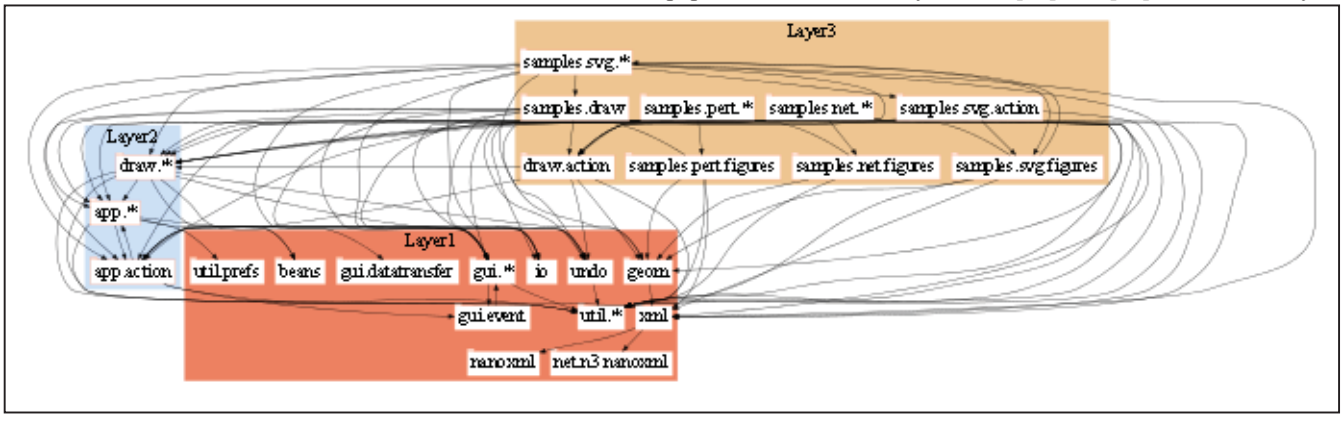


Figure 3: A 3-Layered solution returned by the algorithm for JHotDraw.

on any heuristic to resolve the cyclic dependencies problem. In [27], a semi-automatic approach is proposed to identify software layers. Classes that are used by many other classes are grouped in the lowest layer while classes that rely on many other classes are grouped in the highest layer. The remaining classes are assigned to a middle layer. In both [26] and [27], it is assumed that a module that does not have fan-out dependencies belongs to the lowest-level layer and conversely a module that does not have fan-in dependencies belongs to the highest-level layer. However, when a module represents a common subtask exclusive to components of a middle-level layer, this module will not have any fan-out dependency but still belongs to this middle-level layer. Likewise, a module that starts some specific service of a middle-layer may not have any fan-in dependency but still belongs to this middle-level layer.

VIII. CONCLUSION

The process of reconstructing the architecture of existing systems remains a difficult task and an active research field in software engineering. This process is mandatory in various contexts, spanning from the re-documentation and the understanding of existing systems to their restructuring and migration. In this paper, we presented an approach that reconstructs the software architecture of layered systems. To do so, we defined a set of layering principles, metrics and constraints. These elements enabled us to formalize the layering recovery problem as an optimization problem that was solved using a heuristic search method. Preliminary experimentations on four open source projects yielded interesting results from the approach. The approach is language and platform independent since it relies on the KDM specification standard. In addition, it supports the interaction with users and domain experts to refine the layering results.

While we continue to refine the principles and metrics of our approach, we need to perform more experiments and analyses to properly tune the cost parameters used by our algorithm. In the near future, we are planning to experiment our approach on larger open source systems (e.g., Mozilla and Ant) and compare our results with other approaches. We also intend to explore the usage of domain-specific knowledge information to improve the results; although using domain-specific knowledge is known to confine the scope of the approach.

REFERENCES

[1] R. C. seacord, D. Plakosh, and G. A. Lewis, Modernizing Legacy Systems: Software Technologies, Engineering Process and Business Practices. Addison-Wesley Longman Publishing Co., 2003.

[2] W. Ulrich and P. Newcomb, Information systems transformation : Architecture-Driven Modernization Case Studies, OMG Press, 2010.

[3] M. Shaw and D. Garlan, Software Architecture: Perspectives on an Emerging Discipline, Prentice Hall, 1996.

[4] C. Stoermer, L. O'Brien, and C. Verhoef, “Moving Towards Quality Attribute Driven Software Architecture Reconstruction,” In Proceedings of the 10th Working Conference on Reverse Engineering, 2003.

[5] D.R. Harris, H.B. Reubenstein, and A.S. Yeh, "Recognizers for Extracting Architectural Features from Source Code," The 2nd Working Conference on Reverse Engineering, 1995 (WCRE’95), pp.252-261

[6] OMG Specifications: http://www.omg.org/ [accessed in March 2013]

[7] P. Clements, *et al.*, Documenting Software Architectures: Views and Beyond, Addison-Wesley, 2003.

[8] L. Bass, P. Clements and R. Kazman, Software Architecture in Practice, Addison-Wesley, 2003.

[9] V. Tzerpos and R. C. Holt, “ACDC: An Algorithm for Comprehension-Driven Clustering,” In Proceedings of the Seventh Working Conference on Reverse Engineering, 2000 (WCRE'00), pp. 258-267

[10] B. S. Mitchell and S. Mancoridis. 2007. On the Evaluation of the Bunch Search-Based Software Modularization Algorithm, Soft Comput., 2007, vol. 12, Issue 1, pp. 77-93

[11] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad and M. Stal, Pattern-Oriented Software Architecture: A System of Patterns, John Wiley & Sons, 1996

[12] O. Maqbool and H.A. Babri, Hierarchical Clustering for Software Architecture Recovery, IEEE Transactions on Software Engineering, 2007, vol.33, no.11, pp.759-780

[13] C-H. Lung, M. Zaman and A. Nandi, Applications of Clustering Techniques to Software Partitioning, Recovery and Restructuring, The Journal of Systems and Software, 2004, vol. 73, pp. 227–244

[14] D. Pollet, *et al.*, “Towards A Process-Oriented Software Architecture Reconstruction Taxonomy,” The 11th European Conference on Software Maintenance and Reengineering, 2007 (CSMR '07), pp. 137-148

[15] M. Shtern and V. Tzerpos, Clustering Methodologies for Software Engineering, Advances in Software Engineering, 2012.

[16] Q. Zhang, D. Qiu, Q. Tian, L. Sun, “Object-oriented software architecture recovery using a new hybrid clustering algorithm,” The 7th Int. Conf. on Fuzzy Systems and Knowledge Discovery (FSKD), 2010.

[17] Zimmermann, H., "OSI Reference Model--The ISO Model of Architecture for Open Systems Interconnection," IEEE Transactions on Communications, vol.28, no.4, pp.425,432, Apr 1980.

[18] Clemens A. Szyperski. Component Software. Addison Wesley, 1998.

[19] Eeles, P. “Layering Strategies”, Rational Software White Paper, TP 199, 08/01, 2002.

[20] S. Sarkar, G. Maskeri, S. Ramachandran, “Discovery of architectural layers and measurement of layering violations in source code”, Journal of Systems and Software, Vol. 82 (11), 2009, pp. 1891-1905

[21] Paris, M., "Reuse-based layering: a strategy for architectural frameworks for learning technologies," IEEE Int. Conf. on Advanced Learning Technologies, 2004, pp.455-459

[22] P. B. Kruchten. “The 4+1 View Model of architecture”. IEEE Software, 12(6), Nov. 1995, pp. 42 – 50.

[23] G. El Boussaidi, A. Boaye-Belle, S. Vaucher, H. Mili, "Reconstructing Architectural Views from Legacy Systems", in the 19th Working Conference on Reverse Engineering (WCRE'12), 2012.

[24] Bourquin, F., Keller, R.K,.”High-impact Refactoring Based on Architecture Violations”, In the 11th European Conference on Software Maintenance and Reengineering (CSMR '07), 2007, pp. 149-158.

[25] Lague, B.; LeDuc, C.; Le Bon, A.; Merlo, E.; Dagenais, M.; , "An analysis framework for understanding layered software architectures," Proc. of the 6th IWPC '98, pp.37-44, 1998.

[26] Laval, J., Anquetil, N., Bhatti, M.U., Ducasse, S., “OZONE: Layer Identification in the presence of Cyclic Dependencies”, submitted to Science of Computer Programming, 2012.

[27] G. Scanniello, A. D'Amico, C. D'Amico, T. D'Amico,“Architectural layer recovery for software system understanding and evolution”, Software Practice and Experience, 2010, vol. 40(10), pp. 897-916.

Towards the Establishment of a Reference Architecture for Developing Learning Environments

Ellen Francine Barbosa, Maria Lydia Fioravanti, Elisa Yumi Nakagawa, José Carlos Maldonado
University of São Paulo (ICMC/USP)
São Carlos (SP), Brazil
email: francine, fioravanti, elisa, jcmaldon@icmc.usp.br

***Abstract*—Learning environments are moving away from monolithic applications towards more open, flexible components, capable of interoperating with other learning components. In spite of the diversity of learning environments, there is a lack of uniformity regarding their basic functionalities. Thus, the establishment of core functionalities represents an issue to the development of these environments. In a different but related perspective, reference architectures have emerged as an alternative for promoting reuse of design expertise and facilitating the development of systems. Aspect-Oriented Software Development (AOSD) has also arisen as a promising software development approach, contributing to a better Separation of Concerns (SoC). Motivated by this scenario, in this paper we discuss the establishment of EDUCAR, an aspect-oriented reference architecture for developing learning environments. EDUCAR has been constructed by means of a systematic development process. As a consequence, architectural requirements, concepts and crosscuting concerns regarding the learning environments domain have been established as well. The results achieved from a preliminary evaluation of EDUCAR suggest the adequacy of the proposed architecture with respect to the learning environments domain.**

***Keywords*-learning environment; reference architecture; aspect-oriented software development**

I. INTRODUCTION

The Internet and advances in Information and Communication Technologies have changed the educational setting, both in traditional and distance learning. As a result, there has been a change in the way that educational content is designed, developed and delivered to learners.

Faced with these transformations, in recent years there is an increasing demand for open, scalable and flexible learning environments [1]. In short, a learning environment consists of a software application that automates the administration, tracking and reporting of training events. Good examples are Moodle[1], Sakai[2] and dotLRN[3].

According to Ellis [2], a robust learning environment should be able to: (i) centralize and automate administration; (ii) use self-service and self-guided services; (iii) assemble and deliver learning content rapidly; (iv) consolidate training initiatives on a scalable web-based platform; (v) support portability and standards; and (vi) personalize content and enable knowledge reuse. Each environment has also specific features concerning pedagogical, technical and management issues. More important, learning environments should integrate with other enterprise application solutions used by human resources and accounting, enabling the management to measure the impact, effectiveness and overall cost of training initiatives.

Despite the diversity of existing learning environments, a common weakness observed is the lack of uniformity regarding their basic functionalities. Actually, the establishment of core functionalities constitutes a significant issue to the development of such environments.

Reference architectures can play a fundamental role in this perspective, aiming at guiding the building of learning environments. Basically, a reference architecture refers to a special type of software architecture that captures the essence of the architectures of a collection of systems in a given domain [3]. Its purpose is to provide guidance for the development, standardization and evolution of system architectures of a specific domain.

In a different but related perspective, Aspect-Oriented Software Development (AOSD) has also arisen as a promising software development approach, contributing to a better Separation of Concerns (SoC), resulting in more reusable, maintainable and evolvable software systems [4].

Motivated by this scenario, in this paper we discuss the establishment of EDUCAR – an aspect-oriented reference architecture for developing learning environments. The main goal of EDUCAR is to provide guidance for the architectural design of new versions of learning environments as well as promoting a better reuse, evolution and maintenance of the existing ones. EDUCAR has been constructed by means of a systematic process, referred to as ProSA-RA [5]. ProSA-RA focuses on how to deal with architectural aspects as well as on how to represent and evaluate these architectures.

The remainder of this paper is organized as follows. In Section II, related work is briefly presented. In Section III we provide an overview of ProSA-RA. In Section IV, we

[1] http://moodle.org
[2] http://sakaiproject.org
[3] http://dotlrn.org

describe EDUCAR, focusing on its development through the application of ProSA-RA. In Section V, we summarize our contributions and the main perspectives for further work.

II. RELATED WORK

Arch-int et al. [6] proposed a reference architecture to promote interoperability of existing learning systems by means of web services. Still regarding interoperability, Habraken [1] described a reference architecture in which learning components from different suppliers could be integrated into one e-learning solution for a customer. Anido et al. [7] proposed a reference architecture that identifies common, standardized software services for distributed e-learning systems. In the same perspective, Palanivel and Kuppuswami [8] designed a service-oriented reference architecture for personalized e-learning systems. In Schmidt's work [9], a layered reference architecture for context-aware learning support systems was defined. Li et al. [10] also presented a layered reference architecture for learning environments, aiming at building scalable environments to support an arbitrary number of users, while providing them with a personalized environment.

It is important to point out that reference architectures for developing learning environments are still very specific, sometimes considering only one type of environment, for instance, e-learning systems. Besides that, SOA (Service-Oriented Architecture) [11] has been the basis for almost all architectures, such as [6], [8], [9]. Also, the SoC provided by Aspect-Orientation approach has not been widely investigated for the learning domain. Last but not least, none of the proposed architectures was developed by using a process for designing, representing and evaluating them. Thus, addressing such issues by adopting a systematic process to the establishment of a more general, aspect-oriented reference architecture for developing learning environments is the focus of our work.

III. PROSA-RA: AN OVERVIEW

ProSA-RA is a process that systematizes the design, representation and evaluation of aspect-oriented reference architectures [5]. The process comprises four basic steps.

Firstly, the main information sources are selected and investigated (*Step RA-1*). These sources must provide information about processes, activities and tasks that can be supported by software systems of the target application domain. ProSA-RA highlights people, sofware systems, publications and domain ontologies as the most relevant information sources to be considered.

Secondly, the architectural requirements of the reference architecture are identified, describing the common functionalities and configurations presented in systems of the target domain (*Step RA-2*). To do so, four main tasks are performed: (i) identification of the system requirements (functional and non-functional); (ii) establishment of the reference architecture requirements (architectural requirements); (iii) identification of the domain concepts, i.e., each requirement is associated to a concept that better addresses it; and (iv) classification of domain concepts in crosscutting or non-crosscutting concerns.

The third step consists of establishing the architectural description of the reference architecture (*Step RA-3*). To build this description, well-known architectural styles and patterns (e.g., client-server, three-tier architecture and MVC) as well as a combination of them and other styles can be considered. Besides that, ProSA-RA proposes some architectural views to describe reference architectures: (i) module view: shows the structure of the architecture in terms of packages, classes, containment, specialization/generalization and relationships; (ii) runtime view: shows the structure of the systems (that will be built based on the reference architecture) when they are executing; (iii) deployment view: describes the hardware, the software system or subsystems that are installed on that hardware, and the network connections, if they exist; and (iv) conceptual view: describes the understanding of each domain concept or term used in the reference architecture.

Finally, an evaluation of the resulting architecture is conducted by means of a checklist-based inspection approach (*Step RA-4*). The checklist is composed of 32 multiple choice questions. The main idea is to guide reviewers on detecting defects in the documents related to the reference architecture design.

ProSA-RA has been applied in the establishment of reference architectures for several domains, such as visual mining, software engineering environments, mobile robotics, computer games and marine systems. Next, we discuss its use in the learning environments domain.

IV. ESTABLISHMENT OF EDUCAR

In this section we present EDUCAR, focusing on its development according to ProSA-RA. The establishment of EDUCAR involved different skills from four specialists: one domain expert, one system analyst and two software architects.

RA-1: Information Sources Investigation

We began the establishment of EDUCAR by choosing a set of learning environments to be considered as information sources in this domain. Our selection was based on the following criteria: (i) the first initiatives on learning environments, such as WebCT/Blackboard[4]; (ii) environments widely adopted currently, such as Moodle and Sakai; and (iii) environments with specific features, such as IWT[5] (which explores the use of ontologies) and AdaptWeb[6] (which addresses adaptive issues on learning).

[4]http://www.blackboard.com
[5]http://www.didatticaadistanza.com
[6]http://adaptweb.sourceforge.net

Experts' knowledge was also considered. Both proprietary and open source initiatives were investigated. At the end, 12 learning environments were considered. The complete list of systems selected is available at [12].

Additionally, we also conducted a systematic review in order to identify publications addressing characteristics, functionalities and requirements of architectures of learning environments. In the first round of the systematic review, 60 research works were retrieved. Then, based on the inclusion criterion defined, a subset of these works was selected. In the end, 40 works were considered for full reading. Table I summarizes the systematic review protocol.

At the end of Step RA-1, we were able to get considerable knowledge about the learning environments domain. This knowledge acts as a basis for the development of EDUCAR.

RA-2: Architectural Requirements Establishment

Based on the knowledge obtained from Step RA-1, we were able to identify 13 categories of functionalities with respect to learning environments: (i) *Content*: addresses issues of authoring and presentation of the educational content; (ii) *Learner's Assessment*: addresses issues of authoring, presentation and feedback of the learner's assessments; (iii) *Communication*: addresses issues of communication and collaboration among users of the learning environment; (iv) *Adaptation*: deals with adaptive issues on learning; (v) *Documentation*: deals with course, user and system documentation; (vi) *Course Coordination*: addresses coordination issues related to the course; (vii) *System Administration*: deals with administrative issues related to the system; (viii) *Storage*: deals with the storage of course and user information; (ix) *Standards Adequacy*: addresses issues of adequacy to learning standards (such as IMS, SCORM and LOM); (x) *Multilanguage*: deals with the system support for different languages; (xi) *Interface*: deals with the different available interfaces provided by the system; (xii) *Interaction Mechanisms*: deals with the interaction mechanisms provided by the system; and (xiii) *Access Mechanisms*: deals with the mechanisms provided by the system to access user and course information.

Each category was divided in subcategories and, for each of them, a set of functionalities was identified. Consider, for instance, the *Learner's Assessment* category. *Authoring of questionnaires*, *Delivery of exercises* and *Performance reports* are functionalities addressed by the subcategories *Authoring*, *Presentation* and *Feedback*, respectively. Table II shows the subcategories and functionalities identified for *Learner's Assessment*. Altogether, 18 functionalities were identified for this category.

Based on the set of functionalities established, 123 system requirements for learning environments were identified. Table III shows a small part of them. Then, a detailed analysis of such requirements was conducted to identify the architectural requirements. For instance, system requirements related to the authoring of assessment were mapped to a single architectural requirement (*Allow authoring of assessment*). The 123 system requirements were mapped to a set of 18 architectural requirements (Table IV (column 2)).

[1]http://portal.acm.org

[2]http://ieeexplore.ieee.org

Table III
EXAMPLES OF SYSTEM REQUIREMENTS AND ARCHITECTURAL REQUIREMENTS

Nr	System Requirement	Architectural Requirement
...	...	...
39	Allow authoring of tasks	Allow authoring of assessment
40	Allow automatic authoring of assessments	Allow authoring of assessment
41	Allow authoring of quizzes	Allow authoring of assessment
42	Allow authoring of questionnaires	Allow authoring of assessment
43	Allow authoring of online assessments	Allow authoring of assessment
44	Allow authoring of multiple choice tests	Allow authoring of assessment
45	Allow authoring of exams	Allow authoring of assessment
...	...	...

From the architectural requirements, we were able to determine the main concepts related to the learning environments domain (Table IV (column 3)). For instance, architectural requirements 3 and 4, related to synchronous and asynchronous communication, were mapped to the concept *Communication*; architectural requirements 6 and 7, related to authoring of learning materials and assessments, were mapped to the concept *Authoring*; and so on.

Finally, we identified which of these concepts had crosscutting characteristics (Table IV (column 4)). Among all concepts analyzed, *Personalization* was the only one that presented a crosscutting characteristic, i.e., it is a crosscutting concern. Indeed, functionalities related to personalization are generally spread for several parts (modules) of a learning environment and personalization can occur as needed by the user (no matter if he/she is an administrator, instructor, monitor or learner). At the end, 13 concepts were established, one of them presenting a crosscutting characteristic.

RA-3: Reference Architecture Design

Considering the architectural styles and patterns found in the learning environments previously analyzed, we chose to build EDUCAR based on well-known and consolidated architectural styles of interactive systems and Web systems: the architectural pattern MVC and the three-tier architecture. To adequately represent EDUCAR, we built its architectural views (module view, runtime view and deployment view) using UML. For the sake of space, only the module view (Figure 1) is discussed herein.

The module view is composed of three tiers/layers. The *persistence layer* corresponds to the set of data that needs to be stored by the learning environment. A database (managed by a DBMS), a repository (a central place in which an aggregation of data is kept and maintained in an organized

Table I
SUMMARY OF THE SYSTEMATIC REVIEW PROTOCOL

Name	Description
Research Question	Which features, functionalities and requirements of architectures of learning environment are found in the literature?
Keywords	learning management system, course management system, personal learning environment, requirement, feature, characteristic
Search String	("learning management system" OR "course management system" OR "personal learning environment") AND (requirement OR characteristic OR feature)
Publication Database	ACM Digital Library[1] and IEEEXplore[2]
Inclusion Criterion	Does the publication describe a characteristic, a functionality and/or a requirement of architectures of learning environments?

Table II
SUBCATEGORIES AND FUNCTIONALITIES OF THE *Learner's Assessment* CATEGORY

	AdaptWeb	Atena	AulaNet	CoL	Dotlearn	Eureka	IWT	Moodle	Sakai	Teleduc	Tidia	WebCT
Authoring												
Authoring of tasks			✓					✓	✓	✓		✓
Automatic authoring of assessments					✓			✓	✓			
Authoring of quizz								✓	✓			
Authoring of questionnaries		✓						✓				
Authoring of online assessments		✓	✓		✓			✓	✓	✓		✓
Authoring of multiple choice tests				✓					✓			
Authoring of exams			✓									
Presentation												
Delivery of exercises			✓					✓	✓	✓	✓	✓
Nook study					✓			✓	✓		✓	
Feedback												
Assessment research								✓				
Opinion research								✓				
Peer assessment								✓				
Access statistics			✓			✓	✓	✓		✓		✓
Tracking systems			✓		✓			✓	✓	✓		✓
Grades		✓						✓	✓			
Performance reports		✓		✓				✓	✓			✓
Participation reports		✓	✓		✓		✓	✓	✓	✓		
Frequency reports		✓	✓					✓	✓	✓		

Table IV
DOMAIN CONCEPTS FOR LEARNING ENVIRONMENTS

Nr	Architectural Requirement	Concept	Cc
1	Deliver course information	Course Documentation	N
2	Deliver system information	System Documentation	N
3	Allow synchronous communication	Communication	N
4	Allow asynchronous communication	Communication	N
5	Allow content adaptation	Adaptation	N
6	Allow authoring of learning material	Authoring	N
7	Allow authoring of assessment	Authoring	N
8	Allow learner's assessment	Assessment	N
9	Deliver learning material	Presentation	N
10	Provide feedback of assessment	Assessment	N
11	Manage course	Course Management	N
12	Manage users	Users' Management	N
13	Manage system	System Management	N
14	Allow adequacy to learning standards	Standards Adequacy	N
15	Provide support for multilanguage	Personalization	Y
16	Provide support for templates	Personalization	Y
17	Provide system security	System Administration	N
18	Manage learning material	Presentation	N

way) or a file system (set of files and where they are placed logically for storage and retrieval) can be used.

The *presentation layer* refers to server side modules which are responsible for the user interface presented in client side. This layer is composed of: (i) the `Controller` element, which processes events (typically user actions) and invokes the functionalities implemented by the `Model` element; and (ii) the `View` element, which contains the user interface.

The *application layer* contains the `Model` element, which aggregates the functionalities related to the core of learning environments. This layer comprises six modules:

– `content_authoring`: One of the core modules of EDUCAR, it is responsible for the development of educational content (i.e., materials and assessments). Issues addressed in this module are related to: (i) structuring and modeling of content: involves the identification and representation of concepts and their inter-relationships, and instructional activities (exercises, practical assignments, lab tasks, and so on.); (ii) editing of content: involves the creation of documents and media (e.g., texts, slides, images, and videos); (iii) automatic generation of content: relevant when the content is represented in a machine-readable format; (iv) sharing, reuse and integration of content: they

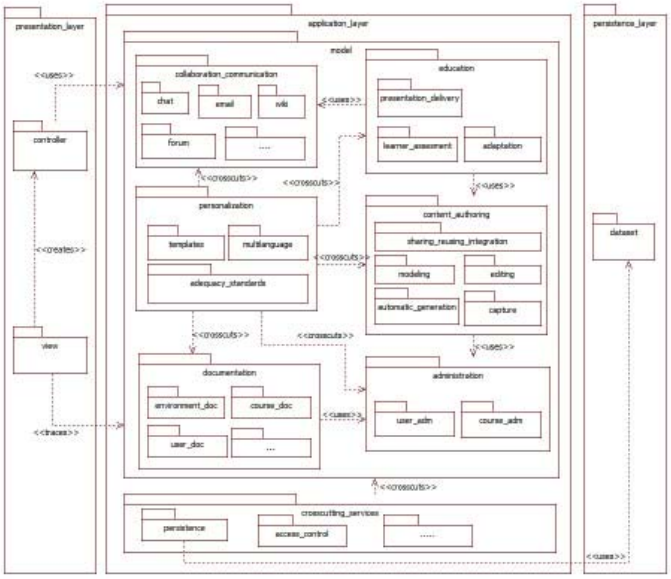


Figure 1. EDUCAR: Module View

refer to the use of domain ontologies, dictionaries of terms, glossaries, among others, as supporting mechanisms to the development and evolution of content; and (v) capture of content: refers to the capture and storage of the discussions and experiences that occurred during classes and later integration and synchronization of the multiple streams of information captured (e.g., audio, video, and notes).

– `education`: Another core module of EDUCAR, it is responsible for the presentation and delivery of educational content as well as the learners' assessments. It also covers issues related to the content adaptation. According to parameters such as background, objectives, interests and learning profile of each learner, different ways of structuring and navigating for the same content is established.

– `collaboration_communication`: Gathers supporting tools for synchronous and asynchronous communication (e.g., chats, web conferences and e-mails) and for collaborative work (e.g., wikis and forums).

– `administration`: Covers administrative issues, focusing on the management of users and courses. Regarding the management of users, it addresses issues of authentication and establishment of access levels to the users, as well as inclusion, exclusion and update of the users' information. Reports of learners' performance, participation and frequency are also considered. In terms of course management, the module covers topics such as course inclusion, exclusion and update, generation of statistics, and course backup, among others.

– `documentation`: Responsible for providing mechanisms for the management and storage of documents. Thus, documentation on the learning environment (e.g., help and FAQ), users and courses (e.g., objectives, lesson plans, schedule of classes and course FAQ) must be considered. Other types of documentation and/or relevant information to the environment can also be considered in this module.

– `personalization`: *Personalization* was classified as a crosscutting concern. So, this module must encapsulate a crosscutting concern and, therefore, it is an architectural aspect. It establishes mechanisms for the creation and use of templates, support of multilanguage and adequacy to standards (such as IMS, SCORM and LOM). As a crosscutting concern, it affects all other modules (represented by dashed lines and stereotype <<crosscuts>>). Indeed, the functionalities implemented in this module impact other modules, changing their behavior to address functionalities related to personalization.

To compose an integrated learning environment, communications among modules/packages must also be established. Regarding relationships among the packages/modules, the module `content_authoring` communicates with the module `education` to make the content available to learners. It also communicates with the module `administration` to get information about the users' access levels for determining, for example, if a given user is allowed to create content in some course. The module `education` communicates with the module `collaboration_communication` since collaborative and communication tools can also be used to support the learning activities. Finally, the modules `administration` and `documentation` communicate each other since `documentation` is also responsible for documenting the information about users and courses managed by the module `administration`.

To promote SoC in the learning environments built from EDUCAR, each module in the application layer was designed as separated as possible, enabling that each one can be designed and implemented as an independent tool (or subsystem). They can be further aggregated in a learning system, composing an integrated environment. Particularly, the module `personalization`, which aggregates a crosscutting concern, can be also developed as an independent tool.

Notice that during the development of all independent tools but for `personalization`, the principle of obliviousness [13], proposed by the AOSD community, can be considered. This principle implies that these tools could be oblivious from the fact that their semantics may be modified at some later stage through the introduction of a tool that implements a crosscutting concern; in this case, a tool that automates `personalization`.

Additionally to the modules in the package `model`, we have also foreseen the package `crosscutting_services`. It is composed by other architectural aspects that automate services considered crosscutting concerns, such as persistence and access control. Observe that this package is not directly related

to learning environments domain, but considering the crosscutting nature of the services it provides, it was inserted into EDUCAR aiming at improving the SoC.

RA-4: Reference Architecture Evaluation

Aiming at evaluating EDUCAR, we conducted an inspection on its architectural views using the 32 questions of the checklist provided by ProSA-RA. We evaluate quality characteristics of the reference architecture: maintainability, performance, security, usability, portability and reuse. Examples of questions are: *Are the interfaces among modules well-defined?* and *Is it easy to remember how the modules are organized and are related among them?* We also evaluate the architectural description through identification and elimination of defects related to omission, ambiguity, inconsistency, as well as strange and incorrect information. Examples of questions are: *Is the reference architecture develop based on the domain terms that are widely used and well-understood?* and *Are the architectural views sufficient and adequate to represent all elements of the domain?*

The results achieved from this preliminary evaluation suggest the adequacy of EDUCAR with respect to the learning environments domain. The consistence among the architectural views of EDUCAR was also verified. Details regarding the evaluation conducted can be found at [12].

V. CONCLUSIONS AND FURTHER WORK

In this paper we described an aspect-oriented reference architecture for learning environments. The main contribution of EDUCAR lies on providing guidance for architectural design of new learning environments as well as for evolution and maintenance of the existing ones.

As a further work, we point out the need of conducting a more complete evaluation of EDUCAR. This evaluation has been planned and will require efforts to develop a considerable set of learning environments. Quantitative studies involving detailed experiments to measure the effort required to use EDUCAR have to be conducted as well. Moreover, the architectural description of EDUCAR has been currently completed through creation of the conceptual view using thesauri, what will contribute to its easier understanding and dissemination.

As a final remark, we highlight that knowledge about any domain emerges, evolves and consolidates over time. Reference architectures must encompass this new knowledge and must also be continually updated. So, EDUCAR must also be continually evolved, inserting these new types of knowledge in order to not deteriorate.

ACKNOWLEDGMENT

The authors would like to thank the Brazilian funding agencies (FAPESP, CAPES, CNPq) and INCT-SEC (CNPq 573963/2008-8, FAPESP 08/57870-926) for their support.

REFERENCES

[1] J. Habraken, "Reference architecture for e-learning solutions," Master's thesis, Open University, UK, Jan. 2008.

[2] R. K. Ellis, "Field guide to learning management systems," ASTD, Tech. Rep., 2009.

[3] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*. Addison-Wesley, 2003.

[4] G. Kiczales, J. Irwin, J. Lamping, J. Loingtier, C. Lopes, C. Maeda, and A. Menhdhekar, "Aspect-oriented programming," in *11th Eur. Conf. on Object-Oriented Programming*, Jyväskylä, Finland, 1997, pp. 220–242.

[5] E. Y. Nakagawa, F. Ferrari, M. M. F. Sasaki, and J. C. Maldonado, "An aspect-oriented reference architecture for software engineering environments," *The Journal of Systems and Software*, vol. 84, no. 10, pp. 1670–1684, 2011.

[6] N. Arch-int, C. Lursinsup, and P. Sophatsathit, "A reference architecture for interoperating existing e-learning systems using metadata and web services model," in *CIMCA-IAWTIC'05*, 2005, pp. 891–896.

[7] L. Anido, M. Llamas, M. J. Fernández, J. Rodríguez, M. Caeiro, and J. Santos, "A standards-driven open architecture for learning systems," in *ICALT'01*, Madison (WI), Aug. 2001, pp. 3–4.

[8] K. Palanivel and S. Kuppuswami, "Service-oriented reference architecture for personalized e-learning systems (SORAPES)," *International Journal of Computer Applications*, vol. 24, no. 5, pp. 35–44, Jun. 2011.

[9] A. Schmidt, "Impact of context-awareness on the architecture of learning support systems," in *Architecture Solutions for E-learning Systems*, C. Pahl, Ed. Idea Group Publ., 2007.

[10] Q. Li, R. W. H. Lau, E. W. Leung, F. Li, V. Lee, B. W. Wah, and H. Ashman, "Emerging internet technologies for e-learning," *Internet Computing*, pp. 11–17, jul/aug 2009.

[11] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-oriented computing: A research roadmap," *International Journal of Cooperative Information Systems*, vol. 17, pp. 223 – 255, 2008.

[12] 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," University of São Paulo (ICMC-USP), São Carlos, SP, Tech. Rep., Nov. 2012.

[13] R. Filman and D. Friedman, "Aspect-oriented programming is quantification and obliviousness," in *OOPSLA 2000*, Minneapolis, MN, 2000, pp. 21–35.

Testing Configurable Architectures For Component-Based Software Using an Incremental Approach*

Chuanqi Tao[1], Bixin Li[1]*, Jerry Gao[2]
[1]*School of Computer Science and Engineering, Southeast University, Nanjing, Jiangsu, China*
[2]*School of Computer Engineering, San Jose State University, San Jose, CA, USA*

***Abstract*—Configurable software lets users customize applications in many ways, such as different configurable environments, diverse functions, and various configurable architectures. As the advance of software component technology, engineers encountered different issues and challenges in testing and automation of configurable components and component-based programs. In previous work, we proposed an approach to configuration testing based on a semantic tree, to model, present, and analyze diverse composite components and configurable software. Various configurations are modeled based on the spanning tree, which is a subtree of semantic tree. For realistic component-based software, there might exist a number of configurable combinations, i.e., the number of subtrees can be unimaginably high. Thus configuration testing becomes very complex and not cost-effective. In this paper, we propose an incremental approach to testing configurable architectures of component-based software based on the semantic tree model. Compared to our existing work, the new approach could reduce the testing complexity significantly. The initial study results indicate the proposed approach is feasible and effective in testing configurable architectures for component-based software.**

***Keywords*-test modeling and analysis; component-based software; configurable testing; test complexity**

I. INTRODUCTION

Today, component-based software engineering is a widely-used approach in software construction. Many modern software systems are constructed based on reusable components, such as third-party components and in-house built components. Hence, testing and retesting components and component-based systems has been a very hot research subject in the past decade [1–5]. As the advance of software component technology, the complexity of modern software and complex components increased from functional component box, frameworks/middleware, and configurable/customizable complex components. Software users (or clients) are allowed to configure, select, and customize them based on their functional requirements, desirable environments, and architectures. Now engineers lack well-defined test process, adequate test models and criteria, as well as test automation solutions for configurable components and systems. Our previous paper [6] used a semantic tree model-based approach to to assist engineers to perform test modeling and analysis for configurable component-based systems. This model can be used to present diverse software component configurations statically or dynamically. However, testing systems of this kind presents significant challenges to practitioners in the field, due to the large number of possible combinations. Currently, it is infeasible to completely test configurable systems before release [3, 7]. In our previous work, various configurations are modeled using the spanning tree model, which is a subtree of semantic tree. In realistic component-based software, the variety of configurable architectures can increase to an unacceptable level, resulting in configuration testing with high complexity.

In this paper, we focus on the configurable architectures which allow users to select different organization and composition structures based on available components and building parts. Therefore, the main research question in this paper is how to effectively testing configurable architecture, while the testing complexity can be reduced. A new approach to testing configurable software systems is presented in the paper, specifically aiming at modeling configurable architectures. In this approach, we do not test architecture thoroughly using diverse configurations modeled by spanning trees. Instead, we use an incremental approach to modeling configuration testing of system architecture. The new approach can reduce testing complexity significantly, yet achieve adequate test coverage. The paper has two primary contributions in software component testing. Firstly, it provides an incremental approach to modeling, presenting and analysis of diverse configurations in configurable components for configurable selective architectures. Secondly, it can reduce test complexity for configurable architectures compared with previous work.

The paper is structured as follows. The next section discusses the basic backgrounds. Section IV presents the incremental approach to configuration testing. Section V reports the results of empirical study. The related work is provided in Section VI. Conclusion and future work are summarized in Section VII.

II. BACKGROUNDS

A. *Configuration Testing*

Configurable software is a program supported with a capability that allows users to make various configuration decisions statically or dynamically to generate different deployment instances based on their desires. Many modern frameworks and component-based software allow users to make configuration decisions such as environment configuration, function configuration, and architecture configuration.

Configuration testing is utilized to validate configurable components and programs to achieve adequate test criteria and support test automation. Configurable component refers to a composable component, which not only provides a specified contracted interface, domain-specific service function and development solution with necessary artifacts. Similarly, configurable software is a program

*Supported partially by the National Natural Science Foundation of China under Grant No. 60773105 and No. 60973149, and partially Supported by Doctoral Fund of Ministry of Education of China under Grant No. 20100092110022, and partially by National High Technology Research and Development Program under Grant No. 2007AA01Z141 and No. 2008AA01Z113.

*Correspondence to: bx.li@seu.edu.cn

Table I
THE NOTATIONS OF SEMANTIC RELATIONS IN THE SEMANTIC TREE

Relations	Semantic descriptions (P is a parent node, and C_i is its child node)
$EOR(P, \langle C1, C2 \rangle)$	P-Node must be configured with two child nodes C1 and C2 exclusively
$AND(P, \langle C1, ..., Cn \rangle)$	P-Node must be configured with all of its child nodes C_1,..., C_n
$SELECT-1$ $(P, \langle C1, ..., Cn \rangle)$	P-Node must be configured with one of its selective child nodes C_1,..., C_n
$SELECT-M$ $(P, \langle C1, ..., Cn \rangle)$	P-Node must be configured with M selective nodes from its child nodes C_1,..., C_n

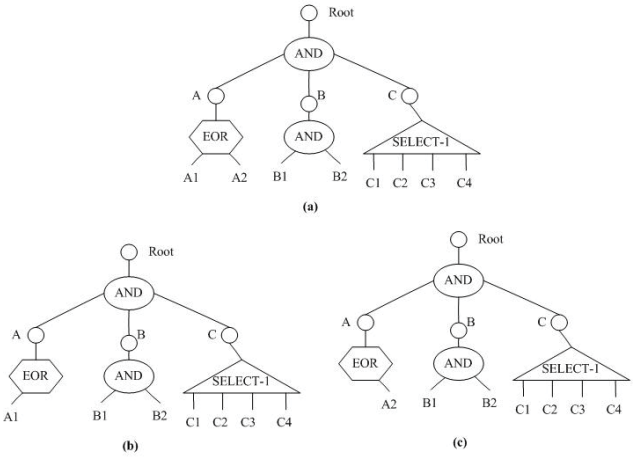


Figure 1. A Sample Semantic Tree and Its Selected Semantic Spanning Trees

supported with a capability that allows its users to make configuration selection. Also various components can be selected configured or customized by Software users. There are numerous models in software testing but very few are exactly suitable to present diverse configurations. In a configurable component based system the program can be configured by using selective components and parts to form different architectures. In this case software testing must address and cover these diverse architectures to make them adequately covered wherever they are required. Although in the past decades there are numerous published papers addressing on testing software components and component-based systems, most of them only focus on basic component testing issues and solutions without considering configurable software architectures and organizations, configuration functions and services in modern components and component-based software.

B. Semantic Tree and Semantic Spanning Tree

In previous work [6], we introduced a sematic tree model to test configurable component-based software. The tree nodes present configurable parts (or elements), such as configurable components. The links present different semantic relations between nodes.

A semantic tree model G_{ST} can be formally defined as 3-tuple = (N, E, R), where

- N is a set of tree nodes. There are three types of nodes: a) a single root node, b) intermediate nodes (or parent nodes), and c) leaf nodes.
- E is a set of links between nodes. Each link connects a parent node and one of its child nodes in a tree. Each link show a part of a semantic relation between a parent node and its child nodes.
- R is a set of relations, and each item in R has a semantic label that presents a semantic relation between a parent node and its child nodes. There are four types of semantic relations with labels: EOR, AND, SELECT-1, and SELECT-M. Their detailed semantics are given in Table I.

We adopted Semantic Tree to model diverse configurable environments, architectures, and functions respectively [6]. To support the model-based analysis, we introduced a concept of semantic spanning trees based on the semantic tree model to present the various configurations. A semantic spanning tree G_{SPT} is a subtree of a given semantic tree G_{ST}. Unlike regular spanning trees, a semantic spanning tree G_{SPT} for G_{ST} only can be derived based on the given configuration semantic properties. As shown in Figure 1, (a) is a semantic tree model, and its two sample spanning trees are shown in (b) and (c).

III. AN INCREMENTAL APPROACH TO MODEL CONFIGURATION TESTING ON SYSTEM ARCHITECTURE

In previous work [6], we proposed a semantic spanning tree model to present the diverse configurable functions, environment, and architectures. However, we discovered in our empirical studies that the configuration testing could be very complicated due to the large amount of configuration choices. Thus, exhausting configuration testing is time-consuming and not cost-effective. Hence, cost-effective methods are needed for configurable component-based software testing.

From our observations and experiences, a large amount of semantic spanning trees need to be generated for configurable architecture with various configurations, especially in retest context. In addition, the test complexity for configuration can be extremely high. As shown in Figure 1, there exists two semantic spanning trees for the component configuration in terms of our previous approach. These two trees have similar architecture except the configuration for component *A*, where one is configured with A1, and the other is configured with A2. According to previous approach, these two spanning tree need to be tested respectively. In our empirical studies, we have used the semantic tree model to present the different configurable architectures for a configurable component-based elevator simulation system, which is developed by SJSU students using modern component-based technology [6, 8]. This system provides a set of configurable components, including Floor Panel, Door, User Panel, and Car components. A user interface is provided to its customer to support a user to select and build diverse elevator system instances based on their need. Figure 2 shows the semantic tree model for the 'lift' component (known as a ICar) of the elevator system.

The semantic spanning tree usually refers to most of the nodes and links in semantic tree, which causes configuration testing with high complexity. In addition, if failures are encountered, all components might be subjected to the possibility of hosting faults. Here, Inspired from the traditional integration testing method, we propose a bottom-up configuration testing method, which integrates configurable elements such as architectures and conducts configuration testing incrementally. Compared to our existing work, the new approach could reduce the testing complexity significantly yet

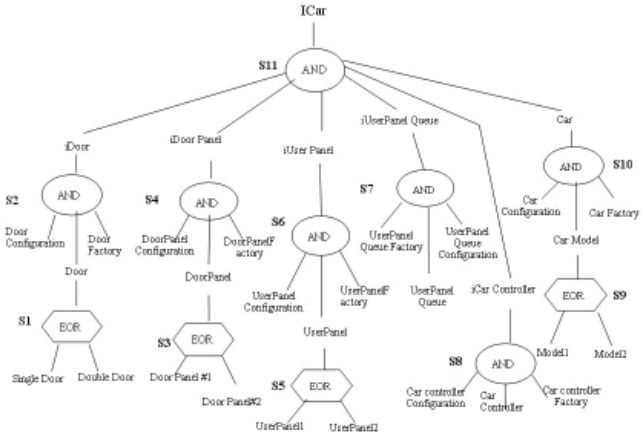


Figure 2. The integrated semantic tree for incremental configuration testing

achieve the testing coverage requirement. During testing of configurable architecture for component-based software, each semantic relation and its corresponding parent nodes, and child nodes could form a configurable unit (or sub-tree). Those configurable units are independent from each other from the system architecture view. In other words, we can perform configuration testing for each configurable unit separately, and then integrate them together to finish the whole configurable architecture testing for component-based system.

In a configurable component-based system, the program can be configured by using selective components and parts to form different architectures. software testing must address and cover these diverse architectures to make they are adequately covered whenever they are required. For instance, in Figure 2, a user can configure the *door* component with two models: single door and double door. For each configurable factor in the semantic tree model, the configuration relation could be independent, i.e., regardless of the configuration relation of parents and children. In Figure 2, if the *door* component is configured with *single door* and *double door*, then we do not need to test *door* configured with each component respectively. Thereby we propose an incremental configuration testing approach, which starts from the leaves of the semantic tree and works up towards the top level.

Algorithm 1 Incremental Configuration Testing

Declare: G_{ST}: semantic tree; $G_{ST} - relation$: relation in G_{ST} ; $G_{ST} - node$: nodes in G_{ST};
Declare: CITO (Configuration-Incremental-Test-Order)
Begin
Postorder $G_{ST} - node$ in terms of $G_{ST} - relation$ R;
Add the traversed $G_{ST} - relation$ nodes (configurable unit) in CITO;
for each configurable unit in CITO **do**
  Semantic-Spanning-Tree($G_{ST} - node$, N_{SPT})
  perform configuration testing on N_{SPT};
end for

Unlike the previous semantic and spanning tree method, the new approach integrates configurable elements like architectures and conducts configuration testing incrementally, i.e., it starts from the leaves of the semantic tree and works up towards the top level. Here, each semantic relation and its corresponding parent nodes, and child nodes form a configurable unit (or sub-tree). From the bottom level to top level of the semantic tree model, each configurable unit can be tested incrementally.

Each semantic relation and its corresponding parent nodes, and child nodes form a configurable unit (or sub-tree). From the bottom level to top level of the semantic tree model, each configurable unit can be tested incrementally. An incremental configuration testing algorithm is given in **Algorithm 1**. In the algorithm, we firstly perform postorder traversal for the semantic tree in order to obtain the testing sequence. Each configured node can be constructed with its child nodes to form a configuration unit. Then, we use the spanning tree algorithm proposed in previous work [6] to do configuration testing for each unit.

For instance, a sample configuration testing order is shown in Figure 2, where S1, S2,..,S11 represent the configurable unit with increasing test order of incremental approach. For example, configurable unit S1 is tested firstly, i.e., door component is configured with single door and double door respectively. Then, we continue testing unit S2, i.e., *idoor* is configured with *door configuration*, *door factory*, and *door*. Sequenced unit S3,...,S11 can be performed in a similar way. For each configurable unit in the order, configurable semantic relations are based on the notations described in Table I.

IV. Test Complexity Analysis for Configurable Architecture

The existing research indicates that complexity can be used to estimate the cost or effort required to design, code, test, and maintain software, as well as predict errors or faults that might be encountered during testing [9, 10]. In addition, complexity measurement provides a guideline and cost indicator for software maintenance. Moreover, through the complexity comparison, we can select a cost-effective configuration testing approach.

For any configurable node N_{Ci} in N_S of G_S in semantic tree model, its test complexity can be computed based on its semantic relation with child nodes. Let $T - complexity(N_{Ci})$ be the configuration complexity for its architectures. To support the evaluation of test complexity of diverse configurable architectures, we provide a detailed computation method for test complexity below.

- Suppose the node N_{Ci} having EOR semantic relation with its child nodes, then it has two different architectures, thus, its configuration complexity will be 2.
  $T - complexity(N_{Ci}) = 2$ (1)
- Suppose the node N_{Ci} having AND semantic relation with its child nodes, then its configuration complexity will be 1.
  $T - complexity(N_{Ci}) = 1$ (2)
- Suppose the node N_{Ci} having a SELECT-1 semantic relation with its child nodes, then its configuration complexity will be n if we assume that n components are allowed to be selected.
  $T - complexity(N_{Ci}) = n$ (3)
- Suppose the node N_{Ci} having a SELECT-M semantic relation with its child nodes. Assuming each architecture can be configured with selected m nodes from a total of n nodes, its configuration complexity will be n!/(m!(n-m)!) .
  $T - complexity(N_{Ci}) = n!/(m!(n-m)!)$ (4)

Following these formulas, engineers can easily implement an automatic solution to compute the test complexity for diverse

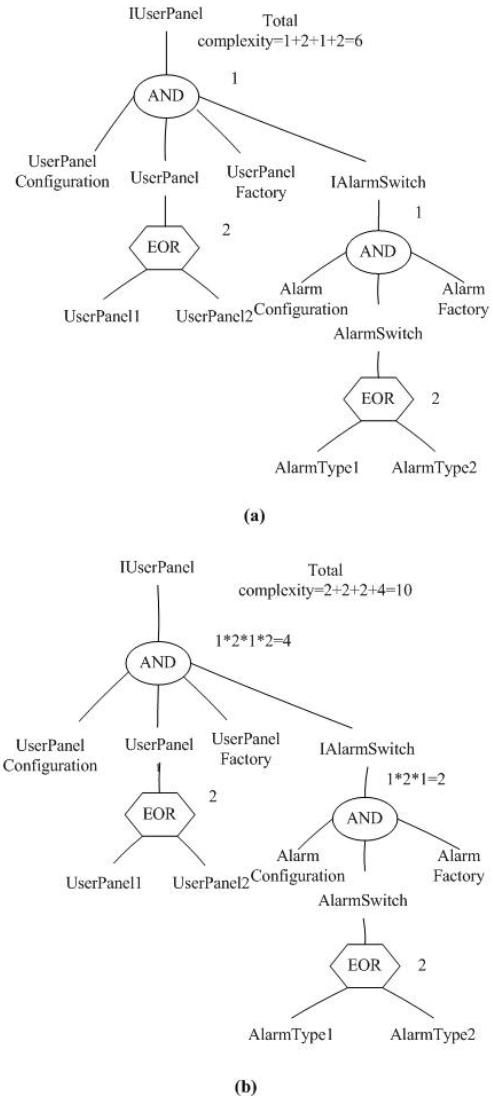


Figure 3. Complexity Computation Based on the Semantic Tree Model

configurable architectures in any given configurable component-based software. Figure 3 presents the detailed test complexity of different configurable architectures for component Elevator Controller (IUserPanel), which is a composite component in the elevator simulation system. Figure 3(a) presents the complexity of incremental approach, and Figure 3(b) shows the complexity of spanning tree approach from our previous work. In previous approach, the test complexity of parent node is affected by its child nodes. In other words, the complexity relation between parent node and its child nodes is a kind of propagating relation. For example, the *IAlarmSwitch* component with an AND relation to its three child nodes. Hence, its test complexity is 1 according to the incremental approach. However, in previous approach, the complexity of *IAlarmSwitch* is affected by its child nodes. Since the complexity of AlarmSwitch is 2, the complexity of its parent node *IAlarmSwitch* should be 2, as shown in Firure 3(b).

Figure 4. The Configuration Interface of The Elevator Simulation System

V. Case Study

We report our case study by applying the proposed incremental testing approach and complexity analysis into a configurable component-based elevator simulation system. We have used two software testing classes and three master project teams to conduct the related experiments in San Jose State University, California, USA. In this study, we primarily focus on the following items:

- Model configurable architecture using the proposed incremental approach based on the semantic tree model.
- Identify and analyze the test complexity of configurable architecture in component-based systems.
- Compare the approach with previous work in terms of complexity analysis.

The system is a well-designed component-based elevator simulation system. It consists of several components, which are *car*, *user panel*, *door*, *door panel*, *userpanel queue*, *car controller*, *floor panel*, *adminconsole*, and *metacontroller*. In previous work, we have adopted this system as our empirical testing or retesting object [6, 8, 11]. We have modified the original system and enhanced the functions in new versions. Here is the latest version. The existing elevator system has been updated with three additions: a)Internal Alarm in the user panel, b) The external alarm status on the floor panel, and c) New algorithm (Least recently used). Figure 4 shows the configuration interface for the elevator system.

A. Study Results and Discussion

Table II presents the detailed complexity of the semantic tree model of the changed components and the entire Elevator Simulation System. For example, the complexity value of component *user panel* is 6, which presents the number of different configured architectures for the component. For another changed component *algorithm*, the complexity value is 5. The total test complexity value for the system is 30, which presents the total number of different configured architectures for the elevator system. Hence,

Table II
THE SEMANTIC TREE AND ITS COMPLEXITY USING THE INCREMENTAL APPROACH

Semantic Tree for Configurable architectures	No. of Nodes	No. of Leaves	No. of Links	Max Height	No. of EOR	No. of AND	No. of Select-1	Testing Complexity
User Panel	13	8	12	3	2	2	0	6
Algorithm	8	6	7	2	0	1	1	5
System	65	44	64	5	6	14	1	30

while validating this software, a vendor's engineers must test its deployed instances to cover its configurable architectures. In practice, they can achieve the defined adequate test criteria in an incremental approach. For example, whenever a customer is deployed one instance, its configured system architectures (or component composition structure) will be recorded. For this system, we found that we need to develop 30 scripts to set up and cover different architectures so that the deployed system instance can be tested with certain adequate test set using the existing test methods.

B. *Study Comparison with Previous Work*

To investigate the effectiveness of our approach, we conducted a comparison study with the previous work. We utilize configuration complexity as the metrics for comparison. Figure 5 and 6 shows the corresponding complexity value in component *user panel* and *algorithm* respectively. We recorded the number of semantic relations, the max height, the links and nodes, the leaves, and the spanning tree in the semantic tree model and incremental semantic tree model. As shown in Figure 5 and 6, the semantic tree and incremental semantic tree have the same number of elements, like leaves and nodes. However, the complexity value of spanning tree approach is more than that of incremental approach. Figure 6 shows the comparison result in modified component *algorithm*. We also discover that the complexity value of spanning tree approach is more than the incremental approach semantic tree model. In addition, we found that at system level the previous spanning tree approach is much more complicated than the incremental approach, as shown in Figure 7. The complexity value is 327 and 30 respectively.

Through the comparison study of these two approaches, we find that in case of highly complex components like Elevator system as a whole, the complexity value is significantly greater in the spanning tree approach as against the incremental approach. We discover that we need to construct a huge semantic tree for configurable architecture in realistic component-based software. Therefore, the proposed incremental approach can less the complexity of configuration testing and achieve cost-effectiveness to some extend.

C. *Threats to Validity*

There are several potential threats to the empirical studies. We only consider the configurable architecture for testing. How to effectively conduct other aspects of configuration testing is still an open issue. In addition, we did not consider the GUI change and impact, external environment software and hardware changes. The component-based system in the case study is built for academic use, hence, the size of the system is relatively not large enough in complex industrial environment. The proposed measurement here is not the only possible measurement for testing complexity. Measurement factors such as human cognitive complexity, manual analysis complexity are not considered in this paper.

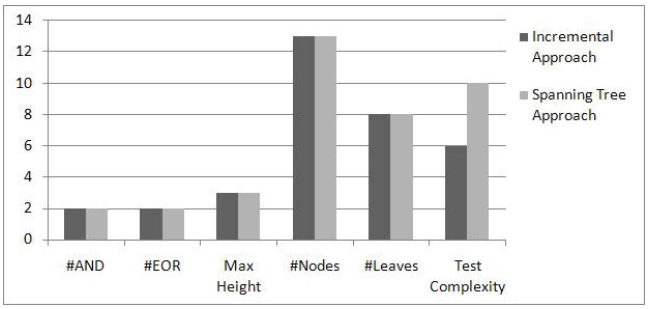


Figure 5. Complexity Comparison for User Panel

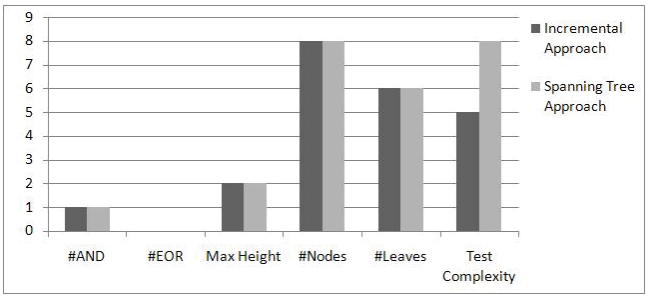


Figure 6. Complexity Comparison for Algorithm

VI. RELATED WORK

A number of papers addressed the testing or re-testing problems existed in component-based software [4, 5, 12–17].

Based on our recent literature survey, there are only a few of papers addressing testing issues and challenges in configurable components. The existing work can be classified into two groups:

- Testing configurable system constraints using combinatorial interaction testing (CIT) [3, 7] - CIT is a method to sample configurations of a software system systematically for testing.

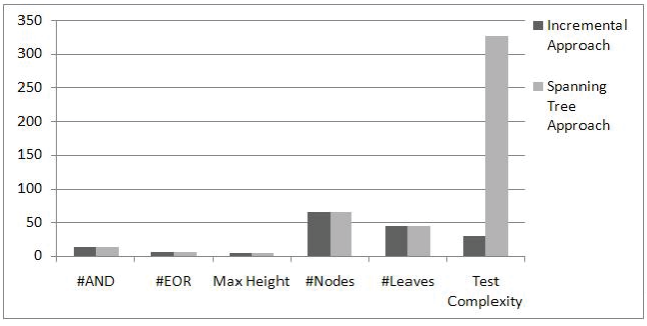


Figure 7. Complexity Comparison for The Entire System

Many algorithms have been developed to create CIT samples. A general constraint representation and the related solving technique are presented in [3]. It focuses on this problem by examining two highly configurable software systems to quantify the nature of constraints in real systems. CIT can provide an effective way to sample configurations for testing. Based on CIT, Qu et al. in [1] introduced an approach to configuration testing using prioritization, which aims at earlier detection of defects.

- Regression testing of system with configurable features - For instance, Robinson et al. proposed a firewall method for regression testing of user-configurable software [2, 18]. Those papers focused on user-centered tests for system configuration. They constructed a firewall to identify the impacted area in system based on setting changes and configurable element changes respectively, then created or selected test cases to cover the impacts. Some case studies are reported.

Our previous work [6] provided a semantic tree to address issues in test modeling, test adequacy, and test complexity analysis.

VII. Conclusions

Although there are numerous papers addressing how to construct configurable software and components, only a few papers discussed how to test configuration features and architectures of component-based software. This paper uses an incremental approach to discussing the relating issues, challenges, and test process. It applied a semantic tree model as a test model to present and analyze the diverse configurable architectures in component-based systems. In addition, the detailed complexity analysis and computation is presented to facilitate the study comparison. Furthermore, some case study results are reported demonstrate its effectiveness and application in test modeling and test complexity analysis. Compared with previous work, the new approach can reduce testing complexity effectively. Currently, we are developing a test automation solution to support automatic testing of configurable component-based software. The future extension of this research is to study how to use a model-based approach to addressing regression testing issues and challenges in configurable features and services in web services or cloud-based applications.

Acknowledgement

We thank the students of SJSU's CMPE 287 course who participated in our study, and the support of Computer Engineering Department in San Jose State University of California.

References

[1] X. Qu, M. B. Cohen, and G. Rothermel. Configuration-aware regression testing: an empirical study of sampling and prioritization. In *International Symposium on Software Testing and Analysis*, pages 75–86, 2008.

[2] B. Robinson and L. White. Testing of user-configurable software systems using firewalls. In *International Symposium on Software Reliability Engineering*, pages 177–186, 2008.

[3] M. B. Cohen, M. B. Dwyer, and J. F. Shi. Interaction testing of highly-configurable systems in the presence of constraints. In *International Symposium on Software Testing and Analysis*, pages 129–139, 2007.

[4] A. Orso, M. J. Harrold, D. Rosenblum, G. Rothermel, M. L. Soffa, and H. Do. Using component metacontents to support the regression testing of component-based software. In *IEEE International Conference on Software Maintenance*, pages 716–725, 2001.

[5] Y. Wu, D. P, and M. H. Chen. Techniques for testing component-based software. In *The 7th International Conference on Engineering of Complex Computer Systems*, pages 222–232, 2001.

[6] J. Gao, J. Guan, A. Ma, C. Q. Tao, X. Y. Bai, and D. C. Kung. Testing configurable component-based software-configuration test modeling and complexity analysis. In *International Conference on Software Engineering and Knowledge*, pages 495–502, 2011.

[7] 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 Transactions on Software Engineering*, 23(7):437–444, 1997.

[8] C. Q. Tao, B. X. Li, and J. Gao. A model-based approach to regression testing of component-based software. In *International Conference on Software Engineering and Knowledge Engineering*, pages 230–237, 2011.

[9] A. E. Hassan. Predicting faults using the complexity of code changes. In *International Conference on Software Engineering*, pages 78–88, 2009.

[10] A. P. Nikora and J. C. Munson. An approach to the measurement of software evolution. *Journal of Sosftware Maintenance And evolution: Research and Practice*, 17(1):65–91, 2005.

[11] J. Gao, K. Kwok, and T. Fitch. Model-based test complexity for software installation testing. In *International Conference on Software Engineering and Knowledge Engineering*, 2008.

[12] J. Gao, D. Gopinathan, Q. Mai, and J. S. He. A systematic regression testing method and tool for software components. In *Proceedings of the 30th Annual International Computer Software and Applications Conference*, pages 455–456, 2006.

[13] J. Zheng, B. Robinson, L. Williams, and K. Smiley. Applying regression test selection for cots-based applications. In *Proceedings of International Conference on Software Engineering*, pages 512–522, 2006.

[14] Y. Wu, D. Pan, and M. H. Chen. Techinques of maintaining evolving component-based software. In *IEEE International Conference on Software Maintenance*, 2000.

[15] C. Y. Mao. Regression testing for component-based software via built-in test design. In *ACM Symposium on Applied Computing*, 2007.

[16] J. Gao, R. Espinoza, and J. S. He. Testing coverage analysis for software component validation. In *The 29th Annual International Computer Software and Applications Conference*, pages 463–470, 2005.

[17] E. J. Weyuker. Testing component-based software: a cautionary tale. *IEEE Software*, 15(5):54–59, 1998.

[18] B. Robinson and L. White. On the testing of user-configurable software systems using firewalls. *Journal of Software Testing, Verification, and Reliability*, 22(1):3–31, 2010.

Using Architecture to Support the Collaborations in Software Maintenance

Yanchun Sun, Hui Song, Wenpin Jiao

Institute of Software, School of Electronics Engineering and Computer Science, Peking University
Key laboratory of High Confidence Software Technologies, Ministry of Education
Beijing 100871, P.R.China
E-mail: {sunyc, songhui06, jwp}@sei.pku.edu.cn

***Abstract*—Software maintenance is inherently a social activity because large systems typically involve the participation of many stakeholders throughout the software lifecycle. The collaborations in software maintenance not only need maintainers and different developers to achieve common understanding on a large number of shared artifacts but also require awareness support to ensure the maintainers and developers aware of the activities of others in order to coordinate their work, identify potential problems and prevent conflicts. This paper puts forward an approach using software architecture to support the collaborations in software maintenance across the whole lifecycle. In the approach, software architecture is used to organize various software artifacts in software development process from a high level perspective, and maintainers cooperate with one another or other developers and coordinate their maintenance activities based on their observations on the modifications taking place over the software architecture. In the end of this work, a case is studied in detail to illustrate how the approach works.**

Keywords-software maintenance; software architecture; collaborative software development

I. INTRODUCTION

Software maintenance is inherently a social activity, large systems typically involve the participation of many stakeholders throughout the software lifecycle [1]. Collaborations in software maintenance of large and complex software systems not only need maintainers and different developers to achieve common understanding on a large number of shared artifacts but also require awareness support to ensure the maintainers and developers aware of the activities of others in order to coordinate their work, identify potential problems and prevent conflict. But until now, there has been limited research on how methods and tools support collaborative software maintenance [2]. How to support both shared understanding software artifacts based collaboration and activity awareness among maintainers and different developers efficiently within a large software maintenance process becomes a big challenge. This is the main concern of this paper.

For collaboration based on software artifacts, the version control systems are frequently used to manage the software artifacts, such as CVS (Concurrent Versions System), SVN(Subversion), etc. [3, 4] These tools are very important for collaborative development among software engineers, because people could easily view what other people change on the source code. However, the artifacts managed by these tools are all in the code level, and lack a reasonable organization from the global viewpoint of software maintenance process.

For activity awareness support, one way is to use software process models [5,4], which attempt to help teams deal with the complexity of activities. But this requires that the collaborative maintainers and developers reach a consensus on the global view (i.e. the software process model). Moreover, the awareness support based on software process models, more or the less, is a kind of centralized control on the whole process, and lacks of the support for distribution and coordination during the collaborative software maintenance. Another way to provide awareness is to build it into a common repository, such as CVS, which monitors the changes made to artifacts and uses the data as the basis for awareness provision. But this method has the similar drawbacks as the software artifacts based collaboration aforementioned.

To support software artifacts based collaboration and activity awareness among maintainers and developers together in the whole software lifecycle, it is necessary to provide software maintainers with an appropriate model to organize various software artifacts in software maintenance process from a high level perspective in a distributed way. At the same time, models can provide awareness support to ensure different maintainers and developers aware of the activities of others so that they can coordinate their work and collaboratively maintain software. In order to utilize the models from different perspectives on the same system, a key challenge is how to synchronize the different models, and make sure the changes on one model could be propagated correctly to the others. To achieve this, a coordination model that captures the key information from all the different views plays an important role.

Software architecture is widely used as such a coordinator model. With software becoming large and complex, software architecture (SA) becomes a blueprint to guide the development and maintenance of software systems. The inherent high-level abstract, coarse granularity and comprehensibility of SA make it an ideal base for collaborative development and maintenance. Now, some SA-centered development methods have been put forward [7,8,9,10,11,12],

but they do not use the semantics of SA adequately, while just support simple collaboration for several authors based on the management of authority. Moreover, they support collaborative development just in special phases rather than in the whole lifecycle.

In this paper, we put forward an approach using architecture to support the collaborations in software maintenance, which extends our previous approach [13, 23] from supporting collaborative design to collaborative maintenance in the whole lifecycle. First, based on version control tool and semantic information of SA, we abstract the information of fine-grained modifications into SA in order to support the collaborative design of software architecture among designers in the design phase of software systems. Then we enlarge this collaboration mechanism from collaborative design to collaborative maintenance during the whole lifecycle because SA is a core artifact in the whole software lifecycle. By introducing bi-transformation technologies [14], we can transform the modification manipulation of other artifacts to the modification manipulation of the central SA model, so as to support the collaborative maintenance among maintainers and different developers. This approach supports the distribution and coordination of the maintainers in the whole software lifecycle through several part views when collaborative software maintenance has no global view.

The rest of this paper is organized as follows. Section 2 presents some related work. In Section 3, we put forward an architecture-centric approach to supporting collaborative software maintenance. Section 4 illustrates the approach by studying one case in detail. Section 5 presents concluding remarks and future work.

II. RELATED WORK

Until now, there has been limited research on how methods and tools support collaborative software maintenance.

Collaborative software maintenance asks maintainers and different developers to achieve common understanding on multiple software artifacts such as requirements specifications, architecture description, design models, testing plans, and so on. For the collaboration based on software artifacts, the version control systems (CVS, SVN, etc.) are often used to manage the software artifacts [4,15]. But the artifacts stored in these tools has their own semantics, ranging from nature language to semi-formal semantics of UML, or formal semantics of a program language, they lack a reasonable organization from the global viewpoint of software maintenance process. As a result, software artifacts based collaboration almost concentrates on special phases and is short of support for the software maintenance goals throughout the whole software lifecycle.

How to assist developers in knowing each other's work status (i.e., awareness) is a key issue in collaborative development [16]. Some researches show that configuration management technologies can provide developers with a view of each other's activities [15,17,18]. They achieve this goal by obtaining the low-level information of modification and displayed in a direct way. Compared with showing different detailed information from several views of these work, we organize the detailed information of modification in multi-views of software architecture for the whole collaborative maintenance lifecycle. Another difference is that this visualized view in our approach is the threshold of collaborative maintenance and not just a static visualized display.

Even with a complex configuration management system available to them, developers still conduct a good deal of communication and monitoring in order to maintain a broad collective understanding of team activities [19]. Formal software process models can provide activities awareness for the developers [6], but this requires that the collaborative developers have general knowledge on the collaborative development process and reach a consensus on the global view (i.e. the software process model). Our approach focuses on non-general knowledge on the collaborative maintenance process, and tries to support distribution and coordination through multi-views of software architecture for the whole software lifecycle.

Software architectures are considered more and more important because they blueprint the target products and determine the system-wide qualities. Many SA-based technologies have been developed to support collaborative work. In the academy, Richard Taylor and David Garland present their own Architecture Description language (ADL) and propose the SA-centered development method based on the ADL [7,8]; ArchStudio from UCI [10] and ACMEStudio from CMU [20] typically support collaborative authoring by versioning architecture description files. MolhadoArch system from University of Wisconsin is integrated with a fine-grained version control tool to afford the collaboration at the level of individual model elements [11]. In the industry, Siemens's Hofmeister et al. describe a set of architecture views and put forward a corresponding software development method from requirement to implementation [9]. Researchers in IBM also focus on SA-centered development method and "Rational Software Architect" is an UML modeling tool focused on software architecture [12]. Developers work collaboratively on diagrams with collaboration mediated via the configuration management system.

Compared with our approach using architecture supporting the collaborations in software maintenance for the whole software lifecycle, most of the collaborative supports provided by these tools above are fine-grained. Moreover, they are just limited in the special phases instead of the entire software lifecycle.

III. APPROACH OVERVIEW

Figure 1 illustrates an approach using architecture to support the collaborations in software maintenance. Our approach is based on a software reuse methodology called as ABC (Architecture Based Component Composition) [21]. ABC method regards that SA should play a centric role in the whole software lifecycle, that is, SA description is used as the blueprint and middleware technology as the runtime scaffold for component composition, maintenance and evolution. We define a model as the abstraction of the software architectural information, and maintain this model throughout the whole software maintenance process. The specific artifacts can be regarded as the different views of this central architecture

model and different developers' manipulating the specific artifacts can be considered as operating the model from their own views. Developers in different roles operate on their own views, and our maintenance environment will synchronize the specific views with the unique architecture model. In this way, maintainers could cooperate with the other developers upon a consistent basis, but through the specific artifacts with which they are more familiar.

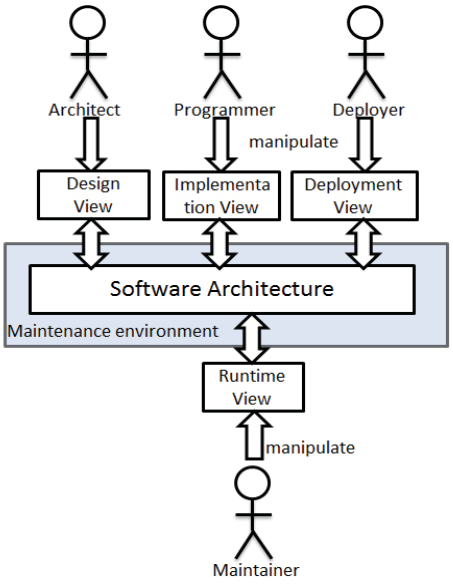


Figure 1. Approach overview

Specifically, the architects design the original version of the software architecture. Programmers implement the system according to the architecture, based on the decided platform and techniques. After that, deployers install the system and configure it for good performance. In the post-development phase, the maintainers may reconfigure the system when needed. Since the maintainers' change usually focus on a part of the whole system, they also doubt if the changed system violate some designed properties, or violate architects' or programmers' intention. And thus, an efficient communication mechanism between them is needed. Considering the different perspective between developers and maintainers, we introduce a system model as a "runtime view" for the maintainers, and maintain a causal connection between runtime view and the software architecture, based on model synchronization. The maintenance environment produces the runtime view from software architecture to help maintainers understand the original design and development decisions. After the maintainers make changes, the maintenance environment transforms the runtime view back to the software architecture to ask for the opinions from architects or developers, and accept their further reconfiguration.

A. Software Architecture Model

By referencing the typical architecture description language (ADL) [22], we define the meta-model of software architecture by using Eclipse Ecore. Figure 2 describes the core elements in the meta-model. The core concept is Component, which is also a basic block with a specific concern of the system. We introduce the concept of InnerStructure, to organize the whole system as a hierarchical structure for supporting the distributed maintenance, i.e., maintainers with different concerns could work together inside a specific component. Based on the meta-model, we construct a software architecture modeling environment by using Eclipse GMF, named as ABCTool, in order that we could assist designers to record their design decisions by recording their manipulations such as additions and deletions of elements and modifications of properties and relationships of elements. The entire architecture model is recorded in the form of XMI in several files.

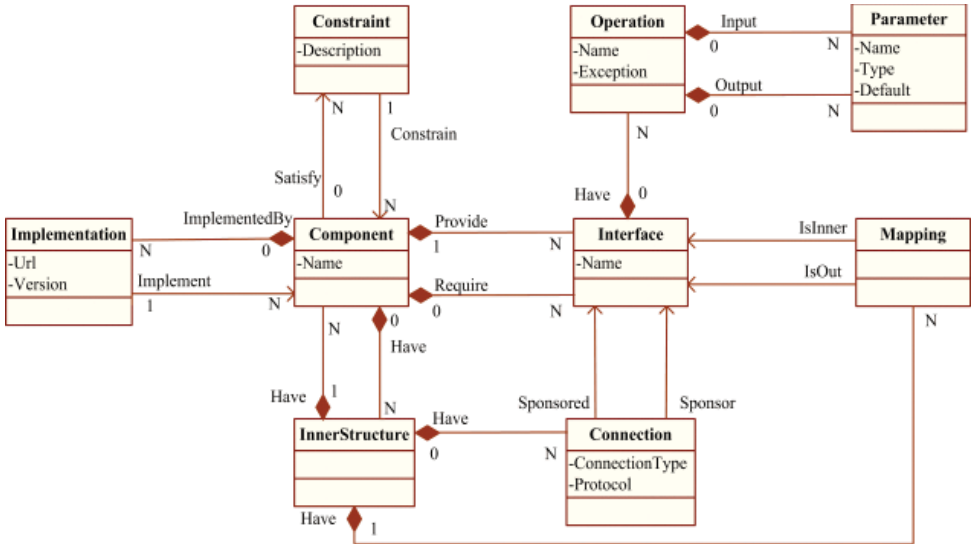


Figure 2. Software Architecture Model

B. Architecture recovery for maintenance

The some of the architecture views in Figure 1 are constructed from raw artifacts, via different architecture recovery techniques, including code analysis and runtime synchronization.

For implementation view, if there is not an existing link between the architecture model and the source code, we employ the code analysis to recover such an architecture model that describes the code structure. The recovery starts from identifying the key elements in the code hierarchy, such as classes, methods, fields, etc. In order to construct the association between these elements, we utilize simple static code analysis techniques, such as data flow tracing, to infer the invocation relation.

For deployment and runtime view, we utilize our previous work on runtime model construction to recover and maintain a runtime architecture model [24]. Such a runtime architecture view describes the elements that constitute the running system, such as the objects of classes. Each element contains a set of states, which keep changing during runtime. Our architecture construction approach maintains a causal connection between the model and the system, so that the system changes will cause the corresponding model changes, and the modification on models will trigger the system change immediately.

C. Low-level implementation of modification management

One core function of our maintenance environment is the management of modification. We implement the low-level version control based CVS. Since the architecture models are stored as XMI files, which are pure text files, CVS could provide enough fine-grained version control capabilities.

We integrate the Eclipse CVS plug-ins inside our maintenance environment. During the maintenance process, we record the modifications, including the original and modified contents and the information about by whom and on which file the modifications are made and the short rationales of the modifications. Thanks to the well implementation of the Eclipse CVS, we can get such information through convenient APIs, and represent on architecture levels, which will be discussed in the next section.

D. High-level representation of modification management

Our maintenance environment retrieves the low level information about modifications from the CVS API, and represents such text-level modifications in the architecture level. It also provides high-level management operations along with the architecture model.

When a model is changed, maintainers can select to accept, reject or add new modifications. The maintenance activities for modifications can be mapped to the operations in the CVS. During the maintenance, different stakeholders can use the modification information retrieved from the CVS to identify the intentions of modifications. When necessary, they also need to contact the developers who made the modifications to discuss their goals. We provide a support mechanism for peer-to-peer communication in ABCTool. Different developers will deliver different artifacts, but most of these artifacts record some core information of SA. In other words, some transformation relationships exist between these artifacts and SA model. Thus, through transforming the core information in SA and adding special information in a given phase, the artifacts in the given phase can be constructed. Using those research fruits in the bi-transformation field [14], we can use a set of transformation rules to reflect the modifications of SA model into other models, and also reflect the modifications of SA level information in other models to SA model. Thus, we can utilize the approach above to assist with the collaborations among a variety of developers participating in software maintenance.

IV. CASE STUDY

We use a simple news subscribing system as a sample, to show how maintainers and different developers maintain the system collaboratively with the help of software architecture.

The architecture of this sample system is shown as Figure 3. The users logging on to the system through a user interface component, which calculates the user's authority through the Logon component, with the help of user information retrieved from the UserInfor component. If the logging on succeeded, the user could use the UI component to browse the news provided by the NewsRetailer component. This architecture model is designed by the architects (as the design view) and directly used as the central software architecture. According to the above design architecture, programmers compose the system as Figure 4, upon the JEE techniques. Currently, all the components are implemented as EJBs. This implementation implies a simple transformation rule from design to implementation: A component is implemented as an EJB, and the connectors are implemented as EJB references. As a simple example, the running system complies with the implementation, and thus Figure 4 actually acts as both the implementation and the runtime view.

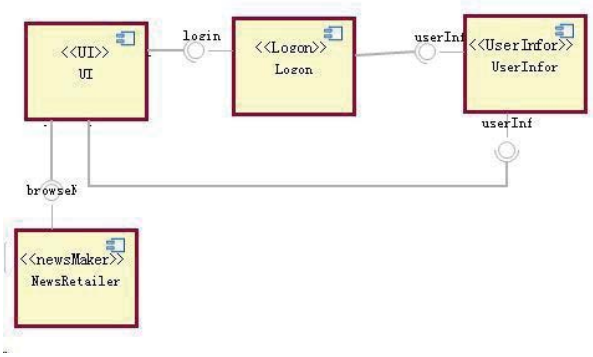


Figure 3. Original Design View

After running a while, the maintainers find that the userInfor component (emphasized in Figure 4) is a bottle-net of the whole system, because every time, the logon component retrieves all the user information. But actually, for logging-on, we only need the user id and the password. To use the user information more efficiently, they split the user information retrieving logic into two EJBs, a simple one for logging-on, and a detailed one for showing the user profile. In the meantime, they also refactor the newsMaker component, reusing the existing XML generator. After the evolution, the

implementation is changed into Figure 5. To make the changes known to the architects, they transform back the runtime view into the architecture model (design view) as shown in Figure 6. Since the change on userInfor does not introduce more functions in the architecture level, the transformed architecture still contains only one UserInfor component, but one of its interfaces is also linked to NewsRetailer. This scenario shows how the maintainers propagate their system level evolution to the architects.

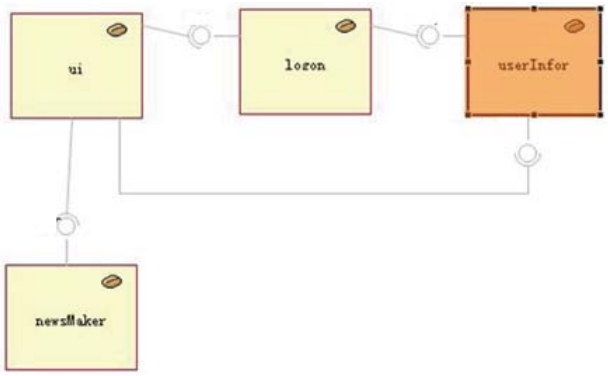


Figure 4. Original Runtime View

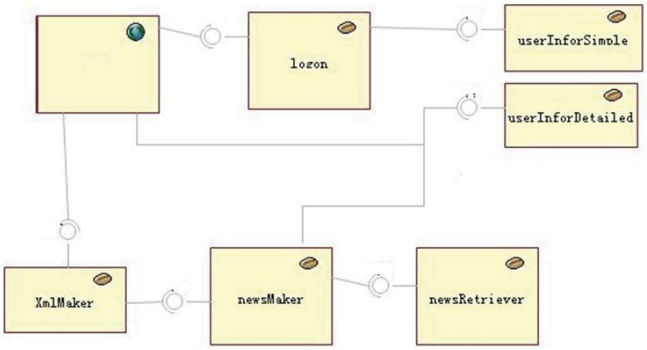


Figure 5. Modified runtime view

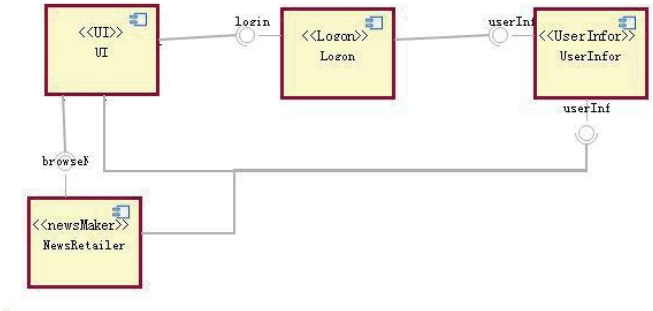


Figure 6. Modified Design View

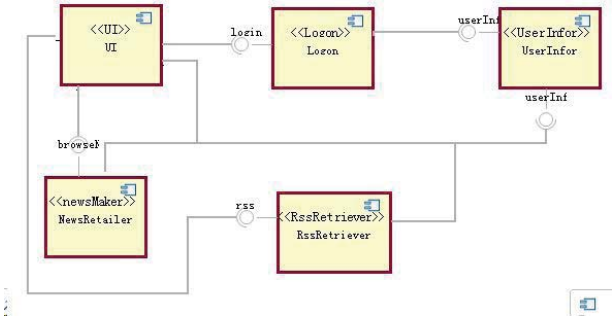


Figure 7. Final Design View

Obtaining the new architecture, the architects need to commit it and do more changes when needed. They first notice that the NewsRetailer actually does two kinds of work: producing the news and revealing the user information. This is not a good design. So they decide to introduce an independent component for revealing the user profile, as shown in Figure 7. This architecture is transformed into a new implementation view as shown in Figure 8, with a new EJB for formatting the user information as RSS seeds. The programmer implements the new EJB according to the implementation view, and deployers deploy the system again. Finally, the maintainers get back a new system as shown in Figure 8.

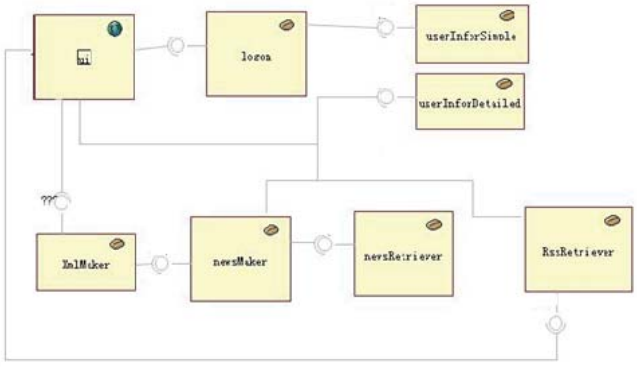


Figure 8. Final Runtime View

V. CONCLUSIONS AND FUTURE WORK

In this paper, we present an approach using architecture to support the collaborations in software maintenance. The key information throughout the whole development process is abstracted into a unique architecture model. Developers in different roles operate on their own views, and our maintenance environment synchronizes the specific views with the unique architecture model. In this way, maintainers could cooperate with the other developers upon a consistent basis, but through the specific artifacts with which they are more familiar.

Our work is still an initial attempt. Currently, synchronizing runtime view and central architecture model needs to be fastened. We are considering adding more assistant around the central architecture model and runtime view, e.g. assist maintainers in recording maintenance process and the rationale

behind maintenance activities, and reuse them to improve the efficiency of software maintenance.

ACKNOWLEDGMENT

This effort is sponsored by the National Basic Research Program of China (973) under Grant No. 2009CB320703, the Joint Fund of the National Natural Science Foundation of China under Grant No. U1201252, the National Natural Science Foundation of China under Grant No. 61073020, and the Science Fund for Creative Research Groups of China under Grant No. 60821003.

REFERENCES

[1] J. Sillito and E. Wynn. "The social context of software maintenance", in Proceedings of IEEE International Conference on Software Maintenance (ICSM07), Oct. 2007, pp. 325–334.

[2] Storey, M.-A., Bennett, C., Bull, R.I., German, D.M., "Remixing visualization to support collaboration in software maintenance", Frontiers of Software Maintenance 2008(FoSM 2008), pp.139 - 148.

[3] J Froehlich, P Dourish, "Unifying Artifacts and Activities in a Visual Tool for Distributed Software Development Teams", in Proceedings of the 26th International Conference on Software Engineering (ICSE 2004), May 23-28, EICC, Scotland, UK,2004.

[4] Grady Booth, IBM Rational, "Introducing Collaborative Development Environments", Dec 2006, http://www.alphaworks.ibm.com/contentnr/ cdepaper.

[5] Boehm, B. and Bose, P. A Collaborative Spiral Software process Model based on Theory W, Proceedings of 3rd International Conference on the Software Process (Reston, VA), IEEE, New York, 1994, 59-68.

[6] Sutton, S. and Osterweil, L. The Design of a Next Generation Process Language. Proc. Sixth European Software Engineering Conf. (Zurich, Switzerland), Springer, 142-158, 1997.

[7] Nenad Medvidovic, David S. Rosenblum, Richard N. Taylor, "A language and environment for architecture-based software development and evolution", in Proceedings of the 21st international conference on Software engineering, Los Angeles, California, United States, 1999.

[8] David Garlan, Shang-Wen Cheng, An-Cheng Huang, Bradley Schmerl, Peter Steenkiste, "Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure" , Computer, vol. 37, no. 10, pp. 46-54, Oct., 2004.

[9] C Hofmeister, R Nord, D Soni, Applied Software Architecture, Addison Wesley, 2000.

[10] UCI Software Architecture Development Environment, 2007, http://www.isr.uci.edu/projects/archstudio.

[11] T.N.Nguyen and E.V.Munson, Object-oriented Configuration Management Technology can Improve Software Architectural Traceability", in 3rd ACIS International Conference on Software Engineering Research, Management and Applications(SERA'05), Mount Pleasant, Michigan, USA, 2005, pp.86-93.

[12] IBM, "Rational Software Architect Overview," 2007, http://www-306.ibm.com/software/awdtools/architect/ swarchitect/.

[13] Yanchun Sun, Hui Song, Xinghua Wang, Wenpin Jiao. Towards Collaborative Development Based on Software Architecture. In the 20th International Conference on Software Engineering and Knowledge Engineering (SEKE'2008), Redwood City, CA, USA. July 1 - July 3, 2008, 250-254.

[14] Yingfei Xiong, Dongxi Liu, Zhenjiang Hu, Haiyan Zhao, Masato Takeichi, Hong Mei, "Towards Automatic Model Synchronization from Model Transformations", in Proceedings of 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2007), Atlanta, Georgia, November 5-9, 2007.

[15] Grinter, R. Using a Configuration Management Tool to Coordinate Software Development. Proc. Conf. Organizational Computing Systems COOCS'95 (Milpetas, CA), ACM, New York, 1995, pp.168-177.

[16] C. Gutwin, R. Penner, and K. Schneider, "Group Awareness in Distributed Software Development," in Computer Supported Cooperative Work Chicago, Illinois, USA.: ACM Press, 2004, pp. 72-81.

[17] Finkelstein, A., Kramer, J., Nuseibeh, B. Software Process Modelling and Technology. RSP Ltd, 1994.

[18] Sarma, A., Noroozi, Z., and van der Hoek, A. Palantír: Raising Awareness among Configuration Management Workspaces. Proc. Intl. Conf Software Engineering, ICSE 2003 (Portland, OR, May), pp.444-454.

[19] de Souza, C., Redmiles, D., and Dourish, P., Breaking the Code: Moving between Private and Public Work in Collaborative Software Development. Proc. Conf. Supporting Group Work GROUP 2003, ACM, New York, 2003.

[20] Kompanek, "Modeling a System with Acme", 1998, http://www.cs.cmu.edu/~acme/html/WORKING-%20Modeling%20a%20System%20with%20Acme.html.

[21] Hong Mei, "ABC: Supporting Software Architectures in the Whole Lifecycle", In: Proceedings of the Second International Conference on Software Engineering and Formal Methods (SEFM'04), 28-30 September 2004, Beijing, China, IEEE Computer Society 2004.

[22] Medvidovic and R. N. Taylor, "A Classification and Comparison Framework for Software Architecture Description Languages", IEEE Transactions on Software Engineering, pp. 70-93, 2000.

[23] Yanchun Sun, Hui Song, Wenpin Jiao Towards Architecture-centric Collaborative Software Development. In the 21th International Conference on Software Engineering and Knowledge Engineering (SEKE 2009), Bosten, USA. July 1 - July 3, 2009.

[24] Hui Song, Yingfei Xiong, Franck Chauvel, Gang Huang, Zhenjiang Hu, Hong Mei: Generating Synchronization Engines between Running Systems and Their Model-Based Views. In ACM/IEEE 12th International Conference on Model Driven Engineering Languages and Systems (MoDELS'09), 2009.

Reverse Engineering of Sequence Diagrams by Merging Call Trees

Seonghye Yoon
Dept. of Comp. Sci. and Eng.
Sogang University
Seoul, South Korea
seonghye@sogang.ac.kr

Sunghyun Min
Software R&D Center
Samsung Electronics Co., Ltd.
Suwon, Korea
shmin02@gmail.com

Sooyong Park
Dept. of Comp. Sci. and Eng.
Sogang University
Seoul, South Korea
sypark@sogang.ac.kr

Soojin Park
Graduate School of
Management of Technology
Sogang University
Seoul, South Korea
psjdream@sogang.ac.kr

***Abstract*—A software system is continuously changing even after development. Thus, most of documents which are written at the end of development phase cannot reflect the current software system. It is a very tedious work for maintainers to understand how the software is implemented by reviewing source code. So, several reverse engineering techniques and tools have been introduced to generate sequence diagrams from source code or behaviors of the software at runtime. However, the legacy techniques have limits not to provide sufficient coverage and readability with the generated sequence diagrams. We propose a method to merge call trees from static and dynamic analysis results to enhance the coverage of reversed sequence diagrams. This paper introduces five rules for merging call trees. The evaluation result explains how our rules effectively generate compact and understandable sequence diagrams without any loss of control flows which are implemented in source code.**

Keywords – reverse engineering, sequence diagram, static analysis, dynamic analysis

I. INTRODUCTION

Software is continuously changing during development phase and even after each release. It is very hard to guarantee that change to the software is reflected by a corresponding update to the document. As time goes on, the gap between software itself and the document becomes wider and wider. Thus, it requires lots of time for maintainers to understand the current state of the software when a change on the software is requested for any reason. According to the survey of [1], around 30% of total maintenance cost is spent on just understanding legacy software system. A method to help maintainers to grasp current software without tedious reviews on source code should be provided to reduce whole cost of software maintenance.

In this background, there has been some work on addressing techniques to extract behaviors of software from source code to sequence diagrams [2][3][4], named as a static analysis based reverse engineering. It is useful to understand all alternatives of software behavior. In object-oriented programs, however, when the concept of dynamic binding or polymorphism is implemented in the source code, it is difficult to understand a whole flow of system behavior using pure static analysis result. Thus, much recent researches focus on generating sequence diagram from the execution traces [5][6][7][8]. It is generally called as a dynamic analysis based reverse engineering method. While the methods can solve the slicing problem due to the late binding or polymorphism, there exists another inherent problem. Execution trace can be captured only if a system behavior is executed in runtime. In other words, unexecuted flow cannot be captured in execution traces. Thus, it can cause the coverage issue on reverse engineering besides several well-known problems of dynamic analysis such as the execution trace explosion problem or infinitely repeated messages appearing in sequence diagrams. To overcome those limits on static or dynamic analysis, a few literatures [9][10] propose hybrid methods utilizing both static and dynamic analysis. However, they do not address the key way to enlarge coverage of reverse engineering because they just focus on generating more compact sequence diagrams by reducing loops or abstracting messages.

To address the above issues, we present a reverse engineering method utilizing information which is captured from source code and execution traces. At first, we generate a static call tree from source code and a dynamic tree from execution trace logs. The two call trees have the root node representing the same starting method of a flow. Then, an integrated call tree is incrementally completed with continuously comparing each node in the static call tree and the dynamic tree. According to the comparison result between two nodes in the static and dynamic call tree, a proper merging rule is selected and applied to add a new node in the integrated call tree. In this paper, we introduce five rules for merging call trees. To demonstrate the application of the rules, we make use of REQ Modeler [11], a UML modeling tool. We evaluate the effectiveness of the method by quantifying merging rules contributions to the generated sequence diagrams, which shows promising results.

The rest of the paper is organized as follows: Section 2 describes an overview of related works on reverse engineering. Section 3 presents the method for generating sequence diagram including call tree merging rules. Section 4 demonstrates a case study using a UML modeling tool and Section 5 discusses the results of evaluation. Section 6 concludes the paper with future work.

II. Related Work

According to literatures dealing with reverse engineering issues, the proposed approaches can be classified by three categories: static analysis, dynamic analysis and hybrid analysis.

At first, [2], [3], [4] introduce static analysis based reverse engineering methods. [2] proposed a technique for generating sequence diagrams. The technique captures all of control flows and loops from source code. [3] suggested a control flow analysis technique. The research invents a UML extension to capture general flows of control and makes an algorithm for mapping a reducible exception-free intraprocedural control-flow graph to UML. [4] introduced a technique for extracting a structural hierarchy of GUI, attributes of the widgets and their values. The extracted information helps maintainers to understand the structure of GUI programs. However, these techniques cannot prevent that a single flow of event is split into several meaningless sequence diagrams due to the implementation of several mechanisms of object oriented programming like polymorphism.

On the other hand, there are several reverse engineering techniques [5][6][7][8] using dynamic analysis information from execution traces which are logged at runtime. [5] introduced a technique which defines two meta models for reflecting execution traces and sequence diagrams, then it defined mapping rules between the two models using the Object Constraint Language (OCL) [12]. The captured execution traces are transformed into sequence diagrams through the mapping rules. [6] proposed a mining technique for identifying groups of core functions that implement software features. The result of the mining technique can be used for program comprehension. [7] suggested a method to reverse sequence diagrams using rules to compact repetitive parts of the execution traces. [8] proposed a formal method to focus on mapping between externally visible behaviors of a system and relevant parts of source code. Those dynamic analyses based reverse engineering methods succeeded in solving the problem of meaningless slicing of a diagram, which is the limit of static analysis methods. However, only executed operations during run time could be logged as an execution trace and transformed as messages in a sequence diagram through the dynamic analysis methods. There exists a coverage issue in capturing execution traces. In other words, the completeness of generated sequence diagrams cannot be guaranteed. The other inherent limit in dynamic analysis based methods is that the messages in a sequence diagram can explode due to repeated messages according to loops. [7] has tried to compress the repetitive parts of execution traces by utilizing predefined repetitive patterns. Nevertheless, there still exists a limit on removing all kinds of the repetitive parts by the pattern matching.

To overcome the limitations of static and dynamic analysis methods for reverse engineering, there have been a few literatures proposing hybrid techniques[9][10]. [9] proposed an algorithm which combines data from source code, execution traces, and debug information in order to reduce the number of messages of a sequence diagram. And [10] introduced a method that abstracts the history of object interactions using Pree's meta patterns[13]. By grouping strongly correlated objects, the sequence diagram can be simplified to focus on representing intergroup interactions. The two techniques based on the hybrid analysis focus on compacting enormous amount of information, which causes scalability issues. On dealing with the exploded number of messages in a sequence diagram, they utilize the information from the results of static analysis. It is a different point in comparison with existing dynamic analysis based methods. However, the coverage issue on unexecuted traces is not discussed in the previous work.

III. Reverse Engineering Method For Generating Sequence Diagrams From Execution Traces And Source Code

We propose a reverse engineering method for generation of sequence diagrams from implementation model (static code analysis results) and execution model (execution trace logs), which is composed of the following five steps; (A) limiting classes, (B) generating call trees, (C) filtering call trees, (D) merging call trees and (E) generating sequence diagram as shown in Figure 1.

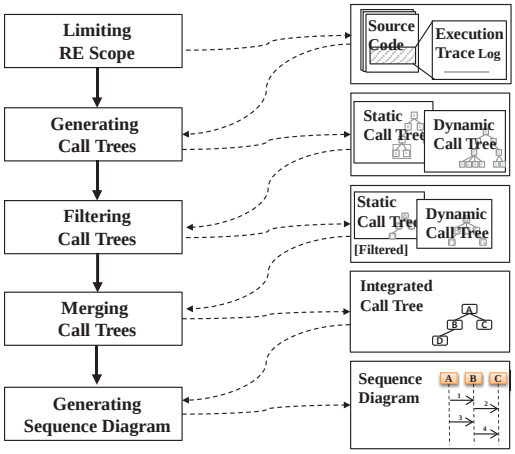


Figure 1. Overview of Reverse Engineering Method

A. Limiting the Scope of Reverse Engineering

Before starting the generation of sequence diagram from source code, we can designate the packages which are excluded from reverse engineering scope. In general, external library packages and the utility packages are excluded. Besides them, if a maintainer decides that interactions with a specific package are not necessary in understanding system behavior, the package can be excluded from the scope of reverse engineering.

B. Generating Call Trees

After limiting the scope of reverse engineering, a static call tree is generated from source code that is corresponding to the selected scenario by maintainer. The static call tree which we work on is based on the Abstract Syntax Tree (AST) [14] that is generated by the Eclipse [15] AST parser. Then, execution traces are captured from the sequence of calls in the executable files which are compiled from the designated source code. The execution traces are sources of a dynamic tree. To get a dynamic call tree, a dynamic tracing tool, BTrace [16] is used in this work.

Figure 2 shows an instance of the suggested static call tree. Each node is represented by a rectangle. A.a is a node for representing method *a( )* which belongs to class A. The AST does not explain which class is the owner of a method. Thus, we extract the ownership information from the tree structure of Eclipse Project Explorer and annotate it on each node. The relationship between classes and the signature of a method is contained as a property in each node. Besides following the general call tree representation rules, two additional ways is invented for representing repetition of methods and reducing duplicated method calls. A repetition of method(s) is outlined with a rectangle and annotated with a recursive arrow, named as a "repeated method block". If a method node is appeared more than twice in a call tree, the duplicated child nodes is omitted and an asterisk (*) is denoted on the node from the second appearance of the node. In Figure 2, the node C.c and the node F.f are repetitive methods. On the other hand, the node B.b is appeared twice in the call tree. Thus, the child nodes of the second appearance of node B.b (the node D.d and E.e) are omitted in the tree to make more compact sequence diagram.

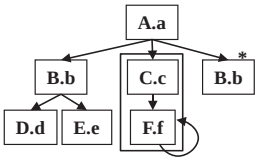


Figure 2. An Example of Static Call Tree

C. Filtering Call Trees

The initially generated static/dynamic call tree still includes trivial method nodes such as getter or setter. In the previous *Limiting the Scope of Reverse Engineering* step, packages which do not give valuable information to understand system behaviors have been already excluded. In this step, the kind of methods that should not be participated in a sequence diagram can be selectively designated.

D. Merging Call Trees

Through the previous steps, the filtered static call tree and the filtered dynamic call tree which have the same root node are obtained. In this step, the two call trees are merged into an integrated call tree according to five different rules. The presenting rules are designed to utilize benefits from both static analysis and dynamic analysis. Thus, the method nodes in both trees are selectively added to the integrated call tree according to given conditions. Basically, two method nodes located in the current order of both call trees are reviewed in sequence. A preorder depth-first traversal algorithm is used to visit each method nodes. To describe the rules more simply, several abbreviations are used as described below:

T_D: a dynamic call tree
T_S: a static call tree
T_C: an integrated call tree
M(i, d): the ith method node of dth depth

The five merging call tree rules are explained in the following.

R1.Identification of Common Method Node

If the current method node M(i,d) in T_D and the current method node M(j,d) in T_S are same methods, add M(i,d) to the T_C.

R2. Deletion of Repetitions in Message Call Sequences

1. If the current method node M(j,d) in T_S is a member of a repeated method block, search for the same repeated sequence of method nodes in dth depth of T_D.
2. If there exists an equivalent repeated sequence of method nodes, the whole repeated method block defined in T_S is added to T_C.

As shown in Figure 3, there exists a repeated method block (C.c and D.d) in T_S and the same repeated sequence of methods (C.c→D.d→C.c→D.d) is appeared in T_D According to the rule R2, T_C has the repeated method block. As shown in Figure 3(c), the repeated method block will be represented as a loop frame in the generated sequence diagram. Consequently, the application of R2 can help to make more compact sequence diagram.

R3. Identification of a Real Participating Object

1. If the current method node M(i,d) in T_D and the current method node M(j,d) in T_S have same signature but the owner classes of the two nodes are different ones, search about the relationship of the two nodes.
2. If the owner of M(i,d) in T_D is a child class of the owner of M(j,d) in T_S, add M(i,d) in T_D to T_C.

As depicted in Figure 4, the method *d( )* in the node D.d of T_S and the node E.d of T_D is identical. And, the class E is a child of the class D. Thus, the node E.d of T_D is added according to the rule R3. Consequently, the callee of the message *d( )* is the class E instead of the class D in the generated sequence diagram. Compared with the sequence diagram from T_D, the sequence diagram from T_C is more clear to understand system behavior by designating the real participating class in execution time.

R4. Identification of an Unexecuted Method

1. Compare with the current method node M(j,d) in T_S and the current method node M(i,d) in T_D.
2. If M(i,d) is null, the node M(j,d) in T_S is added to T_C.
3. If M(i,d) is not null and it is different with the node M(j,d), compare with the M(j,d) in T_S and the next neighbor node M(i+1,d) in T_D again.
4. If the node M(i+1,d) is also different with the node M(j,d) or null, the node M(j,d) is added to T_C.

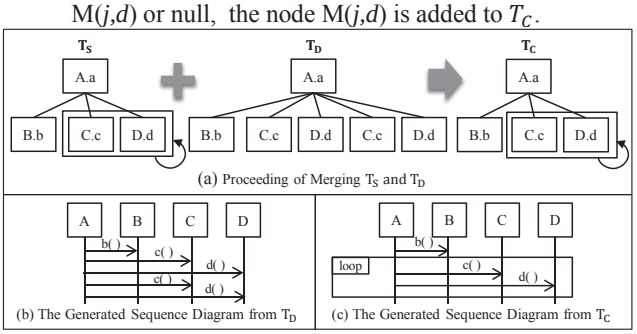


Figure 3. Applying *Deletion of Repetitions in Message Call Sequences* Rule

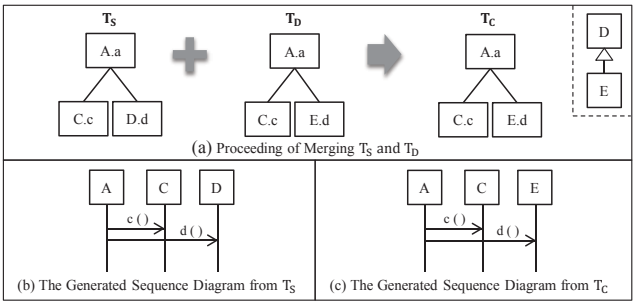


Figure 4. Applying *Identification of a Real Participating Object* Rule

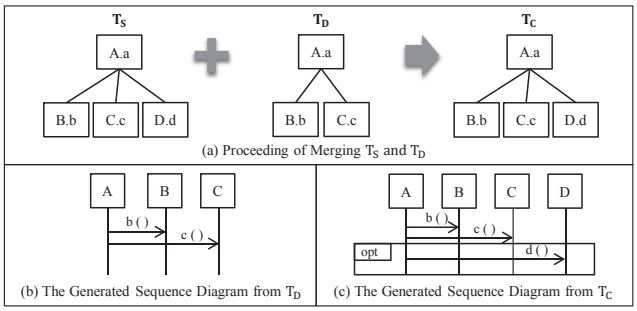


Figure 5. Applying *Identification of an Unexecuted Message* Rule

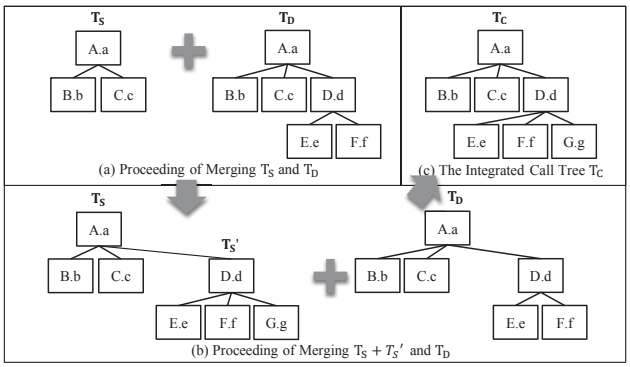


Figure 6. Applying *Identification of a Late Binding Behavior* Rule

5. The method node M(*j*,*d*) is marked as the message included in an option(OPT) frame to denote alternatives in the generated sequence diagram.

Figure 5 depicts a situation for applying R4. The method node D.d in T_S does not exist in the corresponding dynamic tree T_D. It means that the method *d( )* in the class D is implemented but it is not captured in the execution trace logs. According to the rule R4, the method node D.d is added to the integrated call tree T_C. Consequently, the missed *d( )* method call in the sequence diagram from T_D is captured in the sequence diagram from the integrated call tree T_C. As shown in the example, the application of R4 can help to get wider coverage of reverse engineering.

R5. Identification of a Late Binding Behavior

1. Compare with the current method node M(*i*,*d*) in T_D and the current method node M(*j*,*d*) in T_S.
2. If M(*j*,*d*) is null, generate a new static call tree T_S' which has the method node M(*i*,*d*) as a root from source code. And the root node of T_S' to the original call tree T_S as the next neighbor of the node M(*j*,*d*).
3. If M(*j*,*d*) is not null and it is different with the node M(*i*,*d*), compare with the M(*i*,*d*) in T_D and the next neighbor node M(*j+1*,*d*) in T_S again.
4. If the node M*(j+1,d)* is different with the node M*(i,d)* or null, generate a new static call tree T_S' which has the method node M(*i*,*d*) as a root from source code. And the root node of T_S' to the original call tree T_S as the next neighbor of the node M(*j*,*d*).
5. Continue the next node comparison from M(*i*,*d*) in T_D and M(*j*,*d*) in T_S. Application of the five merging rules is restarted.

As depicted in Figure 6, the method *d( )* in the node D.d of T_D is not identified in T_S. From this comparison, the dynamic binding of the method *d( )* can be captured and reflected to the integrated call tree T_C. Without referencing the dynamic call tree T_D, the generated sequence diagram from T_S will have only three method calls (A.a, B.b, C.c). By applying R5 in construction of the integrated call tree, however, the extended static call tree can have the method call D.d and its successive method calls (E.e, F.f, G.g) as nodes. By continued comparison between the extended static call tree and the dynamic call tree, R1 and R4 are applied to construct an integrated call tree T_C. Figure 6(c) shows the final integrated call tree. Application of R5 results in broadening the coverage of reverse engineered messages in common with R4.

E. Sequence Diagram Generation

To generate a sequence diagram from the integrated call tree, each method node in the integrated call tree is traversed according to a pre-order of depth-first traversal algorithm in the same way as merging call trees. Each method in the integrated call tree is mapped to each message in the sequence diagram. We also implement a tool for supporting the sequence diagram generation with the integrated call tree.

IV. CASE STUDY

In this section, a case study is presented using REQ Modeler [11], which is open source software for drawing a UML model. A flow thread of the REQ Modeler is demonstrated to show the application procedure of several proposed merging rules. Figure 7 depicts the proceeding of an integrated call tree construction for *Delete a Package* flow of events.

The first method call in the selected flow is *execute( )* of *DeletePackage* class. At first, the *DrawableNode.delete FromDiagram( )* method node in depth 1 is identically defined in both of the static call tree and the dynamic call tree. Thus, the *DrawableNode.deleteFromDiagram( )* method node is added in the integrated call tree according to the rule R1. At second depth, there exists the *RCanvas.removeChild( )* method node in the dynamic call tree T_D. However, the *DrawableNode. DeleteFromDiagram( )* method node is the last one in the static call tree T_S. The merging rule R5 is applied for adding the successive methods after dynamic binding of the *RCanvas. removeChild( )* into the integrated call tree. With the extended

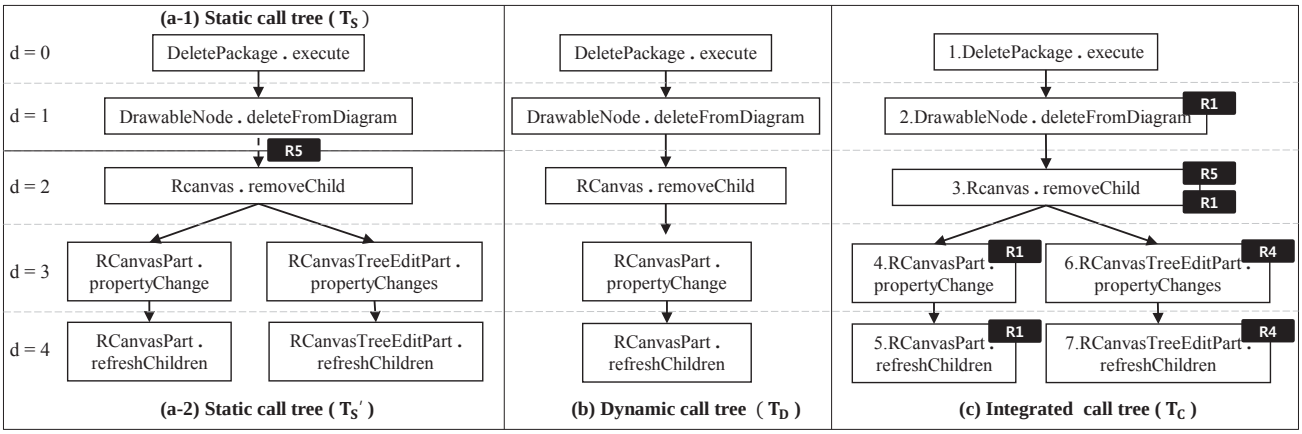


Figure 8. Construction of an Integrated Call Tree: *Delete a Package* in REQ Modeler

static call tree(T_S+ T_S') in Figure 7(a), the comparison from the *RCanvas. removeChild( )* method node at second depth is restarted. The *RCanvas. removeChild( )* method node is added to the integrated call tree because it is identically defined in both trees(R1). With the same reason, *RCanvasPart.Property Change ( )* method at third depth is also added to the integrated call tree. The next method node *RCanvasTreeEditPart. propertyChange ( )* in the static call tree T_S is not defined in the dynamic call tree T_D. In this case, the merging rule R4 is applied to add the unexecuted method into the integrated call tree. The method nodes at fourth depth have the same conditions with the nodes at third depth. In other words, the identically defined method node *RCanvasPart.refreshChildren ( )* and the unexecuted method *RCanvasTree EditPart.refresh Children( )* which is defined in the static tree are added to the integrated call tree by applying rules R1 and R4.

Figure 8 shows the sequence diagrams which are respectively generated from the call trees in Figure 7. Besides one to one mapping between methods in call trees and messages in sequence diagrams, the pros and cons of each reverse engineering method can be discussed through looking at the generated sequence diagrams depicted in Figure 8. As shown in Figure 8(a) and Figure 8(b), two separated sequence diagrams for a single flow of event are generated from the static call trees in Figure 7(a). On the other hand, in Figure 8(c), the sequence diagram which traces only one thread of events is

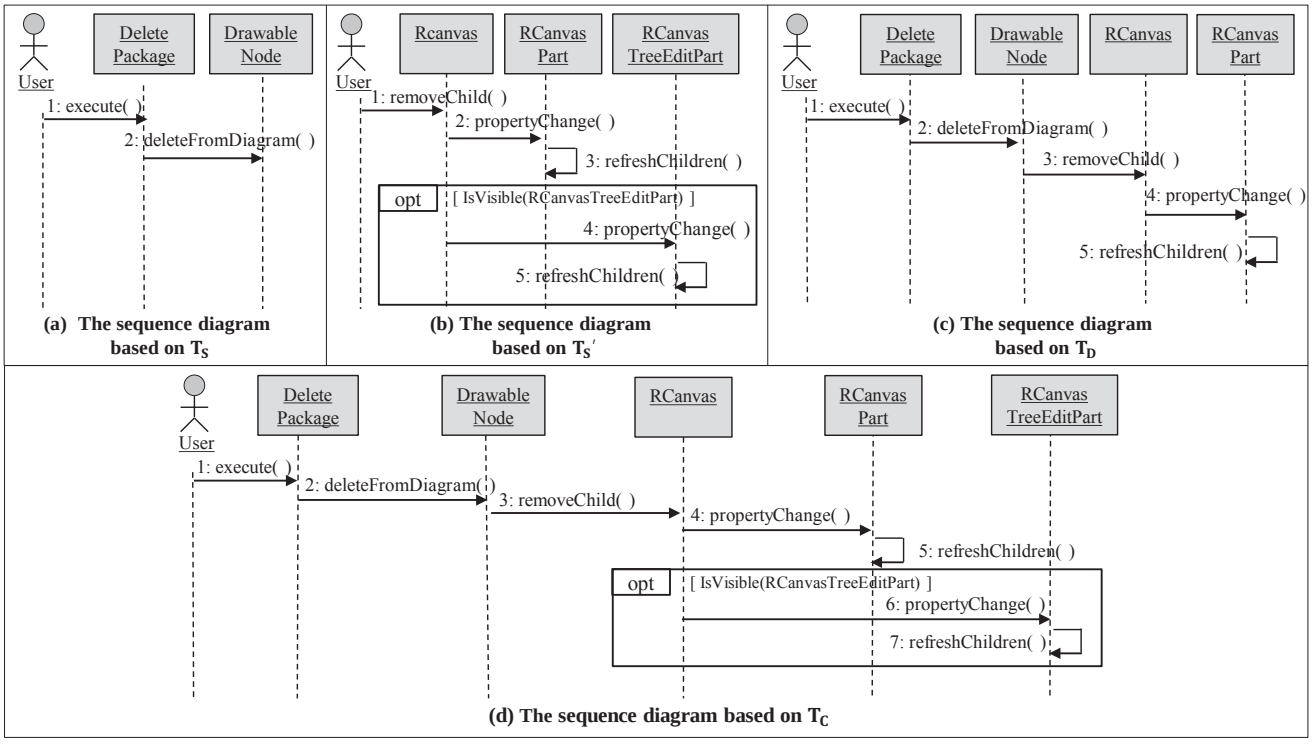


Figure 7. Reversed Sequence Diagrams for *Delete a Package* flow in REQ Modeler

generated from the static call tree in Figure 7(b). However, the sequence diagram generated from the integrated call tree in Figure 8(d) includes the whole main flow and the *opt* frame for an unexecuted alternative flow for *Delete a Package*.

V. EVALUATION

To evaluate the effectiveness of the presenting method, we applied the method to two different pieces of open source software. One is the REQ Modeler which is already demonstrated in the case study. The other one is TerpPaint [17] which is a painting program. REQ Modeler is composed of 220 classes, 1266 functions and 20486 LOC. On the other hand, TerpPaint is a comparatively small sized program. It is composed of 65 classes, 644 functions and 16314 LOC. Both programs are implemented by Java language. And they have lots of user interfaces for authoring UML model or image manipulation. Seven sequence diagrams of REQ Modeler and five diagrams of TerpPaint are respectively generated. To evaluate benefits from using the proposed method, we compared the results from our method and other existing tools, Together [18] and TPTP [19]. The former is a static analysis based reverse engineering tool and the latter is a dynamic analysis based reverse engineering tool.

We set three metrics for measuring the quality of generated sequence diagrams. Table 1 shows comparison between the results from applying the existing tools and the result of our tool supporting the proposed method.

- The number of messages in a sequence diagram

The first metric is the number of messages appeared in a diagram for measuring the size of each sequence diagram. The readability problem is caused by size explosion of sequence diagram in dynamic analysis. Recently presented hybrid reverse engineering method [7][9][10] have been tried to prove their effectiveness by using compaction ratio. The compaction ratio explains how much percentage of trivial messages or repeated messages are removed. Thus, compaction ratio is defined as in the following:

$$\text{Compaction Ratio (\%)} = \left[1 - \frac{\#\text{ of message from our method(C)}}{\#\text{ of all messages from dynamic analysis(A + B)}}\right] \times 100$$

Our averaged compaction ratio is 86.39%. There is a little bit wide deviation on the compaction ratio from two programs. Whereas the averaged compaction ratio of REQ Modeler is 66.11%, the averaged compaction ratio of TerpPaint is 91.98%. We investigated the reason why the compaction ratio of REQ Modeler is much lower than TerpPaint. The reason why the compaction ratio of REQ Modeler is much lower than TerpPaint is investigated. At the result, it turned out that the execution trace logs from REQ Modeler have inherently fewer loops than the execution logs from TerpPaint.

- The number of unintentionally sliced diagrams

The second metric is the number of unintentionally sliced diagrams. It is general to slice a diagram into several fragments to factor out some parts of a flow from the diagram for reusing or other reasons. However, unintentionally sliced diagrams due to polymorphism or late binding which is implemented in source code can hinder in understanding the behaviors of a system. According to the results in Table 1, there is no slicing of diagram in REQ Modeler and TerpPaint. It shows that our call tree merging rules worked well on resolving the unintentionally sliced diagram problem in case of the two programs.

TABLE 1. THE NUMBER OF SEQEUNCE DIAGRAM MESSAGES

	Name of Flow of Event	Together (Static)		TPTP(Dynamic)		The Call Tree Merging Method			
		message	slicing	message (A)	missing message (B)	message (C)	slicing	missing message	Comp-action Ratio
REQ MODELER	Open Project	27	0	95	0	26	0	0	72.63
	Open Diagram	7	9	207	0	37	0	0	82.13
	Add Class	70	2	407	0	41	0	0	89.93
	Move Class	3	1	12	7	14	0	0	26.32
	Delete Class	26	2	50	5	14	0	0	74.55
	Connect Class	10	1	87	12	42	0	0	57.58
	Save Project	4	2	58	4	25	0	0	59.68
	Average	21.00	2.43	130.86	4.00	28.43	0.00	0.00	66.11
TERP PAINT	Drag Rectangle	2	2	358	0	13	0	0	96.37
	Drag Eraser	4	2	551	0	16	0	0	97.10
	Click Pencil	5	1	18	0	12	0	0	33.33
	Click Paint bucket	11	1	46	0	15	0	0	67.39
	Save File	52	0	286	1	45	0	0	84.27
	Average	14.80	1.20	251.80	0.20	20.20	0.00	0.00	91.98
Average		18.42	0.92	181.25	2.42	25.00	0.00	0.00	86.39

- The number of missing messages

The last metric is the number of missing messages. In dynamic analysis result, an unexecuted method cannot be grabbed as an execution trace. Consequently, it can cause leakage of valuable information in reverse engineering. The number of missing message is counted to measure the amount of the information leakage. According to the results in Table1, there is no missing message. It shows that the merging rule R4 works effectively in finding messages which cannot be captured by execution trace logs by searching the corresponding static call tree.

VI. CONCLUSION

We have presented a method for reverse engineering of sequence diagram by applying rules for merging a static call tree from source code and a dynamic call tree from execution trace logs. Unlike other hybrid method in reverse engineering where the contribution is focused only on the compaction of messages captured by dynamic analysis by utilizing the information form static analysis result, our method guarantees the completeness of the generated sequence diagram besides the compactness. To overcome the coverage problem of reverse engineering, the proposed call tree merging rules are designed

to supplement the information leakage from both of a static call tree and a dynamic call tree. We have demonstrated use of the proposed method in generating sequence diagrams for REQ Modeler. Besides the case study with REQ Modeler, the result of applying the proposed method is also evaluated to the REQ Modeler and another open source, TerpPaint. With the two examples, the effectiveness of the presented rules is measured for merging call trees in reducing the size of the generated sequence diagram by measuring the compaction ratio. And the completeness of the generated sequence diagram is also measured from counting the number of unintentionally sliced diagrams and missing messages. The proposed merging rules contribute to provide readable sequence diagrams with reasonable compaction ratio of messages and to broaden the coverage of the reversed sequence diagrams.

However, there exist limitations in capturing a few types of execution traces. At first, if a dynamic binding is occurred in an unexecuted thread of method calls, it is still incapable to capture the succeeding method calls after occurrence of the dynamic binding. For the future work, we have a plan to find more uncovered cases and add more merging rules to provide a perfectly complete coverage of reverse engineering. The second is to capture a single trace from the executions of a multiprocessing system. But the separation of execution traces according to each process is out of our research questions. Furthermore, we will apply the proposed method to more cases to check if the merging rules are biased in a specific system or not.

ACKNOWLEDGMENT

This research was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2012M3C4A7033348)

REFERENCES

[1] R. L. Glass, Fact and Fallacies of Software Engineering. Addison-Wesley, 2002.

[2] J. Kern and C. Garrett, "Effective Sequence diagram generation," Borland white paper, 2003.

[3] A. Rountev, O. Volgin and M. Reddoch, “Static Control-Flow Analysis for Reverse Engineering of UML Sequence Diagrams,” In Proceedings of the ACM Workshop Program Analysis for Software Tools and Engineering(PASTE), pp. 96-102, 2005.

[4] S. Staiger, “Reverse Engineering of Graphical User Interfaces using Static Analyses,”. In Proceedings of the 14th Working Conference on Reverse Engineering(WCRE), pp.189-198, 2007.

[5] L. C. Briand, Y. Labiche, and J. Leduc, "Toward the reverse engineerng of UML sequence diagrams distributed java software," IEEE Transaction Software Engineering, vol. 32, pp.642-663, 2006.

[6] H. Safyallah and K. Sartipi, “Dynamic Analysis of Software System using Execution Pattern Mining,” In the proceedings of the 14th IEEE International Conference on Program Comprehension (ICPC), pp.84-88, 2006.

[7] K. Taniguchi, T. Ishio, T.Kamiya, S.Kusumoto and K.Inoue, "Extracting Sequence Diagram from Execution Trace of Java Program," In the proceedings of the 8th International Workshop on Principles of Software Evolution (IWPSE), pp.148-154, 2005.

[8] T. Eisenbarth, R. Koschke and D.Simon, "Aiding Program Comprehension by Static and Dynamic Feature Analysis," In proceedings of the IEEE International Conference on Software Maintenance(ICSM), pp.602-611, 2001.

[9] D. Myers, M. A. Storey and M. Salois, "Utilizing Debug Information to Compact Loops in Large Program Traces,” In the Proceedings of the 14th European Conference on Software Maintenance and Reengineering(CSMR), pp.41-50, 2010.

[10] K. Noda, T. Kobayashi and K. Agusa, “Execution Trace Abstraction based on Meta Patterns Usage,” In Proceedings of the 19th Working Conference on Reverse Engineering(WCRE), pp.167-176, 2012.

[11] REQModeler, http://sourceforge.net/projects/reqmodeler/?source=directory

[12] J. Warmer and A. Kleppe, The Object Constraint Language: Getting Your Models Ready for MAD, Second Edition. Addison Wesley, 2003.

[13] W. Pree, Design Patterns for Object-Oriented Software Development. Addison Wesley, 1994.

[14] AST, http://en.wikipedia.org/wiki/Abstract_syntax_tree

[15] EClipse, http://www.eclipse.org/

[16] BTrace, http://kenai.com/projects/btrace

[17] TerpPaint, http://sourceforge.net/projects/terppaint/?source=directory

[18] BorlandTogether, http://www.borland.com/products/Together/default.aspx

[19] TPTP, http://www.eclipse.org/tptp/

Mining Architectural Patterns Using Association Rules

Cristiano Maffort, Marco Tulio Valente, Mariza Bigonha
Department of Computer Science
UFMG, Belo Horizonte, Brazil
{*maffort,mtov,mariza*}*@dcc.ufmg.br*

André Hora, Nicolas Anquetil
RMoD Team
Inria, Lille, France
{*andre.hora,nicolas.anquetil*}*@inria.fr*

Jonata Menezes
Department of Computer Engineering
CEFET-MG, Belo Horizonte, Brazil
jonata@dri.cefetmg.br

***Abstract*—Software systems usually follow many programming rules prescribed in an architectural model. However, developers frequently violate these rules, introducing architectural drifts in the source code. In this paper, we present a data mining approach for architecture conformance based on a combination of static and historical software analysis. For this purpose, the proposed approach relies on data mining techniques to extract structural and historical architectural patterns. In addition, we propose a methodology that uses the extracted patterns to detect both absences and divergences in source-code based architectures. We applied the proposed approach in an industrial-strength system. As a result we detected 137 architectural violations, with an overall precision of 41.02%.**

***Keywords*-Software architecture conformance; Frequent itemset mining; Static analysis; Mining software repositories**

I. Introduction

The architecture of a system prescribes the organization of its components, their relationships, constraints, and the principles that guide its design and evolution over time [1]–[3]. An architectural model is a high-level representation of the software that documents and transmits the major decisions and principles that should be followed during the software development project.

However, during development of a software product, programming anomalies regarding the proposed architectural model are normally introduced. These anomalies are classified in this paper as architectural violations [4], [5]. In practice, the introduction of architectural violations is very common [6]. These violations usually make more complex subsequent maintenance tasks since the concrete architecture is not adhering to the planned and documented architecture [7].

Therefore, in this paper we assume that the inception of architectural violations in software products is a common task [8]. Moreover, we assume that some violations are detected and corrected in future revisions through inspection and/or quality assure activities. Furthermore, programs usually follow architectural patterns of implementation. With this in mind, it is observed that, according to software design best practices, classes belonging to the same component follow similar programming conventions.

Based on these assumptions, this paper proposes a method to detect architectural violations in software products. The proposed solution analyzes structural and historical architectural patterns at the level of structural dependencies between classes and considering the versions stored in a version control repository. The ultimate goal is to identify architectural violations from similar dependency patterns. Particularly, a structural dependency denotes any syntactic relation between two classes, including method calls, field and variables declaration, etc.

The proposed approach extracts architectural patterns that can be used, for example, as documentation artifacts. Furthermore, the detection method is statically performed in a non-invasive way, so it does not impact normal system programming activities. In order to evaluate our approach, this paper describes its application in a real information system used by a major Brazilian university. As result, we identified 334 evidences of architectural violations in this system. From such evidences, 137 were confirmed by a senior software developer, which implies in a precision of 41.02%.

The remainder of this paper is organized as follows. Section II presents an overview of the proposed approach. Sections II-A and II-B describe the heuristics to detect absences and divergences, respectively. Section III presents an evaluation of the proposed approach in a real system. Section IV describes related work and Section V presents the conclusions.

II. Proposed Approach

This paper proposes a technique for detecting architectural violations in object-oriented software systems. The proposed approach relies on data mining techniques over historical dependencies between the classes of a target system. This historical information is retrieved from the version control system repository. Basically, the proposed approach mines structural and historical dependencies between the classes of the target system.

Figure 1 illustrates our approach for detecting architectural violations. Initially, a *Code Extractor* component retrieves all source code versions from the version control system repository. Each revision is parsed by the VerveineJ[1] parser that extracts the dependencies from the source code. Next, the extracted dependencies are stored in a relational database. The *Architectural Miner* component relies on two types of input on the target system: (a) dependencies database and (b) high-level component specification. In our approach, we assume that classes are statically organized in modules (packages in Java terminology)

[1] https://gforge.inria.fr/projects/verveinej

and modules are logically arranged in coarse–grained structures called components. The high-level component specification is essentially a mapping from modules to the defined components. Next, the *Architecture Miner* generates a Prolog database describing the structural and historical relations available in the source code. After that, the *Architectural Miner* uses Prolog queries to convert the Prolog database into a consistent frequent itemset mining dataset. Next, an association rule mining algorithm is used to detect structural and historical architectural patterns. Finally, the *Violation Detector* uses such architectural patterns to detect architectural violation evidences according to the methodology described in Sections II-A and II-B.

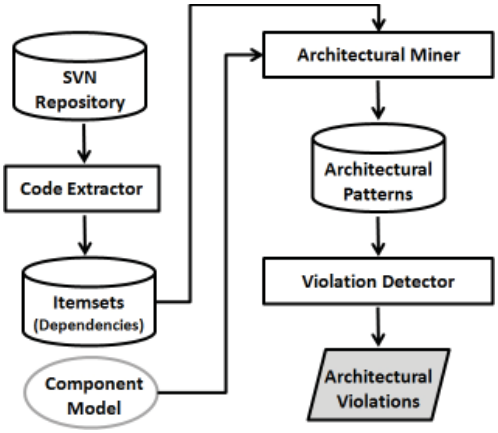


Figure 1. Proposed approach

In the proposed component model, high-level components are represented as simple regular expressions that represent the mapping from modules to components.

Our approach is based on a data mining technique called *frequent itemset mining* [9], which efficiently finds frequent itemsets in a dataset. Basically, this technique defines the *support* as the number of occurrences of a subset of items (sub-itemset). A sub-itemset is considered frequent if its support is greater than a specified threshold called *minimum support*. Thus, support counts the number of times a sub-itemset happens in the itemsets database.

After the frequent itemset has been mined, we can compute *association rules* [10], [11]. From the association rules, we make assumptions that two or more items occur simultaneously or conditionally. Furthermore, association rules can be used to discover causal relationships among elements. Each association rule has a *confidence*, which is a metric that represents the probability of a database transaction covered by a antecedent term (pre-condition of the rule) be covered by a consequent term (consequence of the rule).

To calculate frequent itemsets and to generate association rules, we use a FP-tree-based mining algorithm, called FPGrowth [10]. Instead of generating the complete set of frequent sub-itemsets, this algorithm generates only relevant itemset candidates. After the frequent itemsets are mined, FPGrowth also generates association rules. The proposed approach to architectural violation detection is based on the idea that an architectural pattern is frequently followed and violations represent a small percentage of the cases.

The remainder of this section is organized as follows: Section II-A presents the heuristics used to detect evidences of absences; Section II-B describes the heuristics for divergences.

A. Mining for Absences

An absence is a violation that occurs when a $BaseClass$ does not depend on a $TargetClass$, but a dependency like that *is prescribed* by the planned architecture. In other words, an absence is a violation that happens with a dependency defined by the planned architecture but that does *not exist* in the source code [4], [12]. Figure 2 illustrates an example of absence. In this case, the planned architecture prescribes that classes located in a DTO module must use services provided by a class located in JPA module. In this case, an absence is counted for each class in DTO that does not follow this rule.

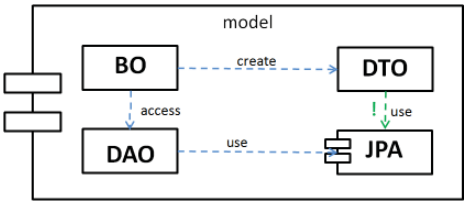


Figure 2. Example of absence (DTO must use JPA)

In order to detect absences we initially search for patterns of dependencies that frequently occur. Next, we search for dependencies that violate such patterns, and therefore denote minorities at the level of components. We assume that absences occur in a small percentage of cases, which are more likely to represent architectural violations. Additionally, we use the history of versions to mine for evolutionary architectural patterns. In this case, we search for patterns in which dependencies are introduced in classes originally created without such dependencies.

The proposed procedure for detecting absences relies on two steps. First, we identify architectural patterns that *frequently occur* in classes grouped as defined by the component model provided as input. Second, from classes in each component, we identify evolutionary architectural patterns. For instance, considering the example in Figure 2, we check how frequently classes in the component $Model$ that depend on $Entity$ (a class of the JPA module) in the current version of the system were initially created without this dependency.

The main idea of the evolutionary architectural patterns is to reinforce the violation evidences suggested by the first step. The assumption is that absences are frequently detected and fixed (i.e., classes created without a dependency prescribed by the planned architecture are frequently fixed in future revisions).

In order to find correlations among the dependencies, initially it is necessary to compute the frequent itemset

mining dataset. For this purpose, we rely on a dataset based on Prolog facts, which describes the dependencies and historical information on the classes of the system under analysis, as follows:

```
[component(CompId,CompName).]+
[module(ModId,CompId,ModName).]+
[class(ClassId,ModId,ClassName).]+
[dependency(DepId,BaseClassId,TargetClassId,
    CreatedWith,ExistCurrently,AddAny).]+
```

The *component* predicate defines the components defined by the architect of the system under analysis. The *module* predicate defines the packages, in Java terminology, of classes of the system. The *class* predicate describes a class in the system. The *dependency* predicate defines a dependency relation between two classes ($BaseClassId$ depends on $TargetClassId$). In the *dependency* predicate, the attribute $CreatedWith$ informs whether the dependency was created together with the $BaseClassId$, the attribute $ExistCurrently$ informs whether the dependency exists on the last version of the system, and the attribute $AddAny$ informs if the dependency existed in some version of the system.

In the first step, each class and its dependencies in the last version under analysis (attribute $ExistCurrently = true$) are written as a row into the itemset database, as follows:

```
BaseComponent(bcomp),BaseClass(bclass)
    [,TargetModule(tmod),TargetClass(tclass)]*
```

By mining this itemset database using FPGrowth algorithm, we can find the frequent sub-itemsets and generate the association rules of the corresponding *architectural pattern*, which represent dependencies that are frequently used together. Moreover, the FPGrowth requires the definition of a support (A_{dps}) and a confidence (A_{dpc}) threshold. For instance, suppose a pattern like that:

```
{BaseComponent('domain')}=>
    {TargetClass('Entity')}
```

This pattern states that all classes on the component *domain* (antecedent term of the association rule) should depend on the class *Entity* (consequent term of the association rule). Therefore, regarding this pattern, classes in the *domain* component that do not depend on *Entity* represent an absence violation.

The second step is used to reduce the amount of false violations. For each component in the system, we select the dependencies and the historical information from the Prolog facts database. In this particular case, we select the attributes $CreatedWith$ and $ExistCurrently$. Each dependency generates a row in the itemset database as follows:

```
BaseComponent(bcomp),TargetClass(tclass),
    CreatedWith([true|false]),
    ExistCurrently([true|false])
```

We compute the association rules of the corresponding *dependency evolution patterns* using the FPGrowth algorithm, using a given support (A_{deps}) and confidence (A_{depc}) threshold. The results are combined with the results obtained in the first step. For example, suppose that in the first step the classes in the $Model$ component that not depend on $Entity$ were classified as evidences of absences. Moreover, suppose that in the second step we found that classes in $Model$ created without a dependency with $Entity$ frequently (i.e., with a high confidence) added this dependency during their evolution, which therefore reinforces the evidence detected in the first step.

B. Mining for Divergences

A divergence is a violation that happens when a $BaseClass$ depends on a $TargetClass$, although such dependency *is not prescribed* by the planned architecture. In other words, a divergence is a violation due to a dependency that is not allowed by the planned architecture, but that *exists* in the source code [4], [12]. Figure 3 illustrates an example of divergence. In this case, the planned architecture prescribes that classes located in the *BO* module must not directly depend on the *JPA* module. In this particular example, a divergence is counted for each class in *BO* which relies on services provided by the *JPA*.

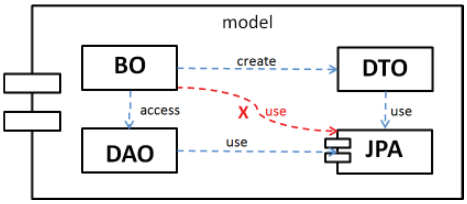


Figure 3. Example of divergence (BO cannot use JPA)

Likewise the heuristic for absences, we assume that divergences happen in a *small percentage* of cases. Therefore, a standard frequent itemset mining technique is not suitable for detecting minorities. However, divergences frequently do *not exist* in most classes of a component. More specifically, the divergences detection relies on two steps. First, we identify the dependencies that frequently do not occur in the classes of a given component. In the second step, we identify how frequently classes in this component have established and then removed a dependency like that in the past.

In the first step, we initially select all classes in the last version of the target system. For each $BaseClass$, we select the dependencies that do *not exist* between $BaseClass$ and a $TargetClass$, where $TargetClass$ is a class used by the component that contains $BaseClass$. Then, these items generate a row in the itemset database, as follows:

```
BaseComponent(bcomp),BaseClass(bclass)
    [,TargetModule(tmod),TargetClass(tclass)]*
```

Using FPGrowth algorithm, we compute the association rules, according to a given support (D_{dps}) and confidence (D_{dpc}). For instance, the following association rule states that classes in the $Model$ component frequently do not depend on $HttpServlet$.

```
{BaseComponent('model')}=>
    {TargetClass('HttpServlet')}
```

Therefore, the classes in $Model$ that depend on $HttpServlet$ represent an evidence of divergence.

In the second step, we perform a historical analysis to reduce the number of false positives. In this case, we select dependencies from the itemset database including the attributes $ExistCurrently$ and $AddAny$. This information generates an itemset in our database as follows:

```
BaseComponent(bcomp),TargetClass(tclass),
    AddAny([true|false]),
    ExistCurrently([true|false])
```

Applying the FPGrowth, using D_{deps} and D_{depc} as support and confidence respectively, we obtain the association rules for the *dependency evolution patterns*. Then, these results are combined with the results obtained in the first step. For instance, suppose that it was previously mined in the first step that the classes in the $Model$ component that depend on $HttpServlet$ represent evidences of divergences. Moreover, suppose that in this second step we discover that such classes removed the dependencies with $HttpServlet$ during their evolution. In this case, the evidence detected in the first step is reinforced by this second finding.

III. Evaluation

To evaluate our approach for detecting absences and divergences, we performed a study in a information system, called SGA, from a major Brazilian university. The SGA system automates many administrative activities, including human and material resource management, incomes/expenses, among others. The last revision considered in our study has 1,852 classes and interfaces, organized in 104 packages, comprising around 127 KLOC.

The SGA system follows a Model-View-Controller (MVC) architecture. The $Model$ layer has three main modules: $domain$, $persistence$, and $service$. The $domain$ module handles business objects, such as Students, Professors, etc. The $persistence$ module provides database transactional methods, such as insert, update, delete, etc, that are used to persist business objects in a relational database. The $service$ module handles the state of the domain objects according to the workflow and business rules required by the information system.

The $View$ layer is implemented in *Java Server Pages* and uses *JavaServer Faces* components. Basically, this layer provides a way to interact with the system, receiving and displaying results of the requests made by the users.

The $Controller$ layer provides a bridge between user interface and business-related components, transferring and adapting the user inputs.

A. Dataset

To detect absences and divergences, initially we retrieved 4,923 revisions of the SGA system, which is maintained in a $Subversion$ repository. Each revision was parsed by VerveineJ and the extracted dependencies were stored in a relational database with 4.5 GB.

Next, an architect defined its high-level component model. Finally, the high-level components and the dataset of historical dependencies were used as input to generate the Prolog facts. We executed our approach as described in Sections II-A and II-B. Then, the architect of the SGA system inspected the selected violations in order to classify them as true or false positives.

B. Results for Absences

As reported in Section II-A, the detection of absences relies on four thresholds: A_{dps} and A_{dpc}, the support and confidence of the structural dependency architectural patterns, and A_{deps} and A_{depc}, the support and confidence of historical dependency evolution patterns. Table I shows the values used for such thresholds:

Table I
Absences Thresholds.

Threshold	Value
A_{dps}	0.1
A_{dpc}	0.9
A_{deps}	0.1
A_{depc}	0.6

Basically, we consider as an architectural pattern only the rules that occurred in at least 10% of the classes and that present a confidence of at least 90%. For the architectural evolution patterns, we consider thresholds of 10% for support and 60% for confidence. Therefore, we consider as evidence of architectural violation classes that violated a rule followed by at least 90% of the other classes. Furthermore, only classes whose historical evolution rules were higher than 60% were considered as violations, i.e., at least 60% of the classes created with a violations regarding the rule have been later refactored to follow the rule.

C. Results for Divergences

The detection of divergences relies on four thresholds: D_{dps} and D_{dpc}, denoting respectively the support and confidence of the structural dependency architectural patterns, and D_{deps} and D_{depc}, denoting respectively the support and confidence of the historical dependency evolution patterns. Table II shows the thresholds values used for divergences.

Table II
Divergences Thresholds.

Threshold	Value
D_{dps}	0.1
D_{dpc}	0.9
D_{deps}	0.1
D_{depc}	0.25

Similarly to the absence detection, for divergences we consider architectural pattern rules with support of 10% and confidence of 90%. On the other hand, for the architectural evolution patterns, we consider the thresholds of 10% and 25% for support and confidence, respectively. In this case, we select as divergences the classes that violate

both considered architectural patterns. More specifically, we select classes that depend on a class when at least 90% of the classes in the same component do not follow this rule. Furthermore, when in the past other classes added this dependency, in at least 25% of the cases the dependency was later removed.

D. Results

Our approach was applied in the SGA system using the thresholds defined in Sections III-B and III-C. The triggered violations were inspected by the SGA architect, who classified them as true or false violations.

As we can observe in Table III, we detected 261 evidences of absence, and 101 were classified as true-positives by the SGA architect. Furthermore, we triggered 73 divergence warnings, which 36 were classified as true-positives. Thus, the precision was 38.7% and 49.32% to absences and divergences, respectively. As total, the architect inspected 334 warnings, which 137 were considered true-positives, resulting in a global precision of 41.02%.

Table III
ARCHITECTURAL VIOLATIONS OF SGA SYSTEM.

	Absence	**Divergence**	**Total**
Warnings (E)	261	73	334
True-positives (TP)	101	36	137
False-positives (FP)	160	37	197
Precision (TP/E)	38.7%	49.32%	41.02%

IV. RELATED WORK

Lint [13] was one of the earliest and most successful tool to detect bugs and bad smells in software products. With the success achieved by Lint, many other static analysis tools to detect questionable programming strategies have been proposed. FindBugs [14] and PMD [15] are examples of tools inspired by Lint that highlight among the most popular tools to detect anomalies on Java programs. Null pointer dereferences, overflow in arrays, uncaught exceptions and security vulnerabilities are examples of suspicious programming constructs and events analyzed by FindBugs and PMD. However, such tools are not designed to detect architectural anomalies such as the ones associated to violations in the planned architecture of object-oriented systems. Moreover, in previous studies we concluded that FindBugs present precision rates less than 50%, which are only achieved when the tool is properly configured to raise particular categories of warnings [16]. In other study, we concluded that there is no static correspondence between field defects and warnings raised by FindBugs, although it seems to exist a moderate level of correlation between warnings and such kinds of software defects [17].

Several tools have been proposed to analyze version control software repositories and extract programming patterns. Zhou and Zhenmin present a tool, called PR-Miner (based on the frequent itemset data mining technique) for automatic extraction of programming rules [18]. The proposed approach uses a formalism to extract dependencies between functions that are heavily dependent on procedural languages. The proposed strategy for detecting violations only considers function call flows, independent of the modular and/or architectural context in which these calls occurred. On the other hand, the approach presented in this paper is focused on the detection of architectural violations. However, it is important to note that the precision values presented by PR-Miner were generally lower than those reported in Section III. For example, from the 60 warnings with higher priority triggered during Linux analysis, only 16 were true programming errors (bugs).

Mileva et al. conducted an analysis on evolution patterns between two versions of a system to detect pending changes in source code [19]. This study is supported by a tool called Lamarck. Such tool mines evolution patterns in software repositories by abstracting object usage into temporal properties in order to detect pending changes. From the pending changes, they recommend fixes based on usage patterns. Similarly to the study of Zhou and Zhenmin the approach used by Mileva et al. also analyze dependencies between functions. Therefore, it targets concepts of procedural languages, disregarding typical object-oriented language dependencies, such as inheritance. Moreover, in this paper, we take into account, in addition to the present formalism of object-oriented languages, the entire history of changes. Despite these differences in focus and languages, it is important to note that the approach described in this paper, in general, was able to discover a greater number of violations. For example, the approach of Mileva et al. was able to find only six violations in a case study involving Eclipse 1.0 and 2.0 platforms.

Among architecture conformance techniques, reflexion models currently highlight as the main technique based on models. Such approach compares a high-level model, manually created by the architect, with a concrete model, automatically extracted from the source code [12], [20]. However, the application of reflexion models for architecture conformance usually requires successive refinements in the high-level model to reveal the whole spectrum of absences and divergences. On the other hand, the approach proposed in this paper is more lightweight, demanding a simple high-level component specification.

Sarkar et al. [7] conducted a study aiming to discover layered organization models of software systems. The architectural model generated was used to detect architectural violations through dependencies among modules. They have only detected calls violating layer hierarchical structures. On the other hand, in this paper, we have also presented a methodology to detect absences between two layers and divergences between components, which do not necessarily need to follow a hierarchical structure.

Besides reflexion models, another common solution for architecture conformance is centered on domain-specific languages. In this case, Terra and Valente proposed a declarative language, called DCL (Dependency Constraint Language), for constraint dependency that statically

checks the architecture of a software in relation to restrictions defined by an architect [8]. Therefore, DCL requires an architect to define constraints, and a tool included in the solution verifies only what the architect has prescribed. On the other hand, in our approach the architect does not need to manually specify architectural constraints, which invariably tends to be a tedious and error-prone task.

V. Conclusion

Software systems frequently follow several programming conventions. During software evolution, the development team commonly uses programming strategies that do not adhere to the planned architecture for the system.

This paper presented an approach that use frequent itemset mining techniques to architecture conformance. We consider both dependencies prescribed in the planned architectural model but absent in the source code, as well as dependencies presented in the source code but absent in architectural model.

We evaluated the proposed approach with a large information system. We detected 137 architectural violations, divided in 101 absences and 36 divergences, with precision of 38.7% and 49.32%, respectively, and a global precision of precision of 41.02%.

As future work, we intend to extend the study evaluating specific correlations between dependencies as well as classifying the dependency type, such as attributes, annotations, inheritance, etc. Additionally, we intend to conduct a sensibility analysis in order to discover the best combination of values for the thresholds used by our approach. Finally, we plan to apply our study in case studies involving systems using architectural patterns different from SGA system. We also plan to integrate our approach for architecture conformance with ArchFix [21], which is a recommendation tool that suggests refactorings for repairing architectural violations.

Acknowledgments

This research has been supported by CAPES, FAPEMIG, and CNPq. We thank the architect of the SGA system for validating the warnings raised by proposed approach.

References

[1] D. Garlan, "Software architecture: a roadmap," in *Conference on The Future of Software Engineering*, ser. ICSE '00, 2000, pp. 91–101.

[2] D. Garlan and M. Shaw, *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996.

[3] M. Fowler, *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2002.

[4] L. Passos, R. Terra, R. Diniz, M. T. Valente, and N. Mendonca., "Static architecture-conformance checking: An illustrative overview," *IEEE Software*, vol. 27, no. 5, pp. 82–89, 2010.

[5] D. E. Perry and A. L. Wolf, "Foundations for the study of software architecture," *SIGSOFT Software Engineering Notes*, vol. 17, pp. 40–52, 1992.

[6] J. Knodel and D. Popescu, "A comparison of static architecture compliance checking approaches," in *6th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, 2007, p. 12.

[7] S. Sarkar, G. Maskeri, and S. Ramachandran, "Discovery of architectural layers and measurement of layering violations in source code," *Journal of Systems and Software*, vol. 82, pp. 1891–1905, 2009.

[8] R. Terra and M. T. Valente, "A dependency constraint language to manage object-oriented software architectures," *Software: Practice and Experience*, vol. 32, no. 12, pp. 1073–1094, 2009.

[9] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in *20th International Conference on Very Large Data Bases*, 1994, pp. 487–499.

[10] M. J. Zaki and W. Meira Jr., *Fundamentals of Data Mining Algorithms*. Cambridge University Press, 2011.

[11] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," in *International Conference on Management of Data*, 1993, pp. 207–216.

[12] G. Murphy, D. Notkin, and K. Sullivan, "Software reflexion models: Bridging the gap between source and high-level models," in *3rd Symposium on Foundations of Software Engineering (FSE)*, 1995, pp. 18–28.

[13] S. C. Johnson, "Lint: A C program checker," Bell Laboratories, Tech. Rep. 65, dec 1977.

[14] D. Hovemeyer and W. Pugh, "Finding bugs is easy," *SIGPLAN Notices*, vol. 39, no. 12, pp. 92–106, 2004.

[15] T. Copeland, *PMD Applied*. Centennial Books, 2005.

[16] J. E. Montandon, S. Souza, and M. T. Valente, "A study on the relevance of the warnings reported by Java bug finding tools," *IET Software*, vol. 5, no. 4, pp. 366–374, 2011.

[17] C. Couto, J. E. Montandon, C. Silva, and M. T. Valente, "Static correspondence and correlation between field defects and warnings reported by a bug finding tool," *Software Quality Journal*, vol. 21, no. 2, pp. 241–257, 2013.

[18] Z. Li and Y. Zhou, "PR-Miner: automatically extracting implicit programming rules and detecting violations in large software code," in *13th Symposium on Foundations of Software Engineering (FSE)*, 2005, pp. 306–315.

[19] Y. M. Mileva, A. Wasylkowski, and A. Zeller, "Mining evolution of object usage," in *25th European conference on Object-oriented programming*, 2011, pp. 105–129.

[20] J. Knodel, D. Muthig, M. Naab, and M. Lindvall, "Static evaluation of software architectures," in *10th European Conference on Software Maintenance and Reengineering (CSMR)*, 2006, pp. 279–294.

[21] R. Terra, M. T. Valente, K. Czarnecki, and R. Bigonha, "Recommending refactorings to reverse software architecture erosion," in *16th European Conference on Software Maintenance and Reengineering (CSMR), Early Research Achievements Track*, 2012, pp. 335–340.

Bug Prediction for Fine-Grained Source Code Changes

Zi Yuan
Software Engineering Institute
Beihang University
Beijing, China
yuanzi@sei.buaa.edu.cn

Lili Yu
Software Testing Center
Second Artillery
Beijing, China
xueer-123@263.net

Chao Liu
Software Engineering Institute
Beihang University
Beijing, China
liuchao@buaa.edu.cn

***Abstract*—Software is constructed by a series of changes and each change has a risk of introducing bugs. Building bug prediction models for software changes can help developers know the existence of bugs immediately upon the completion of the change, which allows them to allocate more resources of testing and inspecting on the current risky changes, and to find and fix the introduced bugs timely. In this paper, we present a bug prediction model for fine-grained source code changes based on machine learning method, which takes a fine-grained source code change as a learning instance and a series of properties of the fine-grained change as features. This model has two desirable qualities: 1) Compared with previous research work that building bug prediction models for software changes at the file level or commit level (including one or more files), this model can predict bugs for changes at the statement level, which increases the granularity of prediction and thus reduces manual inspection efforts for developers. 2) This model can help developers or managers gain better knowledge on key factors of bug injection and provide guidance for software change of high quality. From the experiments on 8 famous open source projects, we observe that when using Random Forest as the classifier, the model proposed in this paper achieves the best performance, which can predict bugs for fine-grained source code changes with 78% precision, 71% recall, and 75% F-measure on average. Furthermore, among all the four feature groups (i.e. *where, what, who,* and *when*) defined in this paper, *where* is most influential, which has the strongest discriminative power in predicting bugs.**

Keywords-bug prediction; fine-grained source code changes; code metrics

I. INTRODUCTION

Software bug is the key risk and major cost driver for both companies that develop software and companies that consume software systems in their daily business [1]. Normally, if these bugs are not removed from the software timely, they will affect the software in three aspects negatively. Firstly, bugs that settle in software for a long time may cause new bugs or subsequent error modifications [2]. Secondly, if these bugs are reported late, developers who introduced them must spend time to reacquaint themselves with the changed source code, which will increase the cost of bug fix effort. Thirdly, if these bugs are reported by customers after software release, it will damage the software reputation significantly. In order to remove software bugs timely, we should firstly know the existence of bugs as soon as possible. One effective way is to build bug prediction models for software changes, which could remind developers of bugs immediately upon the completion of a change. In the past few years, several researchers have proposed various bug prediction models for software changes [3], [4]. Most of them built their bug prediction models on file-level or commit-level changes. However, since there are usually multiple statements modified in a committed file or commit, it will be time-consuming to locate bugs exactly.

In this paper, we concentrate on exploring bug introducing changes at a finer granularity and present a bug prediction model for statement-level changes. We formulate bug prediction as a classification problem and take a fine-grained source code change as a learning instance. The properties of the fine-grained source code change are deeply explored and taken as features of the learning instance. There are four aspects considered as constructing features of the prediction model, namely *where* and *when* a fine-grained source code change happens, *what* happens, and *who* manipulates it. Each aspect corresponds to a feature group. We empirically evaluate the performance of the model and the discriminative power of each feature group with 8 famous open source projects. The contributions of our work are summarized as follows:

Building bug prediction model for fine-grained changes: the novel bug prediction model for changes at the statement level increases the granularity of prediction and thus reduces manual inspection efforts for developers. Our experiments show that when using Random Forest as the classifier, the model achieves the best performance. It can predict bugs with 78% precision, 71% recall, and 75% F-measure on average. The results are encouraging, given that the granularity of prediction is smaller than previous research work.

Exploring key factors of bug injection: In addition to building model with high accuracy, we also explore the key factors of bug injection by inspecting into the model. We evaluate the discriminative power of each feature group defined in this bug prediction model with 8 famous open source projects and discover the most influential factors. These factors give guidance for the software change of high quality.

The remainder of this paper is structured as follows. In Section II, we formulate the bug prediction problem, define a series of features considered in building our bug prediction model, and introduce three famous classifiers used in this study. We describe our experiments and evaluations in Section III and discuss the threats to the validity of this study in Section IV. We present work related to this paper in Section V and conclude with possible future work in Section VI.

II. BUG PREDICTION FOR FINE-GRAINED CHANGES

In this section, we firstly present some necessary definitions and a formal representation of the bug prediction problem, and then elaborate the bug prediction model proposed in this paper, including feature definition and classifier introduction.

A. Problem Definition

Fine-Grained Source Code Change (***SCC***): This basic conception is introduced by Fluri et al [5], which denotes source code changes at statement level. They leveraged the implicit structure of source code by comparing two different versions of the abstract syntax tree (AST) of a program and tracked source code changes down to statement level. Formally, we represent a SCC by a triple, which determines it uniquely:

$$SCC = < type, entity, timestamp > \quad (1)$$

where $type$ is from the taxonomy of fine-grained source code changes proposed by Fluri et al. They defined 48 change types with tree edit operations on AST (e.g. condition changes, interface modifications, inserts or deletions of methods and attributes, return type change, etc). The $entity$ denotes the type of the language constructs provided by an object-oriented programming language, such as single and composed statements as well as method or class declarations. It is the object that the SCC acts on directly. There are totally 104 kinds of entities defined in [5]. The $timestamp$ denotes the time when SCC happens. Fig. 1 shows an example of three $SCCs$ based on the AST comparison of three file revisions. One SCC denotes an *if statement insert*, another denotes a *condition expression update* to *if statement*, and the other denotes a *parameter insert operation* on *method declaration*.

Bug Introducing SCC: In this work, each bug corresponds one or more bug reports in the bug database, indicating one or more fixed problems. We consider a SCC as the bug introducing SCC if it leads to at least one bug report and later fix during the software development process. Fig. 1 shows that *if statement insert* is a *bug introducing* SCC.

Indication Labeling: Given a set of fine-grained source code changes $C = (SCC_1, SCC_2, \dots, SCC_n)$, the label of SCC_i $(i = 1, 2, \dots, n)$ is defined as:

$$l(SCC_i) \in \{clean, buggy\} \quad (2)$$

where $clean$ denotes there is no bug introduced by SCC_i, and $buggy$ denotes there is at least one bug introduced by SCC_i.

Learning Task: Given a set of fine-grained source code changes $C = (SCC_1, SCC_2, \dots, SCC_n)$ and their features $F = (f_1, f_2, \dots, f_m)$, our goal is to learn a prediction model BP to predict whether a SCC has bugs or not. Formally, we have:

$$BP(SCC_i | f_1, f_2, \dots, f_m) \rightarrow l(SCC_i),\ i = 1,2, \dots, n \quad (3)$$

To build the prediction model, we have investigated and constructed a series of relevant features $F = (f_1, f_2, \dots, f_m)$ by taking four aspects (i.e. *where*, *what*, *who*, and *when*) into consideration. Each aspect denotes a feature group, namely a subset of F. In this paper, the prediction model is named as *FWL*. Here *FW* denotes the four important feature groups whose names all begin with the letter *W* and *L* denotes the label (i.e. *clean* or *buggy*) of SCC.

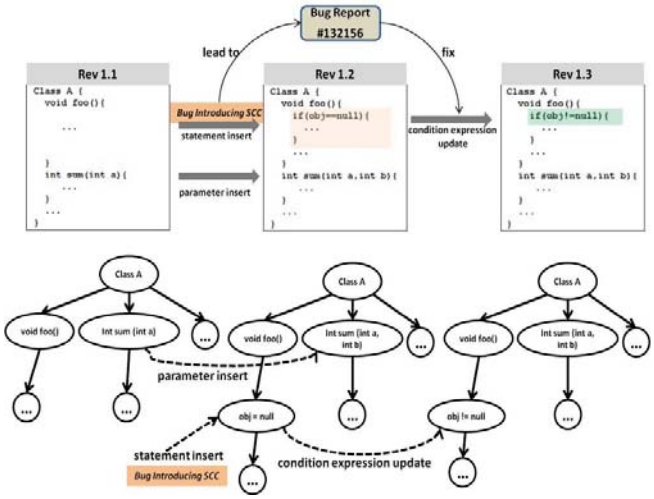


Figure 1. An example of three fine-grained source code changes (i.e. parameter insert, statement insert, and condition expression update) based on the AST comparison of three file revisions as proposed in [5].

B. Feature Definition

In this paper, there are mainly four aspects considered as constructing features of our prediction model *FWL*, namely *where* and *when* a fine-grained source code change happens, *what* happens, and *who* manipulates it. Each aspect corresponds to a feature group. Each feature group consists of one or more sub-feature groups, which is the refinement of it. Fig. 2 shows this two-level structure.

1) ***Feature group where***: In this paper, *where* doesn't indicate the exact location of a fine-grained source code change but the context of it. The characteristics of source code files that have been touched reflect context information of SCC. It is well understood that if it is difficult for a source code file to be understood thoroughly, the risk of changing it will be high. Hence, we assume that the context of *SCC* may correlate with the injection of bugs. A series of metrics reflecting the complexity and activity of the touched file are believed to be related to the difficulty of understanding the file and used as features in bug prediction model *FWL*.

a) Halstead Metrics: *Halstead Metrics* were proposed by Maurice Halstead, who argued that the harder the code to read, the more bug prone the modules are [6]. *Halstead Metrics* are measures of lexical complexity. There are four basic *Halstead Metrics* (i.e. *total number of operators*, *total number of operands*, *unique number of operators* and *unique number of operands*) and two derived *Halstead Metrics* (i.e. *volume* and *programming effort*) considered as features in bug prediction model *FWL*.

b) McCabe Metric: Introduced by Thomas McCabe, the idea behind *McCabe Metrics* is to capture the structural complexity level of a code [7]. The assumption is that it is more likely for the number of bugs to increase as the source code gets more complex. *McCabe Metrics* include *cyclomatic complexity*, *design complexity* and *essential complexity*. Only *cyclomatic complexity* is considered as features in model *FWL*, which is the most famous one among them.

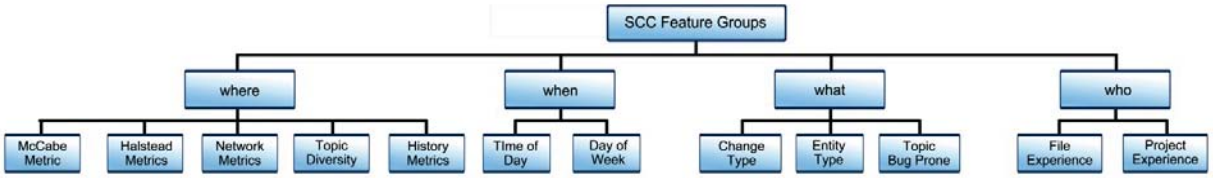


Figure 2. Two-level structure of features groups

c) Network Metrics: The dependency graph of a software system depicts the relational structure between individual source code entities. As indicated by Zimmermann et al [8], dependencies between software entities are crucial for their successful operation. Hence, we assume that the dependency between source code files may correlate with the injection of bugs. In this study, we construct a weighted multiple type dependency graph, in which nodes represent files, edges indicate dependencies between two files, and weight of each edge indicates the amount of dependencies. There are four types of dependencies considered, namely method invocation, field access, inheritance, and implementation. We extract the dependency information of each file touched by SCC and then two *Network Metrics weighted in-degree* and *weighted out-degree* are taken as features in bug prediction model *FWL*.

d) Topic Diversity: Recent studies propose that the domain topics of a software system reflect the business logic of the system and should not be ignored in building bug prediction models [9]. In this study, it is assumed that if there are too many domain topics existing in a file, it will be difficult to understand and change it correctly. *Topic Diversity* indicates domain breadth of source code files, which is measured by the entropy of the file's topic distribution:

$$Diversity(d) = \sum_{k=1}^{|T|} -p(t_k|d) \cdot \log p(t_k|d) \qquad (4)$$

where d denotes a source code file, $T = (t_1, t_2, \dots, t_{|T|})$ denotes the set of topics and p denotes the distribution of topics in a source code file. Since each revision of the source code files should be considered in our study, a topic evolution model DiffLDA [10] that allows the documents to have time-stamps and the corpus to be versioned is used to extract the file's topic distribution.

e) History Metrics: Recently, researchers have shown the usefulness of collecting *History Metrics* from software repositories for bug prediction models [11]. It is assumed that if a source code file has been changed or fixed many times by many developers in the history, the risk of changing it in the future will be high. Hence, three *History Metrics* are considered in model *FWL*, namely *the number of revisions*, *the number of fixes* and *the number of developers* in history.

2) ***Feature group what***: In this paper, feature group *what* has its specific meaning. It mainly denotes the actual content of the SCC, which can be described from three aspects:

a) Change Type: Some researchers [12] have proposed that since the logic complexity of edit operation on AST, some types of fine-grained source code changes (e.g. condition expression change) are more likely to have bugs than others. In this paper, the 48 types of fine-grained changes defined by Fluri et al are considered as building prediction model *FWL*.

b) Entity Type: There are totally 104 types of entities defined in Fluri's work. We assume that entity type can not be ignored in model *FWL* because some types of entities (e.g. if statement) are more difficult to be modified correctly.

c) Topic Bug Prone Metric: As researchers observed, source code responsible for functionality with simple domain logic (e.g. I/O tasks) is likely to have fewer bugs than that responsible for functionality with complicated domain logic (e.g. compiler implementation details) [9]. In this paper, we study the effect of domain topics on quality of fine-grained source code change. We propose a metric to describe the bug proneness of topics that are touched on by SCC.

We firstly define the edits between two successive revisions of the source code file as delta file and propose a metric $ratio$ for each delta file $\delta_j (j = 1,2, \dots, N)$, represented as follows:

$$ratio(\delta_j) = \frac{dirtyLength(\delta_j)}{Length(\delta_j)} \qquad (5)$$

where $dirtyLength$ denotes the number of words in δ_j that are located on the buggy lines. Here, a buggy line means the line of code which have been modified by subsequent bug fix changes. Length denotes the total number of words in δ_j.

Secondly, we use DiffLDA to extract topic distribution of each delta file $\delta_j (j = 1,2, \dots, N)$ and define the *Bug Prone Metric* for each topic t_k $(k = 1,2, \dots, |T|)$ and each SCC_i $(i = 1, 2, \dots, n)$. Formally, we have:

$$bugProne(t_k, SCC_i) = \frac{\sum_{j \in H_{SCC_i}} p(t_k|\delta_j) \cdot ratio(\delta_j)}{|H_{SCC_i}|} \qquad (6)$$

where p denotes the distribution of topics in the delta file and H_{SCC_i} denotes a set of indexes of delta files that correspond to changes that precede SCC_i.

We assume a fine-grained source code change SCC_i $(i = 1, 2, \dots, n)$ touches on some words in the source code file, denoted as $W = (w_1, w_2, \dots, w_p)$, and each word would be assigned to a topic based on DiffLDA. All the topics related to SCC_i are denoted as $T_r = (t_{r_1}, t_{r_2}, \dots, t_{r_q})$. Finally, we get the *Topic Bug Prone Metric* of SCC_i by summing up the *Bug Prone Metric* of each related topics. Formally, we have:

$$topicBugProne(SCC_i) = \sum_{s=1}^{|T_r|} bugProne(t_{r_s}, SCC_i) \qquad (7)$$

The *Topic Bug Prone Metric* can capture unstructured semantic information hidden in source code effectively, which can't be acquired by considering the structure of AST alone.

3) ***Feature group when***: The time when the fine-grained change occurs can usually capture a developer's habit and work cycle (e.g. Some developers always introduce more bugs after midnight). Inspired by the work of Eyolfson [13], we take *Time of Day* and *Day of Week* as two primary features in feature group *when*.

4) ***Feature group who***: Some researchers have proposed that the experience of developers may have effect on the quality of software changes [13], [14]. In this paper, we use the number of times that a developer changes a file and the whole project to describe the developer's experience on the touched file and the whole project and take them as two features *File Experience* and *Project Experience* respectively.

C. Classifiers

There are three classifiers (i.e. Random Forest, Bagging and K-Nearest Neighbor) chosen in this paper, which have been confirmed to have good performances and widely used by other researchers in bug prediction area. The three classifiers are constructed based on different learning mechanisms. Both Random Forest and Bagging belong to ensemble classifiers and K-Nearest Neighbor (KNN) is an instance-based classifier.

III. Experiments and Evaluation

A. Dataset Description

We conduct our study on 8 famous open source projects which are easily accessible to the full source code and entire project history. Table I provides an overview of these projects, including the range of revisions considered, the real world duration of the range, the number of clean and buggy $SCCs$, the percentage of buggy $SCCs$, and the average values for the number of modified statements in a file-level change and in a commit-level change. On average, the number of modified statements in a file-level change is 9, while the number of modified statements in a commit-level change is 27. For example, if a model predicts bugs at the file-level change, it is necessary to inspect 9 statements on average and if it predicts bugs at the commit-level change, it is necessary to inspect 27 statements on average. The bug prediction model *FWL* proposed in this paper helps developers know which modified statement may have bugs exactly, which save their manual inspection efforts significantly.

In this study, there are two criteria considered as choosing the range of revisions: 1) Revisions in the relatively early stage of software development process are chosen, which ensures that bugs introduced in this stage can be found and fixed adequately during subsequent software development process. The adequacy of the bug finding and fixing can further ensure the label quality of SCC. 2) The range of revisions chosen in this work is short considering the lifetime of each software system, which ensures that during the duration of each range, the underlying concept learned by the prediction model is stable. The concept drift problem and incremental learning for the long history of each project are not discussed in this paper, which will be explored deeply in our future work.

B. Data Collection

In our study, the data collection process consists of two parts, namely feature collection process and labeling process (see Fig. 3).

1) ***Feature Collection:*** Feature collection mainly collects features of four aspects mentioned in Section II. For aspect *where*, the *Halstead* and *McCabe Metrics* are collected using a software metrics analysis tool prest[1]. An incremental dependency graph builder[2] developed in our team is used to collect *Network Metrics* for each revision of the touched files. The *Topic Diversity Metric* is collected by following two steps of file's topic distribution extraction and entropy computation. The first step is mainly based on DiffLDA method and implemented by the natural language processing package mallet[3]. Three *History Metrics* are extracted by parsing the change logs of Version Control System (VCS) directly. For aspect *what*, *Change Type* and *Entity Type* are collected by a fine-grained change extraction tool changedistller[4] and *Topic Bug Prone Metric* is extracted also via DiffLDA method and mallet[3] package. In addition, a series of calculations are conducted based on (5), (6) and (7) in Section II. Metrics of aspects *when* and *who* are extracted in a similar manner as collecting History Metrics of aspect *where*.

2) ***Labeling Process:*** In order to label the *SCC* as clean or buggy, we draw lessons from the SZZ algorithm proposed by Sliwerski et al [15], which can automatically identify file-level changes that induce bugs. However, the SZZ algorithm could not be applied directly to identify statement-level changes because of two reasons. One is that annotation by VCS is insufficient for labeling statement-level changes (line number in bug-introducing revisions is missing). And the other is that not all modifications are fixes (e.g. format, comment and blank change). In order to solve the two problems mentioned above, we improve the SZZ algorithm by using a statement tracking tool sDiff[5] instead of the VCS annotation command. Not only can this tool track statements across multiple revisions of source code, but they can also handling format change (e.g. breaking a statement across many lines or re-ordering methods), comment change, and blank change.

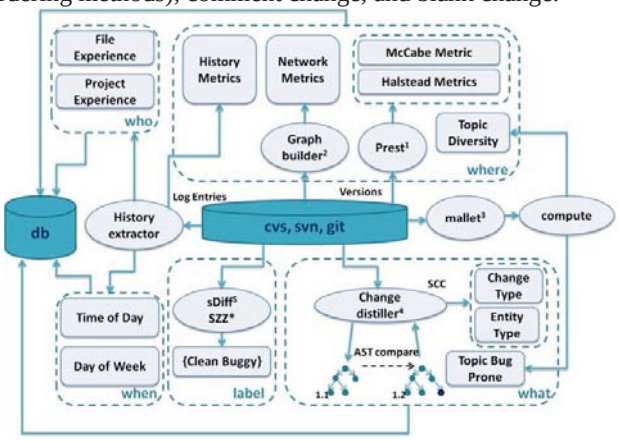


Figure 3. Data collection process

[1]http://svn.cmpe.boun.edu.tr/svn/softlab/prest/trunk/Executable/PrestTool.rar
[2]http://code.google.com/p/dependency-graph-builder
[3]http://mallet.cs.umass.edu/
[4]http://www.ifi.uzh.ch/seal/research/tools/changeDistiller.html
[5]http://code.google.com/p/sdiff

C. Performance and Feature Analysis

1) ***Performance analysis****:* The performances of the bug prediction model *FWL* are evaluated using the 10-fold cross-validation method and three standard evaluation measures (i.e. precision, recall, and F-measure). The results are summarized in Table II. Different classifiers have different performances in our experiments. In general, Random Forest achieves the best performance, which has the buggy *SCC* recall ranging from 62.2 to 93.0 percent and the buggy *SCC* precision ranging from 71.5 to 93.1 percent. While, KNN has the worst performance, which has the buggy *SCC* recall ranging from 53.5 to 92.6 percent and the buggy *SCC* precision ranging from 61.0 to 91.8 percent. Random Forest has the best performance because it is more robust to noise that exists in our dataset. KNN has the worst performance because it merely seeks the most similar neighbors, which only utilizes little information from training data. As a whole, all the three classifiers achieve encouraging performances, given that the granularity of prediction is smaller than previous research work. In order to further validate the effectiveness of the bug prediction model *FWL*, we take a dummy model constructed by random guessing as the baseline. Fig. 4 shows that the model *FWL* always performs much better than the dummy model, whichever of the three classifiers is used.

2) ***Feature group analysis****:* In this study, we examine the discriminative power of each feature group in the first abstraction level (i.e. *where*, *what*, *who*, and *when*) following two steps: First, the Random Forest Classifier that has the best performance in our study is trained using features from one feature group and then its precision, recall, and F-measure are measured. Following this, Random Forest is trained using all feature groups except for the feature group using in the first step and also evaluated by precision, recall, and F-measure.

Fig. 5 shows the F-measure collected by running the bug prediction model (constructed by using only one feature group each time) on 8 open source projects. On the whole, feature group *where* is proved to have the strongest discriminative power in predicting bugs. It is understandable that bugs are likely to be introduced into certain source code files which have some characteristics that make them difficult to be changed correctly. Feature group *what* and *who* are also significant and they are well-matched in discriminative power. It means that the actual content of a fine-grained source code change and the experience of the developer who implements the change are related to the injection of bugs. Unexpectedly, the discriminative power of feature group *when* is weak. The cause for its weakness can be explained that in the 8 open source projects used in this paper, developers' habit and work cycle have no significant influence on the injection of bugs. It is desirable that all the 8 open source projects have consistent performances with the feature group *where* having the strongest discriminative power, *what* tied with *who* taking the second place and *when* having the weakest discriminative power. The results can reveal the important factors of bug injection and provide guidance for developers.

Fig. 6 summarizes the F-measure of the 8 open source projects collected by running the bug prediction model with the corresponding feature group excluded from full feature combination. The "~" mark indicates that the corresponding feature group is excluded. For comparison, Fig. 6 also shows the F-measure using full feature combination. It can be observed that when the feature group *what* is left out from full feature combination, we get the lowest F-measure and when all the feature groups are used, the highest F-measure is reached.

TABLE I. SUMMARY OF STUDIED PROJECTS

Project	Revisions	Period	# of clean *SCC*	# of buggy *SCC*	% of buggy *SCC*	Average # of modified statements in a file-level change	Average # of modified statements in a commit-level change
Eclipse JDT Core	1250-1500	12/2001-01/2002	1730	249	13	6	11
Eclipse UI Workbench	2750-3000	11/2003-12/2003	2384	415	15	8	25
JEdit	750-1000	11/2001-05/2002	6947	1774	20	9	39
Columba	1000-1250	06/2003-07/2003	2572	318	11	7	28
Argouml	2500-2750	01/2001-09/2001	3698	847	19	11	39
Scarab	500-1000	05/2001-08/2001	1613	537	25	7	19
JBoss	3750-4000	03/2002-05/2002	2173	332	13	15	34
Struts	2000-2500	06/2002-10/2002	1698	781	32	11	21

TABLE II. THE PERFORMANCE OF THE THREE CLASSIFIERS FOR FULL FEATURE COMBINATION ON 8 OPEN SOURCE PROJECTS

	Random Forest			Bagging			KNN		
Project	*Precision*	*Recall*	*F-measure*	*Precision*	*Recall*	*F-measure*	*Precision*	*Recall*	*F-measure*
Eclipse JDT Core	0.749	0.622	0.680	0.668	0.631	0.649	0.676	0.586	0.628
Eclipse UI Workbench	0.715	0.622	0.665	0.654	0.598	0.625	0.610	0.535	0.567
JEdit	0.742	0.707	0.724	0.702	0.678	0.690	0.677	0.641	0.658
Columba	0.793	0.686	0.735	0.721	0.642	0.679	0.616	0.616	0.616
Argouml	0.733	0.653	0.691	0.796	0.527	0.634	0.707	0.633	0.668
Scarab	0.788	0.711	0.748	0.737	0.695	0.715	0.762	0.709	0.735
JBoss	0.827	0.750	0.787	0.785	0.726	0.754	0.755	0.726	0.740
Struts	0.931	0.930	0.930	0.936	0.905	0.921	0.918	0.926	0.922

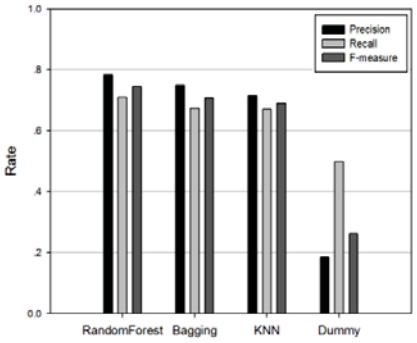


Figure 4. the average performance collected by running the dummy model and the bug prediction model *FWL* (with each of the three classifiers) on 8 open source projects.

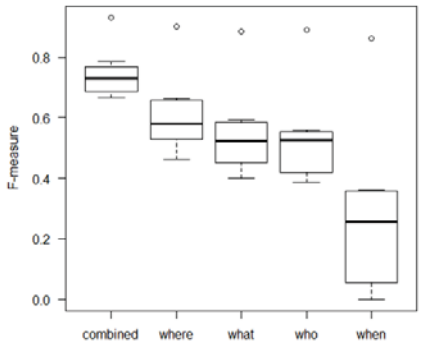


Figure 5. the F-measure collected by running the bug prediction model (constructed by using only one feature group each time) on 8 open source projects.

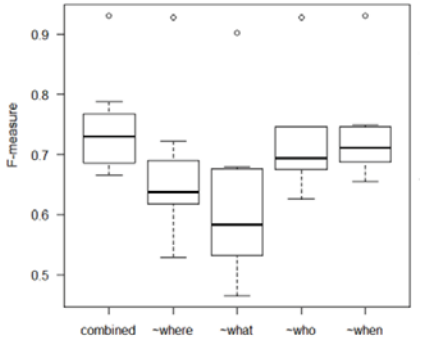


Figure 6. the F-measure collected by running the bug prediction model (with the corresponding feature group excluded from the full feature combination) on 8 open source projects.

3) ***Sub-feature group analysis****:* We further examine the discriminative power of each sub-feature group in the second abstraction level following the similar steps as used in the first level and the results are presented in Fig. 7. It can be observed that the absence of *Topic Bug Prone Metric* leads to the most prominent decrease of performance. *Topic Diversity*, *Halstead* and *Topic Bug Prone* are the three most powerful prediction factors and followed by *Project Experience* and *McCabe*. Two of the least powerful factors are *Time of Day* and *Change Type*. Other sub-feature groups have the moderate performances. The results indicate us that the information of business domain topics which denotes the semantic functionalities of source code should not be ignored in exploring the problem of bug injection and the lexical complexity is also a key factor that leads to bugs. It is easily explained that if a source code file has complex lexical and semantic structure, it will be difficult for developers to understand it thoroughly. Hence the risk of changing it will be high. In addition, developers' familiarity with the whole project, to a certain extent determines the quality of software changes. These results obtained by sub-feature group analysis are very valuable. It can help developers gain better knowledge on internal reasons of bug injection and provide useful guidance for their further improvement of software quality.

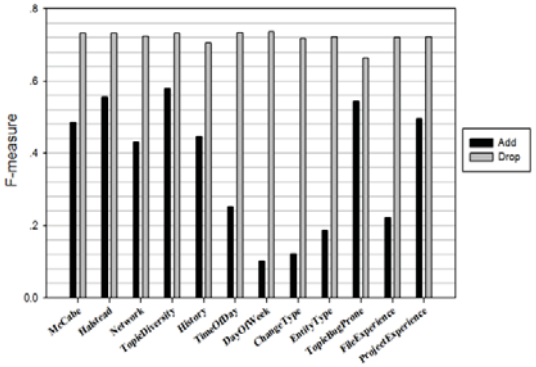


Figure 7. Performance comparison for sub-feature group analysis. Here, the F-measure is an average value on the 8 projects. "Add" denotes the model is trained with the sub-feature group alone and "Drop" denotes the corresponding sub-feature group is excluded from the full feature combination.

IV. THREADS TO VALIDITY

A. The Construct Validity

How accurate we measure a particular concept, is mainly threatened by the bug introducing *SCC* identification process (i.e. labeling process).

First, we identify the bug fix changes by searching for references to bug reports and keywords (e.g. "bug", "fix", "PR", "issue" etc) in VCS commit messages. This method is only reliable as projects have change logs with good quality (i.e. most of references are manually recorded when committing). However, analyzing commit messages to identify the bug fix changes is a common procedure and does also reflect state of the art [15], [16].

Second, we track the buggy introducing *SCCs* based on the SZZ method and improve it by replacing the text comparison with statement comparison, which could track and label changed statements across multiple revisions of source code and reduce some kinds of noises (e.g. format change, comment and blank change). However, there are still two problems in this step remained to be solved in our future work. One is that the bug fix and bug introducing changes are at different locations. The other is that some bug fix changes only add several lines of code, which are missed by current approaches.

B. The External Validity

Only projects following the open source development methodology have been examined in our study. It is possible that the stronger deadline pressure, different personnel turnover patterns, and different development processes used in industry development could lead to different bug introducing *SCC* patterns. Nevertheless, all projects are independently developed and come from different domains. Moreover, although open source, the two projects from eclipse community (i.e. Eclipse JDT Core and Eclipse UI Workbench) have the industrial background.

V. RELATED WORK

Currently, several researchers have been exploring bug introducing software changes and some of them have built models to predict bugs for software changes.

Sliwerski et al [15] examined several properties of bug introducing changes in Eclipse and Mozilla projects and found that the larger a change was, the more likely it was buggy. He also observed that changes on Fridays were buggiest. Eyolfson et al [13] also discussed the time of change, and investigated how the changes' time of day correlates with the changes' bugginess. In addition, they studied developer characteristics, such as change frequency and experience, and found they also correlated with the bugginess of changes. Rahman et al [14] considered the impact of code ownership and developer experience on bugginess of software changes. They found that buggy change was more strongly associated with a single developer's contribution and developer's specific experience in the target file was more important than general experience.

Mockus et al [4] focused on several properties of software changes and built a logistic model to predict the probability that a change will cause a bug. The properties included size in lines of code added, deleted, and unmodified; diffusion of the change; several measures of developer experience; and the type of change. The change in their work is defined at the commit level. Kim et al [3] took the bug prediction for software changes as a classification problem. The features used by the classifier mainly included terms in the source code, the size in lines of code modified in each change, and change meta-data such as author and change time. In their paper, change is at the committed file level.

Different from the related work mentioned above, we build our bug prediction model for the fine-grained source code change at statement level, which increases the granularity of the prediction and thus reduces manual inspection efforts.

VI. Conclusions and Future Work

In this paper we present a novel bug prediction model *FWL* for fine-grained source code changes, which can predict the existence of bugs in source code changes at the statement level. We formally formulate this bug prediction task as a classification problem and build our prediction model utilizing a series of relevant features and three different classifiers. From our experiments on 8 open source projects, our model using full feature combination and Random Forest Classifier achieves the best performance. It can predict bugs with 78% precision, 71% recall, and 75% F-measure on average. Given the fine-granularity, this result is encouraging. We also examine the discriminative power of each feature group and sub-feature group defined in this paper. For feature groups at the first abstraction level, we have observed that feature group *where* is the most influential aspect, which has the strongest discriminative power in predicting bugs; feature group *when* has the weakest discriminative power and feature groups *what* and *who* have moderate performances. For sub-feature groups defined at the second level, *Topic Diversity*, *Halstead* and *Topic Bug Prone* are proved to be the three most powerful prediction factors and followed by *Project Experience* and *McCabe*. Two of the least powerful factors are *Time of Day* and *Change Type* respectively. This work is the first to predict bugs in source code changes at the statement level.

Although these experimental results are encouraging, there are still several work remained to be done in the future, including the following:

- Exploring more sophisticated SCC labeling algorithm, which can handle the problem that bug fix and bug introducing changes are at different locations by dint of control dependency, data dependency and logic coupling dependency.
- Exploring the variation of influencing features with the passing of time and construct incremental learning algorithms to learn and update the bug prediction model as the project progresses.
- Generating more new features for feature group *when*, which is not restricted to the absolute time. Several relative time metrics (e.g. after large-scale refactoring or during the period of change bursts) will be considered as features in our future work.

References

[1] E. Giger, M. Pinzger, and H. Gall, "Comparing fine-grained source code changes and code churn for bug prediction", in Proc. MSR, 2011, pp.83-92.

[2] Q. Song, M.J. Shepperd, M. Cartwright, and C. Mair, "Software Defect Association Mining and Defect Correction Effort Prediction", IEEE Transactions on Software Engineering, vol. 32, pp.69-82, 2006.

[3] S. Kim, E.J.W. Jr., and Y. Zhang, "Classifying Software Changes: Clean or Buggy?", IEEE Transactions on Software Engineering, vol. 34, pp.181-196, 2008.

[4] A. Mockus and D.M. Weiss, "Predicting risk of software changes", Bell Labs Technical Journal, vol. 5, pp.169-180, 2000.

[5] B. Fluri, M. Würsch, M. Pinzger, and H. Gall, "Change Distilling: Tree Differencing for Fine-Grained Source Code Change Extraction", IEEE Transactions on Software Engineering, vol. 33, pp.725-743, 2007.

[6] M. H. Halstead, Elements of Software Science. Elsevier, 1977.

[7] T.J. McCabe, "A Complexity Measure", IEEE Transactions on Software Engineering, vol. 4, pp.308-320, 1976.

[8] T. Zimmermann and N. Nagappan, "Predicting Subsystem Failures using Dependency Graph Complexities", in Proc. ISSRE, 2007, pp.227-236.

[9] T. Chen, S.W. Thomas, M. Nagappan, and A.E. Hassan, "Explaining software defects using topic models", in Proc. MSR, 2012, pp.189-198.

[10] S.W. Thomas, B. Adams, A.E. Hassan, and D. Blostein, "Modeling the evolution of topics in source code histories", in Proc. MSR, 2011, pp.173-182.

[11] T.L. Graves, A.F. Karr, J.S. Marron, and H.P. Siy, "Predicting Fault Incidence Using Software Change History", IEEE Transactions on Software Engineering, vol. 26, pp.653-661, 2000.

[12] K. Pan, S. Kim, and E.J.W. Jr., "Toward an understanding of bug fix patterns", Empirical Software Engineering, vol. 14, pp.286-315, 2009.

[13] J. Eyolfson, L. Tan, and P. Lam, "Do time of day and developer experience affect commit bugginess", in Proc. MSR, 2011, pp.153-162..

[14] F. Rahman and P.T. Devanbu, "Ownership, experience and defects: a fine-grained study of authorship", in Proc. ICSE, 2011, pp.491-500.

[15] J. Sliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?", ACM sigsoft software engineering notes, vol. 30, pp.1-5, 2005.

[16] M. D'Ambros, M. Lanza, and R. Robbes, "Evaluating defect prediction approaches: a benchmark and an extensive comparison", Empirical Software Engineering, vol. 17, pp.531-577, 2012.

An efficient QCL-based alert correlation process

Lydia Bouzar-Benlabiod
LCSI laboratory
Ecole nationale Supérieure d'Informatique (ESI)
Algiers, Algeria
l_bouzar@esi.dz

Salem Benferhat
CRIL-CNRS
Université d'Artois
Lens, France
benferhat@cril.univ-artois.fr

Thouraya Bouabana-Tebibel
LCSI laboratory
Ecole nationale Supérieure d'Informatique (ESI)
Algiers, Algeria
t_tebibel@esi.dz

***Abstract*—Intrusion Detection Systems (IDS) are important tools for network monitoring. However, they produce a large quantity of alerts. The security operator that analyses IDS alerts is quickly overwhelmed. Alert correlation is a process applied to the IDS alerts in order to reduce their number. In this paper, we propose a new approach for logical based alert correlation which integrates the security operator's knowledge and preferences. The goal is to present to the security operator only the most suitable alerts. The representation and the reasoning on these knowledge and preferences are done using a new logic called Instantiated First Order Qualitative Choice Logic (IFO-QCL). Our algorithm performs the correlation process in a polynomial time. Experimentation are achieved on data collected from a real system monitoring. The result is a set of stratified alerts satisfying the operators criteria.**

***Keywords*-IDS, alert correlation, QCL, preferences, knowledge**

I. Introduction

Intrusion Detection Systems (IDS) [1] [2] are important network security tools. They filter traffic data searching for malicious activities. A security operator has to supervise log alerts emanating from different IDS. When the data traffic is important and the number of suspicious activities is high the security operator is quickly submerged by a huge number of IDS alerts [3]. Alert correlation is a process that aims to reduce the produced alerts number. It analyses IDS alerts, gathers them into groups or clusters of similar alerts [4] and provides intrusion reports to security operators.

Different alert correlation methods have been proposed in the literature. We distinguish four main approaches: (1) Alert correlation using alerts' attributes similarity. (2) Alert correlation approaches based on pre-defined attack scenarios. (3) Correlation methods based on preconditions and postconditions. (4) Data mining approaches to alert correlation.

A new approach to alert correlation has been proposed in [15]. This method uses an extension of the preferences logic, called Qualitative Choice Logic (QCL) [16], to integrate the security operator knowledge and preferences resulting from his experience, to the correlation process. QCL increases the propositional logic language with the ordered disjunction operator:"$\times$". This operator expresses preference information. A$\times$B means that "if possible A, otherwise at least B". Our aim is to define a new method for alert correlation which consists of inserting a filter created by the operator to the alert correlation process. The filter is based on the operator knowledge and preferences related to the monitored system, as well as his experience. The output of our model is a set of alerts that satisfy the security operator's criteria. Our proposed algorithm is based on an extension of QCL language. This paper contains three important contributions.

- We first propose a framework for dealing with preferences using IFO-QCL language.
  Our framework is convenient for representing knowledge and preferences on alerts.
- We then propose an algorithm for alerts correlation that takes into account the security operator knowledge and preferences. We simply view an alert as an interpretation. Hence the satisfaction of an interpretation, with respect to the security operator preferences, directly induces a ranking of alerts.
- Lastly, we present experimental studies on real alerts issued from PLACID[1] project.

The paper is organized as follows: Section II briefly describes existing alert correlation approaches. Section III and IV detail our alert correlation model. Section V gives the main steps of our approach. Section VI presents our experimentation results. Section VII concludes the paper.

II. Related works

Alert correlation is a multi-step transformation process. The inputs are IDS alerts. Alerts are analyzed, gathered and then returned to the operator as an intrusion report. One of the main alert correlation objectives is the reduction of the number of alerts. Among alert correlation methods, we can distinguish:

- Correlation approaches based on similarity between alerts' attributes [5][6][7] and alert aggregation mechanisms [17] [18] These approaches are based on the fact that alerts belonging to the same attack have similar

[1] http://placid.insa-rouen.fr/

attribute values (source IP address, etc.). An alert-filter is set up in order to determine the attribute values similarities.

- Correlation by pre-defined attack scenarios [8][9][10]. It is an explicit correlation approach whose objective is to detect complex attack scenarios. These scenarios are stored in signature bases. The bases can be either obtained from historical data or specified by users.
- Preconditions and postconditions alert correlation methods [11] [12] [13]. Each action is associated with a set of preconditions (needed for launching an action), and a set of postconditions representing consequences of the action. These methods correlate alerts if post-conditions of present alerts may be correlated with preconditions of some future alerts.
- Alert correlation as a classification process [14] [19] [20]. An example of used classifiers tools are the Bayesian networks ones. Bayesian classifiers are graphical models that allow efficient treatment of uncertain information. In [14], each intrusion objective is modeled by a naive Bayes classifier which is obtained from historical data. The resulting model is then used for the prediction of the intrusion objectives.

In addition to these approaches, a correlation alert method integrating security operator knowledge and preferences was proposed in [15]. This approach is based on a logic of preferences called Qualitative Choice Logic (QCL)[16]. This logic, and its extensions, have been used to represent security operator knowledge and preferences. In [15], first order logic predicate, denoted by "show-to-operator (x)", has been introduced to indicate whether an alert x should be presented to a security operator or not. The main problem with this approach is its high computational complexity, since the decision problem: "should an alert be presented to the security operator" is at least a Δ_p^2 complete problem.

Compared to the previous approach, we also propose an approach based on expert knowledge and preferences. We use a different modeling logical language from the one used in [15]. This language is more flexible than propositional QCL [16], and less expressive than the FO-QCL used in [15]. The defined logic is called Instantiated First Order Qualitative Choice Logic (IFO-QCL) and is more appropriate for the purpose of our application. Contrarily to the modeling proposed in [15], our approach for alerts correlation can be achieved in a polynomial time. The following section presents this language.

III. Instantiated First Order Qualitative Choice Logic IFO-QCL

QCL [16] is a compact logic for representing and dealing with simple and complex knowledge and preferences. QCL increases the expressive power of the propositional logic with a new connective operator called ordered disjunction operator denoted by: $\times$. This operator expresses users' preferences. QCL allows the preference expression on every kind of variable. It has several extensions: (Minimal QCL "MQCL" [15], positive QCL "QCL+" [15], prioritized QCL "PQCL" [15]). QCL is defined within a propositional language whereas its extensions are defined within a first order language.

The extension to full first order logic language is not needed for alert correlation. As we will show, a small extension (at least from the representation point of view) of propositional logic is enough. More precisely, we propose to use Instantiated First Order QCL (IFO-QCL).

In the following, we will present our language, which is in fact built over three encapsulated languages:(1) IFO formulas which will only be used to express knowledge or pieces of information that each alert should satisfy in order to be considered by a security operator. (2) IFO-BCF formulas (BCF for Basic Choice Formulas) which will be used to express simple forms of preferences. (3) IFO-GCF formulas (GCF for General Choice Formulas) which will be used to express general preferences. IFO (resp IFO-BCF and IFO-GCF) formulas extend propositional formulas (resp BCF and GCF formulas) used in QCL. Our framework is parameterized with a normalized function that transforms any IFO-GCF formula to IFO-BCF one, this transformation can be achieved in a polynomial time.

The vocabulary used to define IFO formulas and IFO-BCF formulas is defined by:

- A set of constant symbols: $\{c_1, c_2, \ldots, c_n\}$.
- A set of predicate symbols:P $\{P_1, P_2, \ldots, P_n\}$. Each predicate has an arity m representing the number of arguments it can support.
- A set of propositional operators $\neg, \vee, \wedge$ and new connector called ordered disjunction $\times$.
- The usual separators (,).

In our application, the predicate symbols will represent alerts' attributes such as kinds of used IDS, operating systems, etc. Constants represent instances of alerts' attributes.

A. Instantiated First Order (IFO) formulas

Let P be a set of predicate symbols. The following defines the language composed of instantiated first order formulas

- (1) if P is a predicate symbol of arity m and $\{c_1, c_2, \ldots, c_m\}$ is a set of constants then P($c_1, c_2, \ldots, c_m$) is an IFO formulas.
- (2) if P and Q are IFO formulas then (P$\wedge$Q), (P$\vee$Q), ($\neg$P) are IFO formulas.
- IFO formulas are only obtained by applying items (1) and (2) a finite number of times.

In fact, IFO language is composed of instantiated first ordered formulas where all terms are constants. The following extended languages allow the representation of simple and complex preferences (respectively IFO-BCF, IFO-GCF).

B. IFO Basic Choice Formulas (IFO-BCF)

IFO Basic Choice Formulas (IFO-BCF) are ordered disjunctions of IFO formulas. They offer a simple way to order available alternatives. The language composed of IFO-BCF formulas is defined as follow :

- If ϕ is a IFO formula then ϕ is a IFO-BCF.
- If P and Q are two IFO-BCF formulas then P$\times$Q is a IFO-BCF.
- Every IFO basic choice formula is only obtained by applying the two rules above a finite number of times.

Intuitively, P$\times$Q means "satisfy either P or Q with a preference to P". Namely, solutions where P is true are preferred to the ones where $\neg$ P and Q are true. The language of IFO-BCF formulas will be simply denoted by IFO_{BCF}.

C. IFO General Choice Formulas (IFO-GCF)

IFO General Choice Formulas (IFO-GCF) allow to represent general forms of formulas. They are defined as follow :

- If ϕ is a IFO-BCF then ϕ is a IFO-GCF.
- If P and Q are two IFO-GCF then (P$\wedge$Q), (P$\vee$Q), ($\neg$P), (P$\times$Q) are a IFO-GCF.
- Every IFO general choice formula is only obtained by applying the two rules above a finite number of times. As we will see later, every IFO-GCF can be reduced to an IFO-BCF by some normalization function "f".

The language of IFO-GCF formulas, will be refered to by IFO_{GCF}.

Suppose that a security operator first prefers analyzing IDS alerts which have a 'high' 'severity' value. He, afterwards, chooses to analyze alerts having the 'medium' value and the last option is to analyze 'low' severity alerts. This preference is expressed using IFO_{BCF} as follows:

$$Severity(high) \times Severity(medium) \times Severity(low)$$

The predicate symbol is 'Severity' and the constant symbols are: "high", "medium" and "low".

We can represent the above preference using QCL[16] and MQCL[15]

1) Using QCL, We define three propositional variables: "high-severity-alert", "medium-severity-alert" and "low-severity-alert". Then we write the preference formula: high-severity-alert $\times$ medium-severity-alert $\times$ low-severity-alert.
2) Using MQCL, let define three propositional variables x, y, z, representing alerts, and four predicate symbols "severity-high", "medium-severity", "low severity" and "show-to-operator". The preference formula is written as follows: $\forall$ x, y, z (severity-high(x) $\wedge$ show-to-operator(x))$\times$ (medium-severity(y)$\wedge$ show-to-operator(y)) $\times$ (low-severity(z)$\wedge$ show-to-operator(z)).

D. Normalization function: From IFO_{GCF} to IFO_{BCF}

Normalization functions transform instantiated first order general choice formulas to instantiated first order basic choice formulas. The advantage of such a transformation is to reuse efficient definitions of satisfaction degree of IFO-BCF for IFO-GCF.

$f_{IFO-QCL}$ is the normalization function which transforms every IFO-GCF formula to an IFO-BCF one. This normalization function considers that all preferences have the same level of importance. It is defined using the following three items:

1. If ϕ is a IFO-BCF then $f_{IFO-QCL}(\phi) = \phi$.
2. $f_{IFO-QCL}$ is decomposable with respect to negation, conjunction, disjunction and ordered disjunction. If ϕ and φ are two IFO-GCF (but either ϕ or φ is not IFO-BCF) then:
   - (a) $f_{IFO-QCL}(\phi \wedge \varphi) \equiv f_{IFO-QCL}(f_{IFO-QCL}(\phi) \wedge f_{IFO-QCL}(\varphi))$
   - (b) $f_{IFO-QCL}(\phi \vee \varphi) \equiv f_{IFO-QCL}(f_{IFO-QCL}(\phi \vee f_{IFO-QCL}(\varphi))$
   - (c) $f_{IFO-QCL}(\phi \times \varphi) \equiv f_{IFO-QCL}(f_{IFO-QCL}(\phi) \times f_{IFO-QCL}(\varphi))$
   - (d) $f_{IFO-QCL}(\neg\phi) \equiv f_{IFO-QCL}(\neg f_{IFO-QCL}(\phi))$
3. Let $\phi = a_1 \times a_2 \times \ldots \times a_n$ and $\varphi = b_1 \times b_2 \times \ldots \times b_m$ be two IFO-BCF formulas.
   - (a) $f_{IFO-QCL}(\phi \wedge \varphi) \equiv c_1 \times c_2 \times \ldots \times c_k$ $c_i = \bigvee(a_j \wedge b_l)$ with $1 \leq j \leq n, 1 \leq l \leq m$ and $j + l = i + 1$.
   - (b) $f_{IFO-QCL}(\phi \vee \varphi) \equiv c_1 \times c_2 \times \ldots \times c_k$ where k=max (m,n) and $c_i = \begin{cases} a_i \vee b_i \text{ if } i \leq min(m,n). \\ a_i \text{ if } m \prec i \leq n. \\ b_i \text{ if } n \prec i \leq m. \end{cases}$
   - (c) $f_{IFO-QCL}(\neg\phi) \equiv \neg a_1 \times \neg a_2 \times \ldots \times \neg a_n$

Property "1" describes the fact that the normal form of an IFO-BCF ϕ is the IFO-BCF ϕ. Property "2" (from(a) to (d)) expresses that the normalization function is decomposable with respect to negation, conjunction, disjunction and ordered disjunction. Property "3" (from (a) to (c)) defines the conjunction, disjunction and negation of IFO-BCF formulas.

Let ϕ: OS(Unix) $\times$ OS(Mac-OS) $\times$ OS(Windows) and φ: Alert-Source(External) $\times$ Alert-Source(Internal), be two IFO-BCFs.

$f_{IFO-QCL}(\phi \wedge \varphi) \equiv$ (OS (Unix) $\wedge$ Alert-Source (External)) $\times$ ((OS(Unix) $\wedge$ Alert-Source (Internal)) $\vee$ (OS (Mac-OS) $\wedge$ Alert-Source (External))) $\times$ (OS (Mac-OS) $\wedge$ Alert-Source (Internal)) $\vee$ (OS (Windows) $\wedge$ Alert-Source (External)) $\times$ (OS (Windows) $\wedge$ Alert-Source (Internal)).

IV. Semantics and satisfaction degrees

This section defines the concept of interpretation in our framework and attributes a satisfaction degree to each preference with respect to an interpretation.

A. IFO-QCL interpretation

An interpretation is a pair I=(D,I_v) where D is the domain of the interpretation and I_v is a function that assigns to each constant C an element of D, and to each predicate symbol P (of arity n) a subset of D^n. Intuitively, I_v(P) represents the set of all n-tuples that make P true in I. Let I_1 =(D,I_v) be an interpretation with D={ Unix, Mac-OS, Windows } and OS be a predicate symbol with arity 1. I_v(OS)=Unix intuitively means that within the interpretation I_1 the Operating System OS has the value UNIX.

B. Satisfaction relation in IFO-QCL

Let I=(D, I_v) be an interpretation, P and Q be two predicates and $\{c_1, c_2, \ldots, c_m\}$ be a set of constants. Let ϕ and φ be two IFO formulas.
We now define the concept of satisfaction of each interpretation with respect to a given IFO-BCF and IFO formula.
This satisfaction relation will be denoted by :$I \models_k \phi$ where I=(D,I_v) is an interpretation,ϕ an IFO or an IFO-BCF formula and k is a positive integer that represents the satisfaction degree of the formula ϕ in the interpretation I.
Let us start with the case of IFO formulas that do not involve the preference operator $\times$. In this case k can only be either equal to 1 or equal to 0.

C. Satisfaction of IFO formulas

If P is a predicate symbol of arity m and $\{c_1, c_2, \ldots, c_m\}$ are a set of constant then:

- $I \models_1 P\{c_1, c_2, \ldots, c_m\}$ iff $\{I_v(c_1), I_v(c_2), \ldots, I_v(c_m)\} \in I_v(P)$
  and $I \models_0 P\{c_1, c_2, \ldots, c_m\}$ iff $\{I_v(c_1), I_v(c_2), \ldots, I_v(c_m)\} \notin I_v(P)$
- $I \models_k \phi \wedge \varphi$ iff $I \models_i \phi$ and $I \models_j \varphi$ and $k = \max(i, j)$.
- $I \models_k \phi \vee \varphi$ iff $I \models_i \phi$ and $I \models_j \varphi$ and $k = \min(i, j)$.
- $I \models_k \neg\phi$ iff $I \models_{1-k} \phi$

In the following, for IFO formulas, we simply write $I \not\models \phi$ iff $I \models_0 \phi$, and $I \models \phi$ iff $I \models_1 \phi$.

D. Satisfaction of IFO-BCF formulas

The satisfaction degree of an IFO-BCF formula can be greater than 1 since an IFO-BCF involves different choices or options. More precisely, if $\phi = \varphi_1 \times \varphi_2 \times \ldots \times \varphi_n$ is an IFO-BCF formula (φ_i's are IFO formulas) then
$I \models_k \varphi_1 \times \varphi_2 \times \ldots \times \varphi_n$ iff $\exists j$, such that $I \models \varphi_j$ and $I \not\models \varphi_i$ for all $i \prec j$.
When $\forall \varphi_i, I \not\models \varphi_i$ then we simply write $I \not\models \phi$
Let Severity and IDS-Type be two predicates, I_1 =(D, I_{v1}) and I_2 =(D, I_{v2}) be two interpretations such that D={ high, medium, low, snort, Brother}, I_{v1}(IDS-Type) = { Snort}, I_{v1}(Severity) = { high}, I_{v2}(IDS-Type) = { Bro} and I_{v2}(Severity) = { Low}.
Let ϕ and φ be two preferences formulas: ϕ_1 Severity (high)$\wedge$ IDS-Type(snort)$\times$ Severity (low)$\wedge$ IDS-Tye(snort). ϕ_2 IDS-Type(snort)$\times$ IDS-Type(bro).

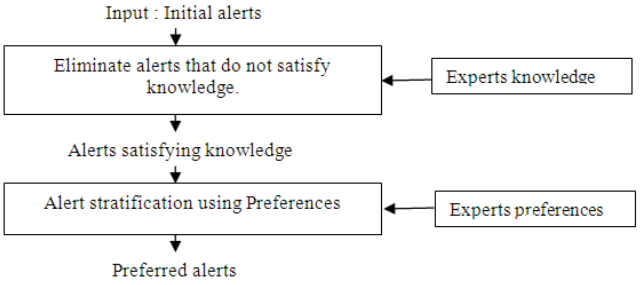


Figure 1. Main steps of the alert correlation process

$I_1 \models_1 \phi_1$, $I_1 \models_1 \phi_2$, $I_2 \not\models \phi_1$, $I_2 \models_2 \phi_2$.

In the proposed model, we rely on the reasonable assumption that the input alerts are viewed as interpretations for the IFO-QCL model. An alert is composed of a set of attributes. The list of predicates P corresponds to the list of attributes that are present in alerts plus the predicates that are present either in knowledge or preferences bases. In order to avoid heavy notations we will use the same symbols for attributes and their associated predicate names.
The list of constants C is the set of all possible values that alert attributes or predicates, used by the security operator, may take.
In the rest of this paper we will assume that the domain associated with each interpretation I=(D,I_v) is the set of constants C. We also assume that for each constant c we have:$\forall$ c $\in$ D_c, I_v(c)=c. This is in the spirit of what it usually known as herbrand models [22].
Let A be an alert. I^A is the interpretation associated with A. If q is an attribute accepting n values, and x is a value of q in the alert A, then I_v^A(q)=x.

V. Main steps of the proposed approach

The inputs, outputs and steps of our method are enumerated below and illustrated in Figure. 1.

A. Set of alerts A

The input of our alerts correlation process is a set of alerts A issued from different IDS. Alerts are provided in IDMEF format (International Data Model Exchange Format) [21]. An IDMEF alert contains information about alerts such as: source, target, analyzers, alert detection time, etc. In our model, an alert will be represented as an interpretation where the domain of each attribute is simply the list of the different attribute values present in the alert. Each alert attribute is in predicate form, for instance: Severity(medium), AnalyzerID(27896), AnalyzerName(prelude-lml).

B. A knowledge base K

This knowledge base encodes the security operator's knowledge. It is a set of IFO formulas. The knowledge

formulas detail the attributes values that the operator does not want to see in the resulting alert set. An example of a knowledge formula is: OS (Windows)∧Classification (http-Inspect).
It encodes the fact that if the Operating System of the machine is "Windows" then the security operator refuses to consider the "http-inspect" alerts.

C. A set of Preferences P

It is a set of the security operator's preferences encoded using IFO-GCF formulas. An example of preference can be: (AnalyzerName(snort) ∧ (Severity(high)∨ Severity (medium))) × (AnalyzerName(snort) ∧ Severity(low)).
This formula means that: "snort" alerts which have a high or a medium severity are preferred to low severity "snort" alerts.

D. Output

The output of our application is a subset of the initial alerts filtered by the knowledge and preferences sets. The resulting alerts are stratified in accordance with the security operator's knowledge and preferences.

E. Algorithm

The inputs are: the set of alerts, the knowledge and preferences sets (K and P). The results are given by the following algorithm.

1) Eliminate alerts that satisfy the knowledge formulas. We denote A' the resulting alerts set.
2) Transform the preference formulas according to the normalization function.
3) Compute the alerts satisfaction degree using the preference formulas.
4) Rank alerts according to their satisfaction degree.
5) Return the max preferred alerts to the security operator (the alerts having the minimum degree of satisfaction).

Intuitively, we can deduce that the algorithm has a polynomial complexity.

VI. Experimentation using real data

This section details the experimental studies on data collected within the PLACID project. This set of data is composed of more than 1,099,302 alerts.
The knowledge and preferences model used in tests is shown below. This model is extracted from the analysis of IDS returned alerts. This analysis showed the network weak points. Tests are done using rather restrictive knowledge and preferences set.

Knowledge base
Classification(successful) ∧ AnalyserName (prelude-lml)).
The knowledge expresses the fact that the security operator assesses that alerts coming from the HIDS[2] "prelude-lml" with a classification value "successful" are probably false positive alerts. Indeed, "prelude-lml" reports all the users logins. Then such alerts must be eliminated.

[2]Host IDS

initial alert number	Preferred alert number	Preferred alert rate
1099302	27810	2,53%

Table I
TEST RESULTS

Preferences base
p_1 : Severity(high) × Severity(medium) × Severity (low);
p_2 : AnalyserName(local-snort) × AnalyserName (public-snort).

The security operator expresses two preferences formulas. The first concerns the alert severity: alerts with a high severity are more important to treat than medium severity ones which are more important than low severity ones.
The second preference formula is about the IDS which generates the alert. The security operator assesses that "local-snort" alerts are more dangerous than the "public-snort" ones. Hence, "local-snort" monitors the local network traffic.
The interface of the developed application is intuitive for the security operator. Thus, he just enters the attributes values corresponding to his knowledge and/or his preferences in their dedicated fields, and runs the IFO-QCL process.
The most preferred alerts are those having a high severity value and coming from the local-snort IDS. The results are summarized in the table. I:
We deduce from this Table that only 2.53 % of the initial alerts are important for the security operator. This rate can be explained by the fact that the security operator knows well the monitored network, its weaknesses and the most dangerous attacks. Thus he expresses knowledge and preferences that allows to return only the most dangerous alerts.
The remaining alerts are not lost, they are ranked in decreasing order of importance (satisfaction). These alerts are shown in a second result application window and can be viewed by the security operator if necessary. Thus the security operator is more efficient if he has to analyse 2.53 % of the issued alerts
Our algorithm efficiency depends on the security operator knowledge and preferences, more they are restrictive less the resulted alert set is voluminous.
The running time depends on the initial alerts number, the knowledge/preference base, the preferred alerts number and obviously the host computer characteristics. The example above has a running time of 656 seconds.

VII. Conclusion and perspectives

We propose, in this paper, a new logic to deal with preferences called Instantiated First Ordered Qualitative Choice Logic (IFO-QCL). This logic treats the IDS alerts using the security operator's knowledge and preferences, in a polynomial time. An algorithm was built upon the defined logic. The application takes the security operator's knowledge and preferences and returns a set of stratified alerts. We test our application on real data using the whole PLACID dataset with a set of knowledge and preferences. The result is a set of filtered alerts which represent 2.53% from the initial ones. Other tests will be done with different knowledge and preference sets (more restrictive and more passive) in order to establish a link between the weakness of these sets and the resulting alert number rate.
Our future work will deal with the incomplete alerts that have some missed attributes. We also intend to extend the proposed approach to the possibilistic logic domain.

References

[1] J. Anderson, Computer security threat monitoring and surveillance, Technical report, James P.Anderson Company, Fort-Washington, Pennsylvania, April (1980).

[2] S. Axelsson, Intrusion Detection Systems: A Survey and Taxonomy, Technical Report No 99-15, Dept. of Computer Engineering, Chalmers University of Technology, Sweden, March (2000).

[3] S. Manganaris, M. Christensen, D. Zerkle, and K. Hermiz, A Data Mining Analysis of RTID Alarms, Recent Advances in Intrusion Detection Systems, Volume 34, Issue 4, pp. 571-577, October (2000).

[4] P. Chifflier and S. Tricaud, Intrusion Detection Systems Correlation: a Weapon of Mass Investigation, CanSecWest, Vancouver, March (2008).

[5] F. Cuppens, Managing alerts in a multi-intrusion detection environment, in: Proc. 17th Computer Security Applications Conference, pp. 22-31,December (2001).

[6] K. Julisch, Clustering intrusion detection alarms to support root cause analysis, ACM Transactions on Information and System Security (TISSEC), volume: 6, n4, pp. 443-471, November (2003).

[7] A. Valdes and K. Skinner, Probabilistic Alert Correlation, Recent Advances in Intrusion Detection (RAID 2001), numro 2212, lecture Notes in Computer Science. Springer-Verlag, pp.54-68, (2001).

[8] S.T. Eckmann, G. Vigna and R.A. Kemmerer, STATL: an attack language for state-based intrusion Detection, in. Journal of Computer Security, volume 10, n 1-2, pp.71-103, (2002).

[9] B. Morin and H. Debar, Correlation of Intrusion Symptoms: an Application of Chronicles,in Proceedings of the 6th International Symposium on Recent Advances in Intrusion Detection (RAID 2003), (Pittsburgh, PA), September (2003).

[10] F. Cuppens and R. Ortalo, LAMBDA: A Language to Model a Database for Detection of Attacks, in Proceedings of the Third International Workshop on Recent Advances in Intrusion Detection, pp.197-216, October (2000).

[11] F. Cuppens and A. Miege, Alert Correlation in a Cooperative Intrusion Detection Framework, in. Proceedings of the 2002 IEEE Symposium on Security and Privacy, pp.202, May (2002).

[12] P. Ning, Y. Cui and S. Reeves, Constructing attack scenarios through correlation of intrusion alerts. CCS '02 : Proceedings of the 9th ACM conference on computer and communications security, New York, NY, USA, ACM, pp. 245-254, (2002).

[13] S.J. Templeton and K. Levitt, A requires/provides model for computer attacks. NSPW '00 : Proceedings of the 2000 workshop on New security paradigms, New York, NY, USA, ACM, pp. 31-38, (2000).

[14] S. Benferhat, T. Kenaza and A. Mokhtari, A Naive Bayes Approach for Detecting Coordinated Attacks, COMPSAC 2008, pp.704-709, July - August (2008).

[15] S. Benferhat and K. Sedki, Two alternatives for handling preferences in qualitative choice logic, In Fuzzy Sets and Systems Journal (FSS'08), vol. 159, n 15, pp. 1889-1912, august (2008).

[16] G. Brewka, S. Benferhat and D. Le Berre, Qualitative Choice Logic, in. Artificial Intelligence Journal (AIJ), vol. 157, n 1-2, pp. 203-237, august (2004).

[17] F. Cuppens, Managing Alerts in a Multi-Intrusion Detection Environment, In proceedings of Recent Advances in Intrusion Detection, Davis, CA, USA, pp.22-31, October (2001).

[18] H. Debar and A. Wespi, Aggregation and Correlation of Intrusion-Detection Alerts, In proceedings of Recent Advances in Intrusion Detection, Davis, CA, USA, pp.85-103, October (2001).

[19] X. Qin and W. Lee, Attack Plan Recognition and Prediction Using Causal Networks, ACSAC-O4, pp.370-379, (2004).

[20] C. Geib and R. Goldman, Plan Recognition in Intrusion Detection Systems. In Proceeding of DARPA Information Survivability Conference and Exposition (DISCEX), Volume 1, pp.46-55, June (2001).

[21] H. Debar, D. Curry and B. Feinstein, The Intrusion Detection Message Exchange Format (IDMEF), Network Working Group, Request for Comments (RFC): 4765, Category: Experimental, SecureWorks, Inc. March (2007).

[22] J.W. Lyod, Foundation of logic programming. Spring-Verlag, 2nd Ed, ISBN-10: 3540181997, September(1987).

[23] M.J. Ranum, False Positives: A Users Guide to Making Sense of IDS Alarms, ICSA Labs IDSC, white paper, 2003.

Security Metrics for Java Bytecode Programs

Bandar Alshammari
College of Computer and Information Sciences
Aljouf University, Saudi Arabia
Email: bmshammeri@ju.edu.sa

Colin Fidge
Science and Engineering Faculty
Queensland University of Technology
Brisbane, Australia
Email: c.fidge@qut.edu.au

Diane Corney
Research Labs, Oracle
Brisbane, Australia
Email: diane.corney@oracle.com

***Abstract*—Although there are many approaches for developing secure programs, they are not necessarily helpful for evaluating the security of a pre-existing program. Software metrics promise an easy way of comparing the relative security of two programs or assessing the security impact of modifications to an existing one. Most studies in this area focus on high level source code but this approach fails to take compiler-specific code generation into account. In this work we describe a set of object-oriented Java bytecode security metrics which are capable of assessing the security of a compiled program from the point of view of potential information flow. These metrics can be used to compare the security of programs or assess the effect of program modifications on security using a tool which we have developed to automatically measure the security of a given Java bytecode program in terms of the accessibility of distinguished 'classified' attributes.**

***Index Terms*—Object-Orientation, Security Metrics, Security Analyser, Java Bytecode**

I. Introduction

Security is a critical aspect of software development, and there exist various approaches which aim to reduce security risks and vulnerabilities either through careful coding practices [1] [2] or through static analysis of the code's properties [3] [4] [5]. However, these techniques require considerable skill and effort to apply successfully, and are not always applicable to pre-existing programs. Alternatively, security metrics which quantify the security level of a given program [6] could offer a 'pushbutton' solution which can be applied easily to given programs.

Most existing security metrics are based on high-level code [6] which does not always give reliable results because analysing source code does not take compiler-specific code generation into account. On the other hand, low-level metrics, e.g., those derived from Java bytecode instructions, could provide better information about the executable program [7] [8].

Dynamic metrics have been studied in many projects due to their importance and reliability [7], including the relationships between static and dynamic coupling metrics [8]. Java dynamic metrics have also been studied extensively, including Dufour et al.'s work [7] which defines a number of Java dynamic metrics to evaluate compiler optimisations. Binder and Hulass' study into control flow metrics is also relevant [9].

In our previous work [10] [11] [12] we defined several security metrics for UML class designs, and described a tool for automatically evaluating such metrics [13]. These metrics assess the potential flow of 'classified' information by measuring the accessibility of selected data items based on the security design principles of "reducing the size of the attack surface" [14] [15] and "granting least privilege" [16] [17] [2]. They allow software developers to easily assess the impact of code modifications on overall program security and to compare the relative security of different versions of the same program.

In this paper, we describe our program code metrics in detail, as an extension of our earlier metrics for UML designs [10] [11] and use a large-scale case study, involving multiple programs, to show how the metrics can accurately measure changes in the security of program code.

II. Program Code Security Metrics Definitions

This section defines our security metrics for assessing the security level of programs. Our approach is based on static analysis of the program code. For instance, when we say that a method 'accesses' or 'interacts with' a classified attribute, it means that the method's compiled code contains an instruction that may read or write an attribute labelled by the programmer as 'classified'. Of course, if this instruction appears within a conditional statement, there is no guarantee that the method will do so every time the program executes. Thus, our metrics are safely conservative and measure the *potential* flow of classified data.

A. Data Encapsulation-Based Security Metrics

Code security metrics based on data encapsulation aim to statically measure the potential flow of information from classified attributes and methods from the perspective of access modifiers. These metrics consider attributes annotated by the programmer as 'classified' or ones which derive their values from classified attributes, and methods which access classified attributes. They also measure whether the program imports the Java reflection library due to the risk that this library can be used by new code to access the value of *any* classified attribute in an existing security-critical part of the program [18].

We divide the metrics for this property into four kinds of accessibility: classified instance attributes (CIDA); classified class attributes (CCDA); methods which access classified attributes (COA) and accessibility through reflection (RPB). The metrics are defined so as to penalise programs that make classified attributes more accessible. Higher values indicate higher accessibility to classified attributes and methods, and

hence a larger 'attack surface'. This means a higher possibility for confidential data to be exposed to unauthorised parties. Aiming for lower values of these metrics adheres to the security principle of reducing the size of the attack surface [14], [18]. Here we define the security metric RPB while the descriptions and definitions of the CIDA, CCDA and COA metrics can be found elsewhere [10].

Reflection Package Boolean (RPB): This code-level metric measures the accessibility through reflection of classified data in a given program. It is defined as *"A boolean value representing whether the Java program imports the Java reflection package (1) or not (0)"*. This metric is only concerned with whether or not the application itself is importing the Java reflection library (i.e., information flow within the program itself) and does not consider an attacker reflecting on the application code elsewhere. The reflection metric equals 1 if the program imports the Java reflection library or 0 otherwise. Importing the Java reflection library means a higher possibility for confidential data to be exposed to unauthorised parties.

$$RPB(P) = \begin{cases} 1, & \textit{if reflection imported} \\ 0, & \textit{otherwise} \end{cases} \quad (1)$$

B. Cohesion-Based Security Metrics

The property of cohesion measures the interactions between attributes and methods within a given class [19]. We have previously defined four cohesion-based security metrics to measure the potential flow of classified information caused by interactions between methods and classified attributes in an object-oriented design [10]. Programs with higher interaction between methods and classified attributes have stronger cohesion, and hence are less secure. The previously-defined metrics are divided into four parts: the interactions of mutators (setters) with classified attributes (CMAI); the interactions of accessors (getters) with classified attributes (CAAI); the weight of classified attributes' interactions with methods (CAIW); and the proportion of classified methods (CMW). In this paper, we extend these design-level metrics to include the proportion of classified writing methods in the program (CWMP).

Classified Writing Methods Proportion (CWMP): This metric aims to measure the proportion of methods which write classified attributes in a particular program. We define this metric as: "*The ratio of the number of methods which write classified attributes to the total number of classified methods in the program*". In our case, we assume a writing method in Java is one which writes an attribute to outside its class by calling a method from the `java.io` package. This includes methods whose class name contains either 'write', 'print', or 'out'. Therefore, a 'classified writer' is a method which uses one of these classes to write a classified attribute. Fewer such methods adheres to the principle of granting least privilege [17].

Consider the set of classified methods in a program P as $CM = \{cm_1, \ldots, cm_n\}$ and the set of the classified writing methods as $CWM = \{cwm_1, \ldots, cwm_n\}$ such that $CWM \subseteq CM$. (Given a set S, let the magnitude operator $|S|$ returns the size of the set.) Then, $CWMP$ is expressed as:

$$CWMP(P) = \frac{|CWM|}{|CM|} \quad (2)$$

C. Coupling-Based Security Metric

Our security coupling metric (CCC) for class designs [11] measures the degree of potential flow of classified data caused by the interactions between classes and classified attributes in a given object-oriented design. This metric is adopted without change for program code.

D. Composition-Based Security Metric

As explained previously [11], composition yields a (weak) possibility of potential information flow for classified data. In the case of programming in Java, it is possible to access composed-part (inner) classes unless they are marked as private. Hence, it is recommended to avoid using non-private inner classes in security-critical code [20]. In our case, we assume that using private composed-part classes should reduce the potential flow of classified data, and hence produce more secure programs. Our design-level composition-based metric (CPCC) [11] is adopted for program code to measure this.

E. Extensibility-Based Security Metrics

To have more secure programs, classes and methods which can access classified data should be prevented from being extended by other classes and methods [20], [21], since doing so makes classified data accessible in the new code. We have identified two metrics (CCE and CME) which reward the use of non-extended classes and methods [11].

Another such threat with regard to this property is code that assigns classified values to a variable or parameter that is not subsequently used, because this makes it possible to add code to the program that accesses the classified data but has no observable effect on the program's behaviour. To prevent this we need to identify classified values that are defined but not used, classified methods that are declared but not called, and critical classes that are never used. Thus, we define three new code-level metrics which penalise unused classified attributes (UACA), uncalled classified methods (UCAM) and unused critical classes (UCAC) in a program.

These features could allow unauthorised parties to acquire privileges on security-critical data without affecting the program's original behaviour. Making code inextensible eliminates this risk, and hence reduces the possibility of information flow from these attributes, methods and classes, which adheres to the principle of granting the least privilege [17].

Unaccessed Assigned Classified Attribute (UACA): This metric is define as "*The ratio of the number of classified attributes that are assigned but never used to the total number of classified attributes in the program*". It measures those classified attributes which are assigned, either directly by an "=" assignment or by parameter passing through value or reference, but never subsequently used.

Consider the set of classified attributes in a program P as $CA = \{ca_1, \ldots, ca_n\}$ and the set of classified attributes which are assigned but never used in the same program as $UCA =$

$\{uca_1, \ldots, uca_n\}$, such that $UCA \subseteq CA$. Then, we define the Unaccessed Assigned Classified Attribute metric as follows.

$$UACA(P) = \frac{|UCA|}{|CA|} \tag{3}$$

Uncalled Classified Accessor Method (UCAM): This metric measures declared methods which access classified attributes but are never called. It is defined as "*The ratio of the number of classified methods that access a classified attribute but are never called by other methods to the total number of classified methods in the program*".

Consider a set of classified methods in a program P as $CM = \{cm_1, \ldots, cm_n\}$ and classified accessors that are never called by other methods $UCM = \{ucm_1, \ldots, ucm_n\}$, such that $UCM \subseteq CM$. Then, we define the Uncalled Classified Accessor Method metric as follows.

$$UCAM(P) = \frac{|UCM|}{|CM|} \tag{4}$$

Unused Critical Accessor Class (UCAC): This measures classes which contain classified accessor methods that are never used in any other classes. It is defined as "*The ratio of the number of classes which contain classified methods that access classified attributes but are never used by other classes to the total number of critical classes in the program*".

Consider the set of critical classes in a program P as $CC = \{cc_1, \ldots, cc_n\}$ and classes which have classified accessors that are never used by other classes as $UCC = \{ucc_1, \ldots, ucc_n\}$, such that $UCC \subseteq CC$. hen, we define the Unused Critical Accessor Class metric as follows.

$$UCAC(P) = \frac{|UCC|}{|CC|} \tag{5}$$

F. Design Size-Based Security Metrics

With respect to security, a program with a large amount of security-critical code has a higher chance of potential flow of classified information, and hence is less secure [6]. It has also been shown that security-sensitive classes must avoid serialisation since this allows the values of private fields to be accessed from outside the program [18]. Our design size-based security metric (CDP) defined previously [11] already measures the proportion of the program that is devoted to security-critical classes (CDP). For program code we define another security metric devoted to security-critical serialisable classes (CSCP).

Critical Serialised Classes Proportion (CSCP): This metric measures the risk associated with critical serialisable classes in a given program. We define it as "*The ratio of the number of critical serialised classes to the total number of critical classes in the program*". It rewards programs with a smaller percentage and number of such classes and penalises the use of security-critical serialisable classes. Therefore, lower values of the CSCP metric indicates a lower proportion of security-critical serialisable classes, which can give privileges over confidential data, and thus satisfies the least privilege principle [17].

Consider the set of the critical classes in program P as $CC = \{cc_1, \ldots, cc_n\}$ and the set of critical serialised classes is $CSC = \{csc_1, \ldots, csc_n\}$, such that $CSC \subseteq CC$. Then, we define the Critical Serialised Classes Proportion metric as follows.

$$CSCP(P) = \frac{|CSC|}{|CC|} \tag{6}$$

G. Inheritance-Based Security Metrics

The earlier design-level metrics [11] which consider inheritance are equally-applicable to program code. The include metrics which penalise classes (CSP and CSI), methods (CMI), and attribute (CAI) hierarchies in which classified data appears near the top and is thus more accessible.

III. Program Code Security Metrics Experimental Results

To demonstrate the validity of our code-level metrics, and show the capabilities of our Java Bytecode Security Analyser [13], we conducted an experiment with several large-scale open source Java programs. We used the tool to assess the relative security of different versions of the same program. Our hypothesis was that a program's level of security should, on average, improve over time, as bugs are fixed and the program code is improved, although the addition of new security-critical functionality may cause a worsening of overall security levels.

A. Approach

We began with five existing open source Java security projects which were chosen from the most frequently downloaded security projects on the SourceForge website [22]. The chosen programs consisted of the following: Jacksum, jGuard, Kasai, JSecurity, and JXplorer. They all mainly concentrate on providing a framework for handling authentication, authorization, enterprise session management, and cryptography services [22]. For each project, we chose a specific version which was modified in a number of subsequent updates, to fix bugs found in the previous releases. In this way we could compare different versions of each program with identical functionality but (hopefully) improved code quality. All of these programs are security-related, so we could reasonably expect successive releases to be more secure than their predecessors.

B. Program Annotations

First, we manually annotated at the Java source code level a number of attributes in each project to be 'classified', choosing attributes whose names and associated code comments indicated that they are likely to store confidential data. We annotated the same attributes for all the different releases of the same program in order to make our comparisons fair. For example, in the program JSecurity, classified attributes were: `username`, `password` and `rememberme` in the `UsernamePasswordToken` class. In the Kasai project, `login`, `password` and `superUser` in the `User` class and in class `Role id` and `name` were

Table I: Program Characteristics

Program	Version	Attributes	Classified Attributes	Methods	Classified Methods	Classes	Critical Classes
JSecurity	0.9.0.A	384	3	1790	24	241	1
	0.9.0.B1	383	3	1822	24	245	1
	0.9.0.RC1	444	3	1996	24	261	1
	0.9.Stable	435	3	2028	23	309	1
Kasai	1.1.0.B1	86	5	498	49	51	2
	1.1.0.B2	86	5	506	54	51	2
	1.1.0.B3	86	5	506	54	51	2
	1.1.Stable	86	5	506	54	51	2
Jacksum	1.2	146	5	179	28	24	2
	1.3	159	7	217	41	29	3
	1.4	260	9	304	44	36	3
	1.5	299	16	355	56	44	3
jGuard	0.65.1	219	4	344	40	45	2
	0.65.2	219	4	344	40	45	2
	0.65.3	219	4	344	40	45	2
	0.65.4	219	4	344	40	45	2
JXplorer	3.2.B1	1985	96	3225	421	406	28
	3.2.B2	2075	113	3332	486	413	28
	3.2.B3	2072	142	3336	558	413	31
	3.2.Stable	2077	151	3345	518	415	31

Table II: Data Encapsulation and Cohesion-Based Security Metrics

Program	Version	CIDA	CCDA	COA	RPB	CMAI	CAAI	CAIW	CMW	CWMP
JSecurity	0.9.0.A	0	0	1	1	0.0228	0.0037	0.0225	0.0134	0
	0.9.0.B1	0	0	1	1	0.0226	0.0036	0.0223	0.0132	0
	0.9.0.RC1	0	0	1	1	0.0208	0.0032	0.0205	0.012	0
	0.9.Stable	0	0	1	1	0.0207	0.0032	0.0206	0.0113	0
Kasai	1.1.0.B1	0	0	1	0	0.031	0.028	0.115	0.09	0
	1.1.0.B2	0	0	1	0	0.033	0.03	0.125	0.11	0
	1.1.0.B3	0	0	1	0	0.033	0.03	0.125	0.11	0
	1.1.Stable	0	0	1	0	0.033	0.03	0.125	0.11	0
Jacksum	1.2	0.8	0	1	0	0.05	0.034	0.08	0.156	0
	1.3	0.6	0	0.976	0	0.047	0.044	0.116	0.189	0.024
	1.4	0.4	0	0.931	0	0.029	0.034	0.088	0.145	0.023
	1.5	0.25	0	0.928	0	0.022	0.03	0.123	0.158	0.036
jGuard	0.65.1	0.5	0	0.85	0	0.026	0.026	0.036	0.116	0
	0.65.2	0.5	0	0.85	0	0.026	0.026	0.036	0.116	0
	0.65.3	0.5	0	0.85	0	0.026	0.026	0.036	0.116	0
	0.65.4	0.5	0	0.85	0	0.026	0.026	0.036	0.116	0
JXplorer	3.2.B1	0.8	0.01	0.94	1	0.0025	0.0028	0.081	0.131	0.007
	3.2.B2	0.8	0.008	0.92	1	0.0025	0.0026	0.088	0.146	0.006
	3.2.B3	0.81	0.02	0.92	1	0.0025	0.0024	0.105	0.167	0.007
	3.2.Stable	0.83	0.019	0.92	1	0.0025	0.0024	0.109	0.155	0.008

annotated as classified attributes. In the Checksum project, `value`, `length`, `separator`, and `filename` in class `AbstractChecksum`, and `val` in class `Crc16` were annotated as classified. In the JGuard project, `name` and `applicationName` in the `JGuardPrincipal` class and `id` and `value` in the `JGuardCredential` class were annoated as classified. In the JXplorer project, `uniqueID` and `addressIP` in the `Name` class and `tag` and `name` in the `ASN1Type` class were annoated as classified.

C. Program Characteristics

Table I shows a number of static characteristics of the studied programs after we annotated our choices of classified attributes for each. The arrows show how each metric has changed since the previous release. Upwards arrows (red) indicate a worsening of security and downwards arrows (green) indicate that security has improved. These characteristics are one of the outputs of the JBSA tool. They include the total number of attributes, classified attributes, methods, classified methods, classes and critical classes for each program. In each successive release of each project, most of these characteristics either grew or stayed the same. For instance, the number of classified methods, i.e., those which may access our annotated attributes, either directly or indirectly, can be seen to grow dramatically in successive revisions of Jacksum and JXplorer.

In order to show that our security metrics reflect the program's true security level, we inspected the code of some of the analysed programs in this experiment. This inspection aimed to show that our security metrics correctly mirror the improvement or worsening of security caused by specific changes to security-relevant code.

For instance, it was found that in program Kasai the second release has added a number of additional methods some of which contain a flow of classified information. One such new method is `overridePassword` in class `User` which interacts with the classified attribute `password` and does similar operations to another existing method `setPassword`. It thus creates an additional access point for classified attributes.

Table III: Coupling, Composition and Extensibility-Based Security Metrics

Program	Version	CCC	CPCC	CCE	CME	UACA	UCAM	UCAC
JSecurity	0.9.0.A	0.0111	1	1	1	0	0.75	0
	0.9.0.B1	0.0109	1	1	1	0	0.75	0
	0.9.0.RC1	0.0103	1	1	1	0	0.75	0
	0.9.Stable	0.0054	1	1	1	0	0.75	0
Kasai	1.1.0.B1	0.036	1	1	1	0	0.38	0
	1.1.0.B2	0.04	1	1	1	0	0.36	0
	1.1.0.B3	0.04	1	1	1	0	0.36	0
	1.1.Stable	0.04	1	1	1	0	0.36	0
Jacksum	1.2	0.087	1	1	1	0	0.71	0
	1.3	0.061	1	1	1	0	0.75	0
	1.4	0.048	1	1	1	0	0.8	0
	1.5	0.033	1	1	0.98	0	0.85	0
jGuard	0.65.1	0.0625	1	1	1	0	0.6	0
	0.65.2	0.0625	1	1	1	0	0.6	0
	0.65.3	0.0625	1	1	1	0	0.6	0
	0.65.4	0.0625	1	1	1	0	0.6	0
JXplorer	3.2.B1	0.0098	1	1	1	0.029	0.74	0.18
	3.2.B2	0.0094	1	1	1	0.032	0.73	0.22
	3.2.B3	0.0086	1	1	1	0.04	0.74	0.23
	3.2.Stable	0.0082	1	1	1	0.037	0.76	0.23

Table IV: Design Size and Inheritance-Based Security Metrics

Program	Version	CDP	CSCP	CSP	CSI	CMI	CAI
JSecurity	0.9.0.A	0.0041	0	0	0	0	0
	0.9.0.B1	0.0041	0	0	0	0	0
	0.9.0.RC1	0.0038	0	0	0	0	0
	0.9.Stable	0.0032	0	0	0	0	0
Kasai	1.1.0.B1	0.039	0	0	0	0	0
	1.1.0.B2	0.039	0	0	0	0	0
	1.1.0.B3	0.039	0	0	0	0	0
	1.1.Stable	0.039	0	0	0	0	0
Jacksum	1.2	0.083	0	0.5	0.174	0.722	0.8
	1.3	0.103	0	0.5	0.232	0.722	0.8
	1.4	0.083	0	0.5	0.2	0.722	0.8
	1.5	0.068	0	0.5	0.221	0.75	0.8
jGuard	0.65.1	0.044	0.5	0	0	0	0
	0.65.2	0.044	0.5	0	0	0	0
	0.65.3	0.044	0.5	0	0	0	0
	0.65.4	0.044	0.5	0	0	0	0
JXplorer	3.2.B1	0.069	0.036	0	0	0	0
	3.2.B2	0.068	0.036	0	0	0	0
	3.2.B3	0.075	0.032	0	0	0	0
	3.2.Stable	0.074	0.032	0	0	0	0

Similarly the second release of the Kasai program overloads an existing security-critical method. Class `KasaiFacade` has two methods called `createUser` that have a flow of classified information. This means that there are more methods in this release which interact with classified information than in the previous release. In fact, these new methods have similar responsibilities as existing ones and could have been avoided.

Another example was found in program Jacksum 1.3 which has a method `getChecksumInstance` that returns classified information and is assigned to a new non-classified attribute `checksum`. The method calling `getChecksumInstance` thus exposes classified information which could be exploited by unauthorised parties. Therefore, we expect the security of Jacksum 1.3 to worsen due to the additional vulnerabilities added to it and our security metrics should reflect this change.

On the other hand, there are cases where a potential vulnerability has been removed from the program in a successive release. This was found in program JSecurity where method `executeLogin` in class `FormAuthenticationFilter` that used to be a potential vulnerability in the third release was deleted from the program's fourth release. This has resulted in reducing the number of insecure methods (i.e., those which interact with security-critical information). Such changes could contribute to improvements in the program's overall security and therefore our security metrics should reflect this improvement.

D. Programs Security Metrics Results

The results of calculating our code-level security metrics (using our JBSA tool) for each release of each project are summarised in Tables II to IV. Given that lower values of each metric are considered more secure, programs whose metrics decrease should be those whose security has improved. We expected these security-related programs would improve their overall security with each new release.

With regard to the results shown in the tables, two of the five programs, JSecurity and Jacksum, show an obvious improvement in their security metrics from previous versions.

An exception is jGuard whose metrics are unchanged for all releases. This suggests that only insignificant changes were made to the code which had no security impact at all. This impression is confirmed by the characteristics in Table I which reveal that no major changes were made to the code's size between releases. (Nevertheless, the program's change log says that a number of small bug fixes were made in each revision.)

The other exception is Kasai whose metrics in Table II have slightly increased in value between releases 1.1.0.B1 and 1.1.0.B2, meaning a worsening in security, after which the program was stable. From Table II this would appear to be because the second release added eight new methods, five of which were 'classified'. These new methods account for the slight increase in three of the cohesion and coupling-security related metrics exhibited by the second version of the program.

A similar case is shown by the results of JXplorer where some of its metrics often increased. The reason for this is clearly shown by the program characteristics in Table I which indicate that the program has had a significant amount of new code added. Thus, the program has major increases in some of its security metrics and worse security overall with regard to those metrics. Nevertheless, some of the program's metrics, including COA, CAAI, CCC and CSCP, have managed to decrease and thus its security has improved in this regard.

Comparing these results with the code inspections described in Section III-C we see that our metrics for these programs have accurately reflected the changes in the security of these programs with regard to either removing or adding potential security vulnerabilities. For instance, in Kasai's second release our security metrics that relate to measuring the security of classified methods have shown that security has worsened in this release as expected due to the addition of new security-critical methods and attributes, which is the case for program Jacksum 1.3 as well. On the other hand, security improved for the JSecurity program's fourth release as a potentially vulnerable method was deleted. This change in the code produced a corresponding decrease in metrics that measure the proportion of classified methods (CMAI and CMW). However because the deleted classified methods also interacted with several *non-classified* attributes, the metric that measures the proportion of classified interactions (CAIW) increased.

From this experiment, we can conclude that our security metrics offer a simple, easy to apply and easy to interpret approach to quantifying the security of a given program, once it is properly annotated.

IV. Conclusion

In this work we have described a number of security metrics for object-oriented programs which are measurable at the level of bytecode instructions. Using this approach we capture the exact behavior of a Java program in the Java Virtual Machine, which gives accurate results. They provide developers with a simple way of identifying and fixing security vulnerabilities which might occur from the perspective of information flow of confidential data. The security metrics were demonstrated using a tool which analyses Java bytecode, applied to actual large-scale Java projects. This case study produced results which match our intuitions about the way a program's security changes as its code is extended or debugged.

References

[1] M. Howard and D. LeBlanc, *Writing Secure Code*. Redmond, Wash.: Microsoft Press, 2002.

[2] G. McGraw, *Software Security: Building Security In*. Upper Saddle River, NJ: Addison-Wesley, 2006.

[3] V. B. Livshits and M. S. Lam, "Finding security vulnerabilities in Java applications with static analysis," in *SSYM'05: Proceedings of the 14th conference on USENIX Security Symposium*, (Berkeley, CA, USA), pp. 18–18, USENIX Association, 2005.

[4] G. Smith, *Principles of Secure Information Flow Analysis*, vol. 27, pp. 291–307. Springer, 2007.

[5] D. Volpano and C. Irvine, "Secure flow typing," *Computers and Security*, vol. 16, no. 2, pp. 137–144, 1997.

[6] I. Chowdhury, B. Chan, and M. Zulkernine, "Security metrics for source code structures," in *Proceedings of the Fourth International Workshop on Software Engineering for Secure Systems*, (Leipzig, Germany), pp. 57–64, ACM, 2008.

[7] B. Dufour, K. Driesen, L. Hendren, and C. Verbrugge, "Dynamic metrics for Java," *SIGPLAN Notices*, vol. 38, no. 11, pp. 149–168, 2003.

[8] Á. Mitchell and J. F. Power, "A study of the influence of coverage on the relationship between static and dynamic coupling metrics," *Science of Computer Programming*, vol. 59, pp. 4–25, January 2006.

[9] W. Binder and J. Hulaas, "Flexible and efficient measurement of dynamic bytecode metrics," in *Proceedings of the 5th International Conference on Generative Programming and Component Engineering (GPCE 2006)*, pp. 171–180, 2006.

[10] B. Alshammari, C. J. Fidge, and D. Corney, "Security metrics for object-oriented class designs," in *Proceedings of the Ninth International Conference on Quality Software (QSIC 2009)*, (Jeju, Korea), pp. 11–20, IEEE, 2009.

[11] B. Alshammari, C. J. Fidge, and D. Corney, "Security metrics for object-oriented designs," in *Proceedings of the Twenty-First Australian Software Engineering Conference (ASWEC 2010), Auckland, 6–9 April* (J. Noble and C. J. Fidge, eds.), (California, USA), pp. 55–64, IEEE Computer Society, 2010.

[12] B. Alshammari, C. Fidge, and D. Corney, "Security assessment of code refactoring rules," in *WIAR'2012; Proceedings of the National Workshop on Information Assurance Research, Riyadh, Saudi Arabia, April 18*, pp. 1–10, VDE, 2012.

[13] B. Alshammari, C. Fidge, and D. Corney, "An automated tool for assessing security-critical designs and programs," in *WIAR'2012; Proceedings of the National Workshop on Information Assurance Research, Riyadh, Saudi Arabia, April 18*, pp. 1–10, VDE, 2012.

[14] M. Howard, "Attack surface: Mitigate security risks by minimizing the code you expose to untrusted users," *Microsoft MSDN Magazine*, vol. 11, 2004.

[15] P. Manadhata and J. Wing, "An attack surface metric," *IEEE Transactions on Software Engineering*, vol. PP, no. 99, p. 1, 2010.

[16] J. H. Saltzer and M. D. Schroeder, "The protection of information in operating systems," in *Proceedings of the IEEE*, vol. 63, pp. 1278–1308, 1975.

[17] M. Bishop, *Computer Security: Art and Science*. Boston: Addison-Wesley, 2003.

[18] Java SE Security, "Secure Coding Guidelines for the Java Programming Language." Sun Developer Network, Version 3.0. http://java.sun.com/security/seccode guide.html [Accessed July 9, 2010].

[19] L. C. Briand, J. W. Daly, and J. K. Wuest, "A unified framework for cohesion measurement," in *Proceedings of the Fourth International Symposium on Software Metrics*, IEEE Computer Society, 1997.

[20] G. McGraw and E. Felten, *Securing Java: Getting Down to Business with Mobile Code*. New York: Wiley Computer Pub., second ed., 1999.

[21] M. Y. Liu and I. Traore, "Empirical relation between coupling and attackability in software systems: a case study on DOS," in *Proceedings of the 2006 Workshop on Programming Languages and Analysis for Security, Ottawa*, (Ottawa, Ontario, Canada), pp. 57–64, ACM, 2006.

[22] SourceForge, "SourceForge source code repository," July 2010. http://sourceforge.net/ [Accessed July 9, 2010].

An Empirical Study of an Improved Web Application Fuzz Testing Technique

Lili Yu[1, 2]
School of Computer Science and Engineering
Beihang University[1]
the Second Artillery Software Testing Center[2]
Beijing, China
xueer-123@263.net

Zi Yuan[1]
School of Computer Science and Engineering
Beihang University[1]
Beijing, China

Chao Liu[1]
School of Computer Science and Engineering
Beihang University[1]
Beijing, China

Feng Chen[2]
the Second Artillery Software Testing Center[2]
Beijing, China

***Abstract*—**Web application security vulnerabilities can be defined as vulnerabilities in the design, the implementation and the security policy of each component of a web system. Fuzz testing, as a brute security vulnerability discovery method, has gained more and more popularity in recent web application. In this paper, an improved web fuzz testing technique is presented and a tool named as Wfuzzer is accordingly designed and implemented. Three aspects (i.e. automatic analysis, fuzz data generation and vulnerability detection method) are enhanced. An empirical study is performed to evaluate the performance of the approach. The experiment results show certain advantages of using Wfuzzer in the degree of automation and vulnerability detection accuracy.

Keywords-Web application; fuzz testing; security vulnerability; security testing

I. INTRODUCTION

With the development of the internet technology, the web application plays more and more important role in life and work. While there is no denying the fact that the potential safety hazard brought by the openness of the internet has been a serious threat to the user and the public information security, among which, the web application own vulnerability is one of the main reason causing the security problem [1]. Web application security vulnerabilities can be defined as vulnerabilities in the design, the implementation and the security policy of each component of a web system (e.g., web application, web server, database, etc.). Web application, as a special kind of software, is faced with more complicated user environments and more serious security threats than the traditional stand-alone software. Web application security vulnerabilities on the internet have become one of the most serious security hidden dangers. According to a Report "the Symantec Internet Security Threat Report", more than 60% of the software security vulnerability is about the web application [2]. The reason that caused the web application security vulnerability is, on the one hand, due to the web application developers insufficient experience, even more important is that during the web application testing process, testers pay more attention to verify the normal function, but often neglect the security needs.

In the web application field, the common detecting security vulnerability methods are the static analysis technique and the dynamic analysis technique [3]. Fuzz testing is recently to be applied to the web application testing. In the web application testing, the fuzz testing technique can purposefully construct a large number of valid or invalid user input data submitted to the web application, at the same time, monitor the behavior of the web application by a variety of approaches, furthermore, analyze the abnormal or collapsed reason, which can help to discover security vulnerabilities existed in the web application.

In this paper, an improved web application fuzz testing method is presented. The following aspects are enhanced: using the web crawler method to implement the automation of scanning and analyzing the target web application, aiming at getting the input vector (e.g., URL, web form); constructing the fuzz data according to the specific vulnerability type and submitting to the target web application, at the same time, using the case mix method or the coding method to deform the fuzz data with the purpose of bypassing the validation and filtering mechanism; designing to monitor and analyze the target web application response behavior by a variety of approaches, aiming at judging whether there are vulnerabilities. Furthermore, based on the above-mentioned method, Wfuzzer, a web application fuzz testing prototype tool, is designed and implemented. Wfuzzer can perform the automation fuzz testing on the target web application, supporting to detect the SQL injection, the cross site scripting, the directory traversal and etc. And an empirical study is performed to evaluate the performance of the Wfuzzer. The obtained results prove certain advantages of using Wfuzzer in the degree of automation and vulnerability detection accuracy.

The remainder of this paper is organized as follows: section II briefly presents the fuzz testing, and Web fuzz testing;

section III reports an improved web application fuzz testing technique and key steps of designing the Wfuzzer tool; section IV performs an empirical study to evaluate the performance of our approach; finally conclusions are presented in Section V.

II. BACKGROUND

A. Fuzz Testing

Fuzz testing is a technique developed by Barton Miller at a Wisconsin-Madison research project in 1989. The project constructs a simple fuzzy unit to send a random string method in order to test the UNIX application robustness [4]. After twenty years of development, fuzz testing has become a synonym of software testing methodology, and has been widely used in various types of software and system testing in practice: the network protocol testing [5] [6], the file format testing, the memory data test, etc. The tool to realize the fuzz testing is usually called as Fuzzer. According to the different fuzz data generation method, Fuzzer can be divided into the two categories: the mutation-based and the generation-based. However, regardless of what kind of the fuzzy device, the implementation is usually divided into the following stages: identifying the target; recognizing the input; generating the test data; implementing the fuzz testing; monitoring the abnormal; determining availability of the abnormal [7].

B. Web Application Fuzz Testing

In recent years, web application fuzz testing technique research has gained more and more attention. Hammersland [8] has considered that the user input may make the web application being in an uncertainty state, resulting in the abnormal behavior of the web application. In his research, the use of a semi-automatic fuzz testing method is proposed to test the web application processing ability for dealing with all type of the user inputs. Graaf [3] has considered Hammersland's fuzz testing method is not intelligent, and on Hammersland's research basis, Graaf has presented a more "intelligent" fuzz testing method. In his research, the semi-legal fuzz data is generated, aiming at using the specific web framework to prevent it from being filtered or rejected by the input validation module of the web framework.

About the open source web application fuzz tools, WebScarab contains a basic fuzzy device and injects the web application parameters into the fuzz data. WSFuzzer can realize the web service communication parameters' fuzzification according to the WSDL documents provided by the user or the XML files defining the service SOAP structure. SPIKE Proxy identifies SQL injection and buffer overflow and XSS, and the other type of vulnerabilities through capturing the web browser requests, and allowing the user to run a series of predefined audit for a target Web application. WebFuzz is a fuzz testing tool based on windows platform graphical interface, which can detect directory traversal, buffer overflow, SQL injection, and the other type of vulnerabilities existed in the web application. About the business tools, the representative is as follows: WebInspect developed by SPI Dynamics Company, is a web application test tool to provide the graphical SPI fuzzy device in the form of the component. BeSTORM developed by Beyond Security can generate the fuzz data set for multi-protocols including HTTP, similarly including the HTTP test tool developed by Codenomicon.

III. WFUZZER-A NEW WEB APPLICATION FUZZ TESTING

In this part, we first study the deficiency of the above-mentioned current tools, then introduce our approach in detail from three stages: identify the input goal, construct the fuzz data and monitor the response behavior.

A. The Existed Shortcomings

The results of comparing the current web application fuzz testing tool are shown in table I.

TABLE I. COMPARE OF WEB APPLICATION FUZZ TESTING TOOL

Tool name	Language	Visual interface	Type of vulnerability	Automation degree	Scalability
WebScarab	Java	YES	Single	Normal	Poor
WSFuzzer	Python	NO	Single	Low	Poor
WebFuzz	C#	YES	Multi	Normal	Poor
WfuSPIzz	Python	NO	Single	Low	Good
SPI	C#	YES	Multi	Normal	Poor
JBroFuzz	Java	YES	Multi	Low	General
RatProxy	C	NO	Single	Normal	Poor
ProxyStrike	Python	YES	Single	Low	General
WFT	C#	YES	Single	Low	General

There is a certain degree of shortage, mainly reflected in the following:

- Automation degree is not high. Some fuzzy tools need personal participation in the web application analysis and fuzz data generation process. Such as WebScarab, fuzz data is generated according to the tester to provide the normal request;
- Omission of misinformation rate is high. The fuzz data generation and the vulnerability detection method provided by some fuzzy tools are relatively simple, which causes the misinformation rate of the vulnerability omission of test results is higher. Such as WebFuzz, when generating the fuzz data, only the request parameters using an attack payload (i.e., the input data may lead to security vulnerabilities), is replaced, and the vulnerability detection is also simply by looking back to the web page for the presence of specific content;
- Scalability is poor. Most of the commercial fuzz tools do not provide the extended function. WSFuzzer and the other open source tools also have the high coupling degree problem. On the basis of the redeveloped to provide the support for the new type of vulnerabilities has large cost.

B. An Improved Web Application Fuzz Testing Technique

1) Automatic analysis of web application: Using the web crawler technique to automatically scan the target web application, we can go through the web application which contains all of the pages and analyze the user input data field of the web application, mainly including the URL and the web form.

2) Improving vulnerability explore methods: Improvement of the vulnerability discovery method is mainly manifested in the fuzz data generation and the vulnerability detection. The fuzz data generation adopts the mutation-based method. Due to the paper limitation, here we mainly introduce the SQL injection discovery method and cross site scripting discovery method in the following.

a) SQL Injection: The principle of SQL injection is that the attacker injects the SQL statement into the input data. The SQL statement may be submitted to the background database and is considered to be a normal SQL statement for implementation, which will cause data disclosure, manipulation, etc. Our research adopts the method based on SQL grammar mistakes injection and the method based on time blind injection:

- Based on SQL grammar mistakes injection: The method inserts special characters into a normal HTTP request parameters, which will destroy the SQL query syntax. If the web application does not carry on the filtering and escaping, these containing errors grammar SQL statements will eventually be submitted to the background database, and result in the return error message. The method to compare the pages containing specific database error message with the normal request page can determine whether the injection succeeds or not.
- Based on time blind injection: Under the condition of no outputting background database error information of the web application, the method based on the time blind injection can be adopted. This method can inject the SQL request statements containing the background database delay returns results into the normal HTTP request parameters. Setting the delay and the HTTP response timeout time, and comparing the normal request response time, can judge whether injection successes.

b) Cross Site Scripting: The principle is that the attacker embeds the malicious code into the input data. If the web application does not validate these data and directly resolve them to the HTML code, these malicious codes will be carried out once the other users visit the page, such as the session hijacking, the shielding and forging page information, the breakthrough network security restriction, and etc. About the cross site scripting vulnerabilities, the fuzz data generation mechanism is relatively more complicated. According to the different attack payload position, the cross site scripting can be divided into two categories: the outside-site labels and the inner-site labels. The former is directly injected against the payload, such as*<script>alert ("xss")</script>*. While the latter shuts the incomplete HTML tags according to the actual situation, then injects the attack payload, such as *</title><script>alert("xss")</script>*. In addition to support the pseudo agreement in the tag attributes, the pseudo agreement can be injected into the attack payload, such as javascript: alert (" XSS "). Therefore, to the cross site scripting vulnerability testing, first of all the attacker need to inject a unique KEY into the target web page, secondly analyze the returned page for obtaining the KEY location, then construct the corresponding fuzz data and implement the injection, at last analyze whether the attack payload will occur at the specified position of the returned page, thereby to judge whether there are cross site scripting.

In addition, in view of part of the web application has set up different levels of filtering mechanism, generating the fuzz data needs to consider various forms of varieties (e.g., coding, case mix, inserting blank characters).

3) Improve the scalability: Separating the fuzz testing organization, scheduling, logic implementation from the fuzz data generation, the vulnerability detection; designing more than one module, and respectively implement the discovery according to different type of vulnerabilities.

C. Wfuzzer

We design and implement a web application fuzz testing tool named as Wfuzzer. The overall design is divided into the web application analysis module, the fuzz testing execution module and the test report generation module, the system framework is shown in figure 1.

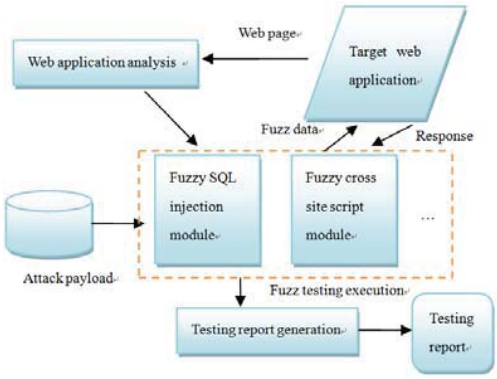


Figure 1. Architecture of Wfuzzer

The web application analysis module is used to scan and analyze the target web application to get the input vector (e.g., the URL and the web form), so as to determine the fuzz data injection point. The fuzz testing execution module can construct the fuzz data by different type of vulnerabilities and then send them to the target web application, based on the injection point provided by the web application analysis module. At the same time, the fuzz testing execution module can monitor the response of the target web application and determine whether there are vulnerabilities through the analysis of the response information. The test report generation module is used to provide the test results generated test report according to the fuzzy test execution module.

- Web application analysis module: A simple web crawler unit is contained, which can automatically identify all accepted user input data of the target web application, and store the results in the XML document. The implementation is based on the breadth-first strategy, providing some strategies to avoid analyzing the results which contain a large number of redundant data, and supporting to automatically skip some specific pages in the scanning process.
- Fuzzy testing execution module: This module is the core of the Wfuzzer, which realizing the fuzz data generation, injection, response analysis and vulnerability detection capabilities. Due to the different type of vulnerabilities are related to the different methods, and considering the scalability problem, Wfuzzer is designed that different type of vulnerabilities are corresponding to different sub-modules: all sub-modules are inherited from the same parent class. The sub-module only need to implement two abstract method *fuzzingGET()* and *fuzzingPOST()* of the father class, corresponding to GET and POST request execution fuzz testing.
- Test report generation module: WebFuzzer supports the test results derived for XML and HTML, which can directly show the vulnerability type, its location, the fuzz data caused the vulnerability, the vulnerability describe, the repair advice, the reference material and the other various information of the measured web application, with the purpose to help testers' further analysis and processing.

IV. EMPIRICAL STUDY

This section explains how we conducted our experiment. To empirically investigate the use of our technique, we performed three studies. Here we describe each study individually and provide the experiment results.

A. Study1

1) Objectives: The objectives of our initial study were to verify whether Wfuzzer can effectively detect kinds of vulnerabilities.

2) Procedure: We respectively designed the simple Web application which contained the corresponding bugs for each type of vulnerabilities. At the same time, we modified part of the software configuration items at the server-side. The purpose is to make the fuzz testing can focus more on the target web application itself and not affected by the other factors, thus benefiting the verification of Wfuzzer vulnerability discovering ability. Such as the change in php.ini configuration items: *display_errors = On, magis_quotes_gpc = Off, allow_url_fopen = On, allow_url_include = On, safe_mode = Off.* In addition, in order to increase the difficulty of vulnerability detection, in the experiment we also has considered the input validation and filtering of part of the input data.

3) Results: Experimental results are shown in the following (due to the paper length, only part of the vulnerability testing results is listed).

- SQL injection vulnerability

Testing URL：http://justfortest/sql_query.php
Request type: GET
Normal request：http://justfortest/sql_query.php?id=100
Fuzz data：
http://justfortest/sql_query.php?id=%BF%27%22%28&submit=submit

- Cross site scripting vulnerability

Testing URL：http://justfortest/xss_comment.php
Request type：POST
Normal request: comment=test & submit=submit
Fuzz data：
comment=%3Cscript%3Evar%28%22u0tj1i2nz4%22%29%3C%2Fscript%3E&submit=submit

- File containing vulnerability

Target URL：http://justfortest/include.php
Request type: GET
Normal request: http://justfortest/include.php?file=test
Fuzz data:
http://justfortest/include.php?file=http%3A%22F%2Fwww.google.com%2F

- Directory traversal vulnerability

Target URL: http://justfortest/path.php
Request type: GET
Normal request: http://justfortest/path.php?path=test
Fuzz data:
http://justfortest/path.php?path= ..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2Fetc%2Fpasswd

During the testing process, Wfuzzer output result is shown in figure 2. The experimental results show that Wfuzzer can effectively detect all supported vulnerability type of the target Web application, meet the basic function of the fuzz testing requirements.

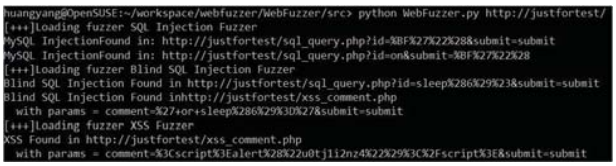


Figure 2. Wfuzzer output result

B. Study2

1) Objectives: The objectives of our second study are to verify Wfuzzer can execute the effective fuzz testing for a certain scale, relatively complete web application.

2) Procedure: We choose WackoPicko and a few typical web applications for testing. WackoPicko is designed by Adam Doupe and Marco Covaa, a fragile web application including safety hidden danger. It has the basic function (e.g., user register, login, messages, file upload, comments, search, etc.),

and the artificial hidden sixteen security vulnerabilities, covering cross site scripting, SQL injection, remote command, file containing vulnerability and etc. The main purpose is to be close to the real environment for testing all kinds of web security testing tool vulnerability discovery effect. The similar web applications include WebGoat, Web Security Dojo, Vicnum, OWASP InSecure Web App Project, etc.

3) Results: Wfuzzer has detected the WackoPicko vulnerabilities without omission and misinterpretation, listed in table II. The whole process of fuzz testing takes 256 seconds, which proves Wfuzzer has the higher efficiency. In addition, Wfuzzer has good stability without any collapse and abnormality during the testing process.

TABLE II. WACKOPICKO FUZZ TESTING RESULT

Vulnerability type	Vulnerability number
SQL injection	2
Cross site script	3
Remote command execution	1
File containing vulnerability	1
Directory traversal vulnerability	1

C. Study3

1) Objectives: The objectives of our third study are to validate Wfuzzer's automation degree and vulnerability detection effect.

2) Procedure: We choose a few open source web application fuzz testing tools (e.g., Wfuzz, WebFuzz, JBroFuzz) for the contrast experiment. The test object is Damn Vulnerable Web Application (DVWA) programs， a web application which contains various type of security vulnerabilities. The initial purpose is to train testers' vulnerability discovering ability and development personnel's understanding of all kinds of security vulnerabilities.

3) Results:

Table III lists the fuzz testing results of Wfuzzer and the other testing tools, when implementing the DVWA project. We find that Wfuzzer has effectively improved the accuracy of the vulnerability detection and reduced the misinformation and failing to report when implementing the fuzz testing on the target web application, compared with the other three tools.

TABLE III. TESTING RESULT COMPARISON

Tool name	SQL inject -ion	Cross site script	Remote command execution	File containing	Directory traversal	Testing time(s)
Wfuzzer	3	2	1	1	1	324
Wfuzz	0	1	0	1	0	853
WebFuzz	2	1	0	1	0	1052
JBroFuzz	2	1	1	1	0	1206

V. CONCLUSIONS

In this paper, we have presented an improved web fuzz testing technique and designed a tool (i.e., Wfuzzer). The automatic analysis, the fuzz data generation and the vulnerability detection method are respectively enhanced. Then we have performed an empirical study to evaluate the performance of the approach. The experiment results show certain advantages of using Wfuzzer in the degree of automation (mainly reflected in vulnerability quantity) and the vulnerability detection accuracy (mainly reflected in testing time).

There are also ways to extend this research work. The further study needs to increase the other type of vulnerability detection support, such as XPath injection, SOAP injection, and so on. And the vulnerability detection method in this paper is mainly based on the response analysis of the target web application. But to cross site scripting vulnerability, because of the diversity of the client script (e.g., JavaScript) call, the use of this method to detect the injected script implementation may have the missing problem. The further research needs to improve the vulnerability accuracy. For example, we can consider using the API Hooking method to increase the monitoring and analysis of the client browser behavior.

REFERENCES

[1] Oehlert, P, "Violating assumptions with fuzzing", IEEE Security and Privacy Magazine. v3 i2. 58-62.

[2] Symantec Corporation, Symatec Internet Security Threat Report, Trends for January-June 07, Volume XII, 2007.

[3] Hammersland, R, and Snekkenes, E, "Fuzz testing of web applications", 1-6. http://rune.hammersland.net/tekst/fuzzing_article.pdf, 2010.

[4] Miller, B. P., Fredriksen, L., & So, B, "An empirical study of the reliability of UNIX utilities", Communications of the ACM, 33(12), 22, December 1990.

[5] Kaksonen, R., Laakso, M., Takanen, A, "Software security assessment through specification mutations and fault injection", In: Proceedings of Communications and Multimedia Security Issues of the New Century, 2001..

[6] D.Aitel, "The advantages of block-based protocol analysis for security testing", Immunity Inc., February 2002.

[7] Sutton M, Greene A, Amini P, Fuzzing Brute Force Vulnerability Discovery, MA: Addison-Wesley. 2007:13-17.

[8] MARTIN, M., AND LAM, M, "Automatic generation of XSS and SQL injection attacks with goal-directed model checking", In Proceedings of the USENIX Security Symposium, pp. 31-43, July 2008.

A Petri Net Model Specification for Delivering Adaptable Ads through Digital Signage in Pervasive Environments

Frederico M. Bublitz[1], Lenardo C. e Silva[2], Elthon A. da S. Oliveira[3]
Saulo O. D. Luiz[2], Hyggo O. de Almeida[2], and Angelo Perkusich[2]
fredbublitz@uepb.edu.br, lenardosilva@copin.ufcg.edu.br, el7hon@gmail.com
saulo@dee.ufcg.edu.br, hyggo@dsc.ufcg.edu.br, perkusic@dee.ufcg.edu.br
[1]State University of Paraiba, [2]Federal University of Campina Grande, [3]Federal University of Alagoas

Abstract

Currently, there are environments in which public devices are positioned at strategic points such as elevators, hallways and showcases. Although these devices are visible to a lot of consumers, advertisers have little information about users who are present in the environment. However, such information may be acquired from the users' personal devices in a Pervasive Environment, where computing devices are seamlessly integrated to the users' life. In this way, Pervasive Advertising may be applied to maximize the relevance of the advertised products by adapting to the group of people in the environment. In this work, we present a Coloured Petri Net model to verify how the symbiotic use of the devices may be applied for creating a group-context, and how Pervasive Advertising impacts on the relevance of displayed advertisements.

1 Introduction

Advertising has a special role in marketing. It is the responsible for making the consumers aware of a product or service. It allows consumers to recognize their needs in the products and services announced, and it is a way of keeping alive the brand name in the mind of the consumers [1].

There are several media that can be used to promote a product or service. The printed media (e.g., newspapers, magazines and pamphlets) is one of the best known and widely used today. This kind of media also provides the use of a form of advertisement very familiar to us, the outdoor media, which had also adopted the use of neon signs. Another kind of media are the radio and the television, which make use of audio and video broadcast.

The pervasive computing paradigm [2] allows delivering effective ads. This is because the pervasive computing offers the possibility of adapting or customizing the ads according to users' context [3, 4]. An example is the use of the locations information to provide relevant ads [5], which is known as *Location-based Advertising*. Some other examples are the use of image recognition to adapt content [6]; the use of television for detecting the audience amount [7]; and the interactivity with a display as a means for customizing the ads [8].

Currently, devices such as smart televisions, are positioned at strategic points such as elevators, hallways and showcases in universities, airports, and malls. These devices are used with the purpose of promoting products and services, although acting as a form of entertainment to the people who frequent such environments. This form of ad serving is also known as *Digital Signage*.

Although these devices are able to reach a lot of consumers, the ads in such media have little relevance, because advertisers have little information from the users in the environment. For example, in a university it is common that topics or products related to education are announced in these displays. It is also common that classmates walk together. So, for a group of students from a computer science course an ad of a product that is related to the field of dentistry probably has no relevance. Hence, the audience is going to be small. It would be more interesting if it is shown an advertisement related to technology.

That is one of the reasons that motivate the customization of ads, thus offering products and services to the right customers, at the right time [9]. We believe that the current technological resources can enable this kind of media (i.e., the indoor media). In this way, it is possible to improve the advertised products and services by adapting the ad serving for the context of people who are in the environment. For this, it is necessary to have information about the context of the users. More precisely, it is necessary to know the location of users, and their profile, because this can help to know "who" is located nearby such a display. For this, it is necessary a treatment of the context of the users, not only

individually, but as a group.

This means that applications must be *group aware*, i.e., they must be capable of dealing with the context of a group of individuals as a new context. Notice that a group context is more than a collection of individual contexts, because it is necessary to deal with conflicts that possibly emerge from the composition. This is the first step in order to achieve the Pervasive Symbiotic scenario proposed by Narayanaswami [10], in which targeted ads are delivered on public displays by incorporating information from personal devices.

In this work, we present a Coloured Petri Net model to verify how the symbiotic use of the devices to create a group context impacts on the relevance of the displayed advertisements. Moreover we want to assure that the ads are exhibited whenever there are consumers nearby the display. Otherwise the display is turned off.

This paper is organized as follows: in Section 2, we present some related works on pervasive advertising. In Section 3, we present a theoretical background about Petri Nets. In Section 4, we descried a Coloured Petri Net model that enables the delivering of adaptable ads in pervasive environments. The verification of such model is presented on Section 5. Finally, we present some conclusions and future works in Section 6.

2 Related Works

There are a number of areas which are related to the work presented here, such as Internet Advertising [11] and Mobile Advertising [12]. But, in this section, we have presented works that are directly related to the use of public displays for ad serving.

Müller et. al. [6] described in their work that, because of the technological advances, there was a reduction in the hardware cost, allowing the popularization of electronic displays and their use in public places as a means of replacing the media on paper. In the environments where displays are installed, there are sensors picking up signals from people who are in transit. The content to be displayed is selected based on these signals.

Roges et. al. [13] described the *BlueScreen*, which consists of a system that tracks and detects users by means of *bluetooth* devices. This system uses a decentralized mechanism for a multi-agent bidding and that can efficiently allocate specific time within each exposure of selected advertisements. Each listing is represented by an individual advertising. The agent maintains a history of users who have been exposed to some type of advertising, allowing to know exactly to whom the ad was displayed.

Yin et al. [14] evaluated the effectiveness of the digital signage advertisement. Face recognition was applied to determine whether a consumer was watching advertisements. The proposed solution was only used for tracking the viewing duration, but the ads were not customized according to the face detection to allocate the ads.

More recently, Want and Schilit [15] emphasized that Digital Signage will support communication in the 21^{st} century. According to those authors, to gain a perspective on interactive Digital Signage, it is necessary to consider the vision of ubiquitous computing. They have also stressed that the opportunity for interfacing with mobile devices may lead to useful personalization of information presented on digital signs. This aligns with the purpose of this work.

Although these works represent significant advances, the literature lacks on solutions that deal with the group context, resulting in the decreased relevance of the advertisements. Moreover we need to increase the autonomy of the display by turning it off when the place is empty.

3 Coloured Petri Nets

To demonstrate that the above challenges can be overcame we created a Coulored Petri Net model to verify the effectiveness of the delivered advertisements and ensure some properties. In this section we give an overview about what is a Coloured Petri Net.

Coloured Petri Nets (CPNs) are a high-level Petri Nets [16] that make it possible to represent complex data types. A CPN specification is used to model concepts such as concurrency, conflicts, synchronization and resources sharing [17]. Figure 1 shows a simple CPN model and its main elements, which are:

- ◇ place: represents a condition, an activity or a resource;
- ◇ transition: represents a certain event;
- ◇ arc: indicates the input/output place of a transition.
- ◇ token: represents the availability of a certain resource;

A state of the model, called marking, is represented by a "picture" of the net at a certain moment: the amount of tokens each place has. The firing of a transition generally changes the net state. Such firing can happen if the transition is able to do so. Basic transition firing rules are:

- ◇ a transition will be able to fire in the case of each one of all its input places has, at least, as many tokens as the weight of the arc that connects them;
- ◇ the firing of a transition removes a certain quantity of tokens from each one of its input places (according to the weights of the arcs that connect them);

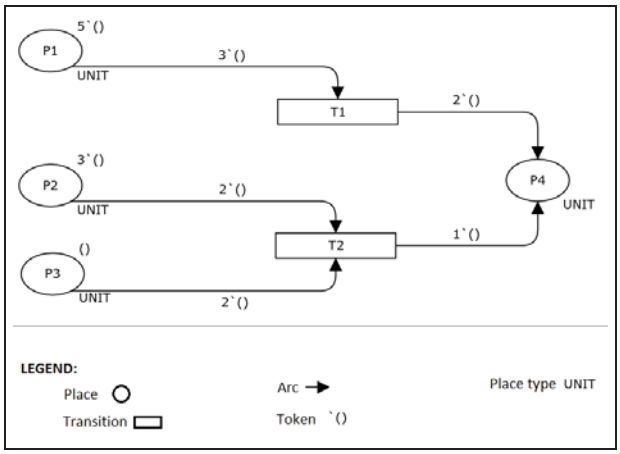


Figure 1. A simple CPN model and its elements.

◇ the firing of a transition produces a certain quantity of tokens to each one of its output places (according to the weights of the arcs that connect them).

In the net illustrated in Figure 1, only the *T1* transition is enabled to fire. Its input place has five tokens, what overcomes the weight of the arc that connects them (three). The firing of this transition removes three tokens from the *P1* place and puts two tokens in the place *P4*. In the same figure, the transition *T2* is not able to fire. The marking of the place *P2* overcomes the weight of the arc that connects it to transition *T2*, but the same does not happen to the place *P3* that has no token and is also an input place of that transition.

In a CPN model, all the net inscriptions are written in a programming language that is a variant of SML named CPN/ML [1].

4 Coloured Petri Net Model

In this section we expose the Coloured Petri Net Model specification of the system. This model was conceived to verify the following hypothesis:

Null hypothesis - H_0: Our model bring advertisements equal or less relevant than the random model. The relevance, represented by the symbol Φ, in this work is measured as match between the student and the advertisement. Formally the null hypothesis is represented by:

$$H_0 : \Phi_{this} \leq \Phi_{other} \quad (1)$$

Alternative hypothesis - H_1: Using just a few information about the people in the vicinity of a display is useful to determine the group-context and display more relevant advertisements. More precisely we use the location, i.e., if the student is close to the display; the gender; the age; and the course. Formally:

$$H_1 : \Phi_{this} > \Phi_{other}. \quad (2)$$

We also desire to guarantee that our system only display advertisements if there is people to watch them, that is:

1. Always that the place has students advertisements will be displayed;
2. When the place is empty the display is turned off.

4.1 Scenario

This petri net model is used to describe the ad allocation according to students' movements inside the university. The idea is that the students presence is detected through their *bluetooth* devices. These devices were previously registered and their MAC address associated with a student. When the presence of the students are detected a new context, called group context, is formed and then the system processes what ads are more suitable to the group of students in the vicinity of the display, and the ads are exhibited, as shown in Figure 2.

Figure 2. Scenario: the advertisements are exhibited according to people in the vicinity of the display.

4.2 Model Specification

The developed Petri Net Model is demonstrated in Figure 3. In the remaining of this section we are going do describe it.

The main colors defined to the model are presented in the Listing 1. For instance, the Device is represented by the color `colset Device` that contains its MAC; and the student is represented by the color `colset Student` and its attributes are the id, course, gender, age, and device.

[1] http://wiki.daimi.au.dk/cpntools-help/cpn_ml.wiki

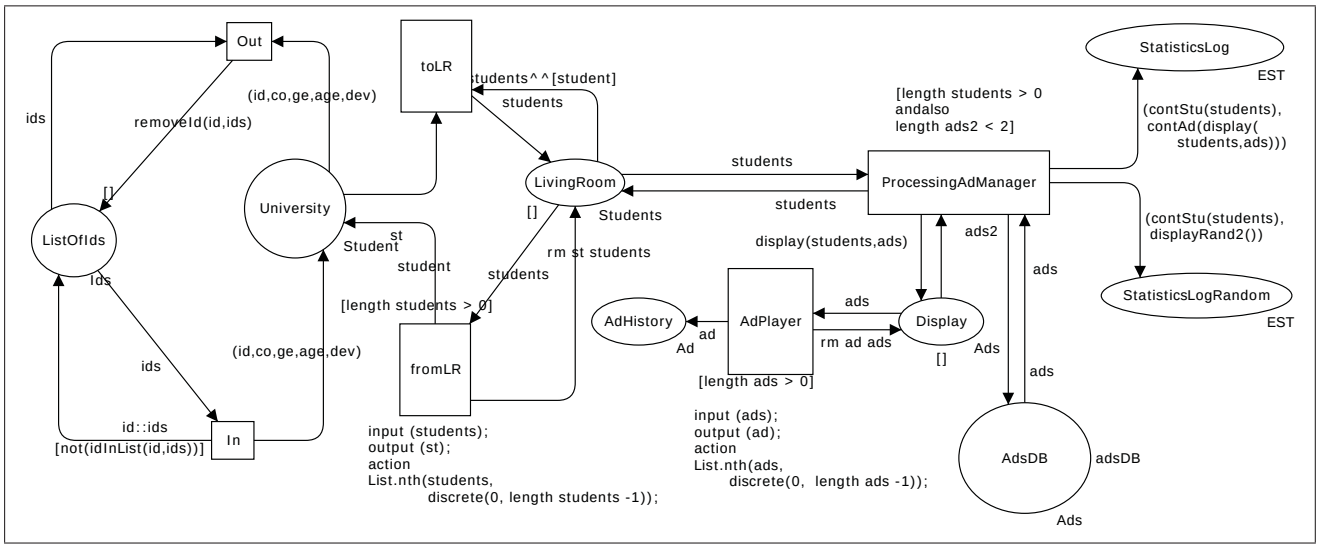


Figure 3. Coloured Petri Net Model describing the ad allocation according to the students movements.

Listing 1. Declarations of some of most important colors.

```
colset Device = MAC;
colset Gender = with m | f;
colset Student = product UserId*Course*Gender*Age*Device;
colset Ad = product Course*Gender*AgeMAX*AgeMIN*STRING
colset Ads = list Ad;
colset Students = list Student;
```

When the net simulation starts, there are no students in the system. In the places *ListOfIds* and *Living Room*, for example, there is an initial marking to assure that there are no students.

The place *ListOfIds* contains a list of identifiers of the devices in the range of university (the Listing 2 shows the ids generation). The transitions *In* e *Out*, when enabled, add or remove students from the *University* place, which represent any place inside the university where there is no display. The guard in the *In* transition is used to avoid that two different students have the same *id* in place *ListOfIds*. On the other hand, the inscription in the output arc of *Out* transition free the student, by removing its *id*, meaning that the student get out of the university.

Listing 2. Function used to generate Ids.

```
fun idInList(id,nil) = false
  | idInList(id,h::t) = if (id = h) then true
                        else idInList(id,t);
```

The simulation of a student entering and exiting in the places available at university, in this case *Living Room*, is done by the transitions *toLR* and *fromLR*, respectively. After some time, the net marking contains some students in the places *University* and *LivingRoom*.

The allocation of ads is done when the transition *ProcessingAdManager* fires. In order to fire, this transition requires that there are students in the *LivingRoom* and that the *Display* is turned off or it have at most one ad being exhibited. This is necessary because the *ProcessingAdManager* fires groups of 3 to 5 ads according to the group context in order to give time for a new processing of group context. This was done due to the fact that we are using Bluetooth devices, that requires a delay for device discovery. These requirements are verified by the guard `[length students > 0 andalso length ads2 < 2]`.

When this transition fires, the *SML* function `display (students,ads)` gets the list of students (see Listing 3); process the group context; query the ad database represented by the *AdsDB* place; and a list of ads (represented by the token "Ads") is displayed on the *Display* place. In this way, the *ProcessingAdManager* transition enables the delivery of personalized content for available spaces at university.

The group context is treated by each characteristic. In Listing 3 the ads are selected according to the courses.

Listing 3. Function to select ads according to the group context.

```
(*allocate ads to the display according to the
group of students*)
fun display(nil,nil) = []
| display(st,ad) = getAds(0,#1(contCourseComp(st)),ad)
                  ++getAds(3,#2(contCourseMed(st)),ad)
                  ++getAds(7,#3(contCourseLaw(st)),ad)
                   ...;
(*return a list of advertisements*)
fun getAds(x,y,nil) = []
| getAds(x,y,h::t) = if x = 0 then getNAds(y,t)
                     else getAds(x-1,y,t);
```

In this sense the the `getAds` function return the amount of advertisements proportional the amount of students of each course.

Notice that the transition *ProcessingAdManager* has high priority (P_HIGH), this means that whenever they are enabled they should fire. This is a way of assure that the Display will not to become off when there are people in the *LivingRoom*.

The *StatisticsLogRandom* and *StatisticsLog* places are used as data collectors based on the firing of *ProcessingAdManager* transition. Both places maintain the history of ad delivery, however the first intended to random approach and the last inherent to the group aware model.

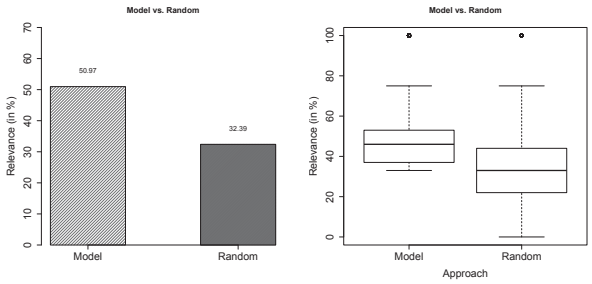
The *AdPlayer* transition is responsible for consuming the tokens of *Display* place, simulating the exhibition of an ad. The place *AdHistory* acts as historical, but the emphasis is, respectively, in chronological order of what ads were fully displayed to students and which were aborted before displaying as a consequence of output students of the place frequented.

Although this model is limited to only one place frequented at university, it is possible to extend it to incorporate other places, such as *library, restaurant* and *gymnasium*, following the same logic presented.

5 Model Verification

The verification of the CPN model was made from the analysis of the state space and based on *reachability* properties [18]. As this is an exhaustive search technique, it allows us to answer certain questions that ensure the reliability of the proposed model for the scenario under study.

The first aspect that we want to verify is concerning the relevance of the exhibited ads for the students in the place. Regard this issue our model obtained a level of relevance better then the random selection, as showed in Figure 4.



(a) Average relevance. (b) Distribution of relevance.

Figure 4. Ad Relevance: model vs. random.

This means that negotiation model is most efficient in finding the appropriate ads for consumers, validating our **Alternative Hypothesis -** H_1 (Section 4, Equation 2).

Another aspect that we want to verify is that the CPN model is designed in accordance with the specification of requirements for proper operation of the ads delivery system, the following properties were observed:

- ◇ **PI:** For all paths, there is an immediate successor state, to the current state, so that the place *LivingRoom* is not empty and at least one ad appears on the display?
- ◇ **PII:** When *LivingRoom* is empty the display is turned off?

The ASK-CTL expressions used for this verification are represented in Table 1.

Table 1. Model Checking Results.

	ASK-CTL EXPRESSION	RESULT

PI	`fun PropertyI n = (Mark.Model'LivingRoom 1 n <> []) andalso (Mark.Model'Display 1 n <> []);`
	
	`val myASKCTLformulaI = FORALL_NEXT( NF("PropertyI:",PropertyI));`
	
	`eval_node myASKCTLformulaI InitNode;`
		TRUE

PII	`fun PropertyII n = (length (hd (Mark.Model'LivingRoom 1 n)) = 0) andalso (length (Mark.Model'Display 1 n) = 0);`
	
	`val myASKCTLformulaII = EXIST_UNTIL( TT,NF("PropertyII:",PropertyII));`
	
	`eval_node myASKCTLformulaII InitNode;`
		TRUE

Such questions are formulated using a library that implements CTL-like temporal logic called ASK-CTL, which additionally contains a model checker [19]. From the model checking results for both properties we can verify that they are `true`. This means that the display only plays advertisements when there are students in the environment.

6 Final Remarks

In this work, we described a Couloured Petri Net model concerned with functional evaluation of a context-aware advertising approach. In this approach, specific ads are presented on a display based on the group of individuals which are in its vicinity. We demonstrate a methodology to obtain better and adapted ads to a specific group of students, by means of simple context characteristics, such as gender, age, and course, thus increasing the ads relevance.

The Coloured Petri Nets model was built to capture the behavior of the system. By means of this Petri Net model, it

was possible to verify that Pervasive Advertising may be successfully applied to maximize the relevance of advertisements. Model checking was applied on the constructed model to automatically verify some key properties, such as the ads delivering only when there are users nearby. Otherwise, the display is blanked, thus saving power and the advertiser investment, because only the exhibited ads are paid.

As future work, we are developing a model of a platform that captures images from a video camera to identify the users. In a first moment, it may estimate the age and the gender of the users. In this way we will be able to exhibit customized ads for most dynamic and heterogeneous places.

References

[1] E. Bernays, *Propaganda*. Horace liveright, 1928.

[2] M. Weiser, "The Computer for the 21st Century," *Scientific American*, vol. 265, pp. 66–75, September 1991.

[3] F. Bublitz, H. Almeida, A. Perkusich, E. Loureiro, E. Barros, and L. Dias, "An Infrastructure for Developing Context Aware Applications in Pervasive Environments," in *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, (New York, NY, USA), pp. 1958–1959, ACM, 2008.

[4] F. Bublitz, H. O. de Almeida, and A. Perkusich, "A Context Ontology Model for Pervasive Advertising: a Case Study on Pervasive Displays," in *Proceedings of the 24th International Conference on Software Engineering & Knowledge Engineering (SEKE'2012)*, pp. 426–431, Knowledge Systems Institute Graduate School, 2012.

[5] A. Kupper, *Location-Based Services: Fundamentals and Operation*. Wiley, October 2005.

[6] J. Muller, J. Exeler, M. Buzeck, and A. Kruger, "ReflectiveSigns: Digital Signs That Adapt to Audience Attention," in *Pervasive '09: Proceedings of the 7th International Conference on Pervasive Computing*, (Berlin, Heidelberg), pp. 17–24, Springer-Verlag, 2009.

[7] K. Maeda, M. Nishi, T. Yoshida, K. Suzuki, and H. Inoue, "Digital Signage with Audience Detection Using TV Broadcasting Waves," in *Applications and the Internet (SAINT), 2010 10th IEEE/IPSJ International Symposium on*, pp. 225 –228, 2010.

[8] A. Erbad, M. Blackstock, A. Friday, R. Lea, and J. Al-Muhtadi, "MAGIC Broker: A Middleware Toolkit for Interactive Public Displays," in *Pervasive Computing and Communications, 2008. PerCom 2008. Sixth Annual IEEE International Conference on*, pp. 509 –514, march 2008.

[9] G. C. Bruner and A. Kumar, "Attitude toward Location-Based Advertising," *Journal of Interactive Advertising*, vol. 7, no. 2, pp. 3–15, 2007.

[10] C. Narayanaswami, D. Coffman, M. C. Lee, Y. S. Moon, J. H. Han, H. K. Jang, S. Mcfaddin, Y. S. Paik, J. H. Kim, Lee, J. W. Park, and D. Soroker, "Pervasive Symbiotic Advertising," in *HotMobile '08: Proceedings of the 9th workshop on Mobile computing systems and applications*, (New York, NY, USA), pp. 80–85, ACM, 2008.

[11] Z. Wendan and W. Dingwei, "Study on Internet advertising placement problem," vol. 3, pp. 1798 –1801, jan. 2010.

[12] D. Drossos and K. Fouskas, "Mobile Advertising: Product Involvement and Its Effect on Intention to Purchase," pp. 183 –189, jun. 2010.

[13] A. Rogers, E. David, T. R. Payne, and N. R. Jennings, "An Advanced Bidding Agent for Advertisement Selection on Public Displays," in *AAMAS '07: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, (New York, NY, USA), pp. 1–8, ACM, 2007.

[14] K.-C. Yin, H.-C. Wang, D.-L. Yang, and J. Wu, "A study on the effectiveness of digital signage advertisement," in *Computer, Consumer and Control (IS3C), 2012 International Symposium on*, pp. 169 –172, june 2012.

[15] R. Want and B. Schilit, "Interactive digital signage," *Computer*, vol. 45, pp. 21 –24, may 2012.

[16] T. Murata, "Petri nets: Properties, Analysis and Applications," *Proceedings of the IEEE*, vol. 77, pp. 541–580, April 1989.

[17] K. Jensen, *Coloured Petri nets: basic concepts, analysis methods and practical use, vol. 2*. London, UK: Springer-Verlag, 1995.

[18] K. Jensen and L. M. Kristensen, *Coloured Petri Nets: Modelling and Validation of Concurrent Systems.* Springer-Verlag, 2009.

[19] S. Christensen and K. H. Mortensen, *Design/CPN ASK-CTL Manual, Version 0.9*. University of Aarhus, 1996.

An Approach for Analyzing Software Specifications in Petri Nets

Junhua Ding
Department of Computer Science
East Carolina University
Greenville, NC, USA
dingj@ecu.edu

Dianxiang Xu
National Center for the Protection of the Financial Infrastructure
Dakota State University
Madison, SD, USA 57042
dianxiang.xu@dsu.edu

Jidong Ge
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing 210093, China
gjd@nju.edu.cn

Abstract — **In this paper, we present an approach for analyzing software specifications in Predicate/Transitions (PrT) nets. The approach is designed as a two-phase process to ensure it is not only practical but also rigorous for analyzing formal software specifications in PrT nets. In the first phase, a PrT net model is analyzed using model based testing technique with tool MISTA. Then the important properties defined in temporal logic formulas for the PrT net model is further checked using model checking technique with tool NuSMV to ensure the correctness of the model. The errors found in the first phase and the counter examples generated in the second phase can help developers to find problems in the model. The analysis process is illustrated by analyzing an Alternating Bit Protocol (ABP), and problems found in one ABP model in PrT nets have demonstrated the effectiveness of the approach.**

Keywords- model-based software testing; model checking; Software Specification; Petri net

I. INTRODUCTION

Formal specifications offer a solid foundation for understanding and implementing software systems. Many formal specification languages such as VDM, Z, and Petri nets have been designed for specifying software systems, and many verification techniques like theorem proving and model checking have been developed for analyzing software specifications. However, a rigorous and also practically useful analysis technique for analyzing formal specifications is rare. In this paper, we introduce a rigorous analysis approach for analyzing software specifications in Predicated/Transition (PrT) nets. The approach is developed through integrating model-based software testing and model checking technique in a two-phase analysis process that analyzes software specifications by gliding from informal to formal analysis. In this approach, the formal specification is first simulated, verified and tested for a set of test adequacy criteria. If any error is found, the formal specification will be iteratively analyzed until the error is understood and the specification is updated. As soon as the formal specification passes the first phase testing, it is again checked using model checking tools against a set of properties defined in temporal logic.

The main contribution of this paper is due to a two-phase rigorous and practically useful approach for analyzing software specifications. Through analyzing some errors found in a case study, we believe it is important to provide an easy to use technique such as simulation and testing for analyzing software specifications when they are still in the early development phase. In addition, we also understand testing is not enough to ensure the correctness of a software specification. Model checking is necessary for verifying a software specification in the later development phase. In our approach, the model-based testing assists ones to understand the specification, to check simple assertions and test special scenarios for building correct software specification and reducing the model checking tasks, and the model checking ensures the correctness of the specification. We have performed a case study on an Alternating Bit Protocol (ABP) protocol, and the result has shown the effectiveness of the proposed approach.

The rest of this paper is organized as follows: Section 2 presents a brief introduction to PrT nets, model-based software testing and its tool, and symbolic model checking and its tool. Section 3 introduces the proposed analysis approach that integrates model-based testing with model checking for analyzing software specifications in PrT nets. Section 4 reviews the related work, and section 5 concludes this paper.

II. BACKGROUND

A. PrT Nets

Predicate/Transition (PrT) nets, a high level Petri net, are used for specifying software systems. The formal definition of PrT nets used in this paper is same as the one defined in [15]. A PrT net is a tuple $(P, T, F, \Sigma, L, \varphi, M_0)$, where: P is a finite set of predicates (first order places), T is a finite set of transitions and F is a flow relation. (P, T, F) forms a directed net. Σ is a structure consisting of some sorts of individuals (constants) together with some operations and relations. L is a labeling function on arcs. φ is a mapping from a set of inscription formulae to transitions, and M_0 is the initial or current marking.

Figure 1 shows a PrT net model for the 5 dining philosophers' problem, which includes transitions *Pickup*, and *Putdown*, places *Phi*, *Chop* and *Down*. Places *Phi* and *Chop* include tokens that are nature numbers representing philosophers or chopsticks, and each token in place *Down* includes represents a philosopher and his/her two chopsticks. Transition *Pickup* has two input places *Phi* and *Chop*, and one output place *Down*. The guard condition in transition *Pickup* is defined based on the relation between the tokens in place *Phi* and *Chop*. The guard condition in transition *Putdown* is defined based on the relation between the tokens in place *Phi* and *Chop*.

B. Model-based Testing and MISTA

MISTA [14] 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) [15] for specifying test models so that complete tests can be automatically generated. It also provides a language for mapping the elements in function nets to implementation constructs, which makes it possible to convert the model level tests into program level tests that can be executed against the system under test. MISTA includes several important components: model editor, model parser, simulator, reachability analyzer, test generator, and test code generator. MISTA support the step by step execution and random execution of a function net, and the execution sequences and token changing in each place are visualized for inspection. The test generator generates model level tests (i.e., firing sequences of the function net) according to a chosen coverage criterion such as transition coverage or state coverage. The tests are organized and visualized as a transition tree. MISTA supports a number of coverage criteria for test generation from function nets, including reachability graph coverage, transition coverage, state coverage, depth coverage, and goal coverage. Test code generator generates test code in a chosen target language like Java or C++ from a given transition tree [14].

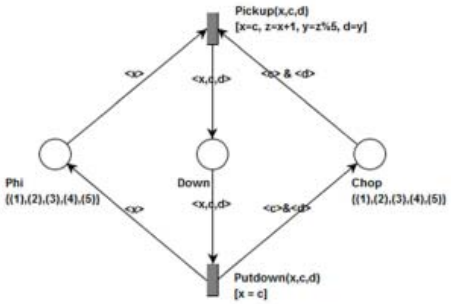


Figure 1. A PrT nets model for dining philosophers

C. Model Checking and NuSMV

NuSMV [11] is a reimplementation and extension of symbolic model checker SMV for checking finite state systems against property specifications in temporal logic. To verify a finite state system in NuSMV, it has to be described in NuSMV input language. NuSMV then translates the description into a system model and uses the symbolic model checking algorithms to check if a given specification in temporal logic is true or false. In the latter case, a counterexample is given when possible. A NuSMV program consists of a definition of a finite state transition system and a list of properties written in temporal logic formulas. A transition system is defined in terms of a state space, a transition relation, and a set of initial states. Since NuSMV is intended to describe finite state systems, the only data types in the language are finite ones such as Booleans, scalars and fixed arrays. The complete syntax of NuSMV language is described in [11]. A PrT net model has to be converted in to NuSMV model before it is able to be checked using NuSMV, and the properties to be checked have to be written in NuSMV format. The translation of a PrT net model to SMV input has been defined in [5], which is also applied to NuSMV.

III. Analyzing PrT Nets

In this section, we are going to discuss an approach for rigorously analyzing a PrT net model using model-based software testing supplemented with model checking. In order to illustrate the basic idea and the process of the two-phase analysis approach, we model and analyze an ABP model.

A. ABP

ABP is a protocol for reliable transmission of messages over unreliable communication channels that may lose or corrupt transmission messages. The protocol consists of a *sender*, a *receiver*, and two channels: *message channel and acknowledgement channel*. The main goal of ABP is to ensure that the receiver will eventually deliver an accepted message from the sender to the receiver. The channels can detect the lost or corrupted messages. Messages are sent from the sender to the receiver via the message channel. Each message from the sender includes a data part and a one-bit sequence number. The receiver includes a one-bit acknowledgement character for checking the sequence of a received message and sending acknowledgement to the sender. If a message is lost or corrupted, the *ack* sent to the sender through the acknowledgement channel will not match to the expected sequence number, and then the message will be resent. If an *ack* message is lost or corrupted, it will be resent. Next message only can be sent from the sender to the receiver until an expected *ack* is received by the sender. The protocol guarantees that (1) an accepted message will eventually be delivered, (2) an accepted message is delivered only once, and (3) the accepted messages are delivered in order. We build a PrT net model for ABP without a timeout mechanism, which assumes that the channels may lose or corrupt but not duplicate messages and the lost or corrupted messages are detectable. A PrT net model is defined in Figure 2. Although the PrT net model of ABP defined in Figure 2 includes 4 small nets, these small nets have to be connected together via the common places for execution and analysis. The model defined in Figure 2 may have some issues, which we will discuss in the following sections.

B. The Approach

The approach we introduce in this paper for analyzing a PrT nets is a two-phase analysis approach. First a PrT net is checked using the model-based testing tool MISTA, and then the PrT net is further verified using the symbolic model checker NuSMV. The first phase of analysis using simulation and testing can help model developers understand the model in details and find problems that could be skipped by model checking. As soon as a PrT model is passed through the first phase checking, its properties are verified again using model checker NuSMV to ensure the correctness.

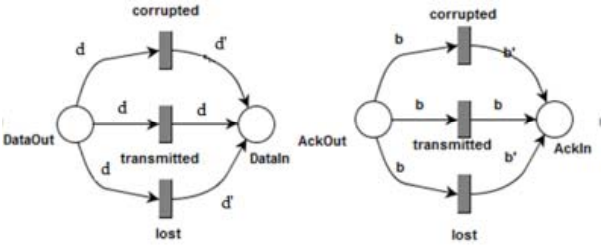
The analysis functions in MISTA include simulation, model level test generation, and verification of goal reachability, deadlock states and simple assertions. The simulation is used to execute the model with random inputs to check its running at different scenarios. The simulation process can assist developers to detect some unreasonable behaviors in the model

and give them an intuitive understanding of the model. The verification of goal reachability is used to check the reachability of goal states defined in the PrT net model. If a goal is known to be reachable (or unreachable), but the verification reports that it is unreachable (or reachable), then the PrT net or the set of initial states must be specified incorrectly. A goal state is not limited to a specific marking. Generally, it is specified by a logical formula $P \rightarrow Q$. The reachability analysis of $P \rightarrow Q$ checks to see if there exists a reachable marking that satisfies $P \rightarrow Q$. Verification of an assertion $P \rightarrow Q$ is to check if the given assertion is satisfied by all states reachable from the given initial states. Generally, verification of goal reachability aims at the analysis of existential properties ("*there exists*") whereas verification of assertions targets on analyzing universal properties ("*for all*"). Verification of deadlock states is to check if the given PrT net can reach any deadlock/termination state under which no transition is enabled. If the verification result is different to our expectation, the given net might be specified incorrectly. As soon as the simulation and verification have been done, one or more test coverage criteria are chosen for generating model level tests, which are guaranteed to be adequate for each selected criterion. Complex execution scenarios of the PrT net model can be tested with selected tests using MISTA. The model level tests can be used for generating program level tests that are used for testing the model implementation.

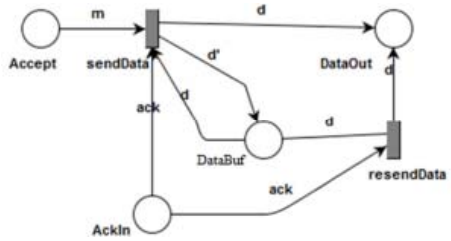
In order to model check a PrT net model using NuSMV, we need to translate the model into a NuSMV program and define interesting properties in temporal logic. It is well known that a finite state transition system is a special case of a Petri net [10]. It is also well known that a state based verification approach is adequate to ensure general safety and liveness properties [8]. Therefore, restricting a Petri net's behavior into interleaved executions (*i.e*. firing one transition at each step) by translating a Petri net into a state transition system does not affect the satisfiability or validity of a property specification. To ensure the system containing only finite states, we require the boundedness of all places in a PrT net to be checked [5]. Some properties are not feasible to be tested by MISTA, but they can be checked using NuSMV. Interesting properties that haven't been checked in the first phase such as temporal properties ***AF*** ("*future*"), ***AU*** ("*until*") and ***EFAG*** ("*exist future for all*") are needed be checked using NuSMV. In practice, all interesting properties of a PrT net model should be re-checked using model checking to ensure the correctness. If a false is found for a given property in the model, the counterexamples generated from the model checking will be used for debugging the PrT net model using MISTA thanks to the visualization of its execution process.

C. Model-based Testing of PrT Nets

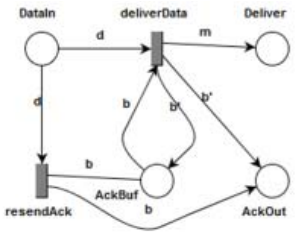
As soon as a PrT net model is created and successfully compiled in MISTA, it is executed with random inputs (*i.e*., initial markings) to help developers to understand the model and detect easily found problems. If the simulation result is acceptable, we can start the verification of the goal reachability, assertions and deadlock states. After that, a set of tests can be generated based on selected testing coverage criteria, and finally selected tests are performed to test the model. Here we discuss the testing process through testing the PrT net model of ABP shown in Figure 2.



2.1 PrT net models for ABP channels (left: message channel; right: acknowledgement channel), where *d* is a message including a datum part *m* and a sequence bit *b*, *d*' is a corrupted or lost message, *b* is an ack bit, and *b'* is a corrupted or lost ack.



2.2 PrT net models for ABP Sender, where *m* is a datum to be sent, *d* is the message to be sent out, *d'* is a message with alternating bit to *d*, *ack* is an ack bit. Guard for *sendData* is (*d.ack = 1 –ack, belongsTo(ack, 0, 1), ack = d'.ack*), and guard for *resendData* is (*ack <> d.ack*).



2.3 PrT net models for ABP Receiver, where *d* is a message, *m* is the datum part of *d*, *b* is an ack bit, and *b'* is the alternating bit of *b*. Guard for deliverData is (*b.ack = d.ack, b'.ack = 1-b.ack*), and guard for *resendAck* is (*d.ack <>b.ack*).

Figure 2. A PrT net model for ABP

First, execute the PrT net model for ABP with some valid initial markings using MISTA. For example, we assign an initial marking for the PrT net in Figure 2 as: *Accept*(*"message1"*), *DataBuf*(*1, ""*), *AckIn*(*1*), *AckBuf*(*1*). Multiple transitions can be enabled under certain marking, but only one transition can fire each time. Which transition will be selected from multiple enabled transitions to fire is nondeterministic. For example, when place *DataOut* has token (*1, "message1"*), transitions *corrupted*, *lost* and *transmitted* are enabled. MISTA allows users to select an enabled transition for firing or let the system to randomly choose one for firing during the running. MISTA also allows users to start the random simulation of the whole model. During the random simulation of the ABP model, we found the execution was started and continued for several rounds, and then it stopped running without delivering the accepted message, which is conflict to the ABP specification. Something must be wrong in

the model. The initial marking was fairly simple, and we could not find any problem in it. We executed the model step by step with manually controlling the execution process to look for problems in the model. A normal execution scenario for an ABP should be like this: (1) when one message is accepted in the sender, a sequence number is attached to the message to form a datum package; (2) as soon as the package is transmitted, the sequence number of the transmitted message is compared to the expected sequence number in the receiver; (3) if the two sequence numbers are identical, the message is delivered, and the sequence number in the receiver is flipped and stored in a buffer for checking next message; (4) after that, an acknowledgement message *ack*, which is the previous expected sequence number in the receiver, is sent to the sender; (5) as soon as the *ack* message is transmitted to the sender, the sequence number in the sender is flipped, and the next message is ready to go.

The normal scenario can be easily checked. We started the manual play with the same initial marking defined above. In the beginning, only transition *sendData* was enabled. When it was fired, transitions *corrupted*, *lost* and *transmitted* in the message channel were all enabled. Transition *transmitted* was selected for firing. After that, we found transition *deliverData* was enabled, and it was the only enabled transition. Firing transition *deliverData* would deliver the message to place *deliver*, transmit acknowledgement message *ack* (*i.e.* put the *ack* token in place *AckOut*), and flip the sequence number in place *AckBuf*. Now, transitions *corrupted*, *lost* and *transmitted* in the acknowledgement channel were all enabled. Transition *transmitted* was selected for firing, and *ack* was successfully delivered to place *AckIn,* which terminated the execution since no any transition was enabled under the marking. The simulation of normal scenario of delivering message demonstrated that a message had a chance to be successfully delivered in the model, and it narrowed the problem of the model on handling the corruption of a message. Therefore, we simulated the scenarios with corruption of a message or an *ack*. In order to find subtle errors, we simulated the combination execution scenarios that combine transmitted, lost, and corrupted messages with transmitted, lost, and corrupted *acks*, which generate 9 different execution scenarios for one trip of execution. The first problem we found was when a message was unsuccessfully delivered, and further the *ack* from receiver was corrupted during the transmission, then the *ack* had to be resent. However, the resending of *ack* had to be gone through the sender side (see Figure 2.1, where the output place of transitions lost, corrupted of the acknowledgement channel is in the sender side), and then the resending *ack* from the receiver to the sender couldn't be accepted in the sender side due to the expected sequence number in the sender had been updated, which caused inconsistency of the sequence numbers in both sides. Although the problem could be resolved by adding new parameters into the places, it was unreasonable to let the sender side to handle corrupted *ack* message from receiver. Although we also can fix above issue with adding more complex constraints into transitions, we found duplication of messages could still happened due to other problems. Therefore, we created a new model that could properly handle the message corruption issue. Figure 3 shows an updated PrT net model for ABP, which can pass the simulation testing. We added 8 messages in place *Accept*, and these messages were sequentially delivered to place *Deliver* without duplication even when messages or *acks* were corrupted or lost during the transmission.

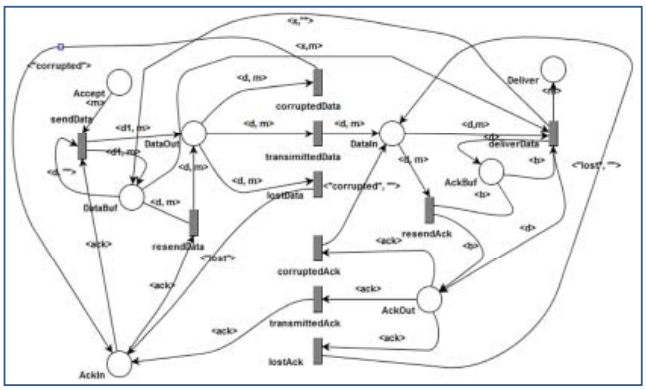


Figure 3. An updated PrT net model for ABP. The sender handles the resending of corrupted or lost message directly, and the receiver handles the resending of the corrupted or lost ack itself. The transitions and labels defined in this figure are slightly different to those in Figure 2 for technical convenience, but they have exactly same meaning.

Second, verify the reachability of goal states, assertions and deadlock states using MISTA. Under the initial marking *Accept*(*"message1"*), *Accept*(*"message2"*), *DataBuf*(*1,""*), *AckIn*(*1*), *AckBuf*(*1*), we checked the reachability of several goal states. Accepted messages *message1* and *message2* could be delivered to place *Deliver* (i.e., *GOAL Deliver("message1)*, *Deliver ("message2"*)); *AckIn*(0) was reachable, which means an acknowledgement message *ack* for successfully delivering an accepted message was acknowledged. Considering the reachability of above goal states, we concluded that when an acknowledgement of delivering of a message could cause the sending of the next message in the sender. We also checked whether an undesirable state was reachable in the model. If an undesirable state was reachable, then something must be wrong in the model. For example, messages should not be duplicated during the transmission, which could be checked via checking a goal state with duplicated messages in place *Deliver*. MISTA is able to check deadlock states, reachability of transitions and verify simple assertions of a PrT net. The only deadlock state in the ABP model under above initial marking was the state when all messages had been delivered, and all transitions are reachable. The assertions like *Deliver("message1")* → ***not*** *Accept("message1")* was hold in the model defined in Figure 3, which means *message1* will be removed from place *Accept* when it is delivered. If MISTA found an assertion that was not hold in a model, it generates counterexamples to show how the assertion was violated in the model, which can help us to understand and debug the model.

Third, generate adequate tests according to selected test coverage criteria using MISTA. For the PrT net model defined in Figure 3, we can generate model level tests covering all states, all executable paths, all assertions, goal states, and others. Since complex scenarios are not feasible to be checked during simulation or verification using MISTA, these complex scenarios can be rigorously tested with model level tests thanks to the executable capacity of PrT nets. The model level tests are

also used for generating program level tests via mapping the model to its corresponding program. The program level tests will be used for testing the model implementation. Figure 4 is a snapshot of the generated tests covering all paths for the ABP model defined in Figure 3.

Figure 4. Generated tests for ABP model defined in Figure 3

D. Model Checking PrT Nets

In order to ensure the correctness of a PrT net model, we use model checking tool NuSMV to check the behavior model specified in PrT nets against the important properties defined in temporal logic. The main goal of model checking the ABP model is to verify the three properties that should hold in a PrT net model for ABP:

(P1) *Liveness property*: Accepted message will be eventually delivered, whose CTL formula is: ***G****(Accept(x)* ->***F*** *Deliver(x))*.

(P2) *Safety property*: Messages will be delivered in order, e.g. if message *x1* is sent by the sender before message *x2*, *x1* will be delivered by the receiver before *x2*, whose CTL formula is: ***G***((*Accept(x1)* **U** (*Accept(x2)* $\wedge$ $\neg$*Accept(x1)*)$\rightarrow$ ($\neg$*Deliver(x2*) **U** *Deliver(x1*))).

(P3) *Safety property*: Each message is delivered only once (i.e., no duplication of messages) with the assumption that all accepted messages are distinct, whose CTL formula is: ***G***(*Deliver(x)* $\wedge$ *Deliver(y)* $\rightarrow x \neq y$).

Following the same idea discussed in [5], a PrT net model for ABP can be easily transformed into a NuSMV model. The first NuSMV model translated from the PrT net model defined in Figure 2 is similar to the one discussed in [5], and the second NuSMV model translated from the PrT net model defined in Figure 3 was rewritten based on an ABP example included in the NuSMV distribution [11]. To make the underlying transition system finite and to simplify the analysis, we assume that there are eight distinct messages accepted initially. Namely, there are eight distinct tokens in place *Accept*, and no any other message is accepted. We verify that all eight messages and only eight messages are delivered, and eight messages are delivered in the order they are sent by the Sender. The property part in NuSMV is defined as follows according to the properties we intend to verify.

(1) All messages are eventually delivered. This formula ensures property (P1): ***AF***(*Deliver* = 8). 8 means the place *Deliver* has received 8 messages.

(2) Messages are delivered in order and all delivered message are distinct. This formula ensures properties (P2) and (P3): ***AG***(*deliverData* & ***!next***(*deliverData)* $\rightarrow$ *DataIn/2* = *Deliver* +*1*). For modeling convenience, messages are encoded as: its sending order (from 1 to 8) times 2 then plus bit number 1 or 0.

In addition, fairness constraints have to be added to the NuSMV model to ensure the fairness. Otherwise, above properties do not hold because a message could always be lost or corrupted so that it will never be delivered, which is inconsistent to the protocol specifications.

When we verified the three properties in the NuSMV model for the PrT net defined in Figure 2, we found several interesting problems. Although the three properties were evaluated as true in NuSMV, we do find that one message could be duplicated and delivered to the receiver, which has been confirmed in the simulation. We change the property (1) as ***AF***(*Deliver* = 9), and then the formula was still evaluated as true. Since only 8 messages total have been accepted, 9 messages have been delivered means some message has been duplicate. We checked the PrT net model defined in Figure 2 and found that: when a message was successfully delivered, the message in the *DataBuf* was not removed. If the *ack* message is lost or corrupted, the sender (but not the receiver!) in the model has to tell the receiver to resend the *ack*. When the sender asks the receiver to resend the *ack*, the message stored in place *DataBuf* is also sent out, since the sequence numbers in both receiver and sender were flipped when the original message was sent and delivered (*i.e*., they are still consistent), so that the message was delivered to place *Deliver* again.

What has happened on property (2), which should verify the duplication of messages and the ordering of the delivery? We checked the original NuSMV program, and found the message sent from the sender was encoded based on the formula defined in property (2). Therefore, if a message is delivered, but its *ack* is lost, then the message is sent out with the command for resending *ack* from the sender is different to the original message (suppose it should be the same one, but it is the next one) because the sequence number has been flipped as soon as the message is delivered. In other words, duplication of messages in the model did not really duplicate a message but it causes additional messages were sent (*i.e*, 9 messages but not 8 messages had been delivered, and all of them were different). Therefore, property (2) is still hold, but the model has bugs.

Although model checking is an excellent technique for rigorously analyzing software specifications, it is fairly

challenge to ensure the model checking quality when the model become complex. The example we just investigated tells that combing a lightweight analysis tool with model checking technique is great help to understand a formal model and ensure the analysis quality.

IV. Related Work

Model checking has been widely used for analyzing software specifications [2]. In [5], He and *et. al*. reported a method for formally analyzing Petri nets using model checking and formal proof techniques. Ding and He discussed an approach for modeling checking a type of high level Petri nets in [3]. Several other researchers have defined an executable semantics for a software specification and supported some analysis capabilities through simulation and or formal verification [9]. For example, Rapide [7] supports simulation, the Chemical Abstract Machine [6] and Wright [1] support limited formal verification. Several researchers also explored testing of software specifications [12]. Test generation is the most important task for software testing. Software testing mainly has four types of test generation methods: program-based, specification-based, model-based and random test [13]. Model-based test generation generates model level tests from formal specifications like Z or Petri nets of the program under test, and then the model level tests are transformed into program level tests to test the program [14][18]. The approach discussed in this paper is different to existing work. Our approach focuses on the process of analyzing software specifications in particularly Petri nets via naturally combing informal and formal analysis techniques in two-phase analysis. The two-phase analysis approach bridges the gap between formally modeling software systems and formally analyzing software systems through injecting the model-based software testing technique between them. The model-based testing performed in the first phase of analysis uses the similar techniques used in model checking, which is performed in the second phase of analysis. The similarity of the techniques used in two phases is important for sharing analysis results in both phases so that to improve analysis effectiveness and efficiency.

V. Summary and Future Work

In this paper, we present an approach for analyzing software specifications in PrT nets. The approach is designed as a two-phase process to ensure it is not only practical but also rigorous for analyzing formal software specifications in PrT nets. In the first phase, a PrT net model is analyzed using model based testing techniques including simulation, verification and testing with tool MISTA. Then the important properties defined in temporal logic formulas for the PrT net model is further checked using model checking techniques with symbolic model checking tool NuSMV to ensure the correctness of the model. The errors found in the first phase and the counter examples generated in the second phase can help developers to find problems in the model. The analysis process is illustrated by analyzing an ABP model in PrT nets. We created two versions of PrT net model for ABP, but one of them has some issues. The issues had not been found during the model checking phase, but they had been easily detected during model-based testing phase. In addition, we also discussed how to use the model-based testing and model checking results to create a correct model. The analysis results have demonstrated the effectiveness of the approach. In the future, we are going to investigate rules on guiding model checking with model-based testing results and on directing model-based testing with model checking results. We also plan to develop a guideline for splitting analysis tasks in model based testing phase and model checking phase.

Acknowledgments

This work has been partially supported by NSF REU award No. 1262933, and 2012 Open Fund of State Key Laboratory for Novel Software Technology at Nanjing University.

References

[1] R. Allen, D. Garlan. “A formal basis for architectural connection.” ACM TOSEM 6 (3), 213–249, 1997.

[2] E. M. Clarke, O. Grumberg, D. Peleg. “Model Checking.” The MIT Press, 1999.

[3] J. Ding, X. He. “Formal Specification and Analysis of an Agent-Based Medical Image Processing System.” Intl. Journal of SEKE, Vol. 20, No. 3, pp. 1 – 35, 2010.

[4] J. W. Duran, S. C. Ntafos, "An Evaluation of Random Testing", IEEE TSE, Vol. SE-10, No. 4, pp438-443, July 1984.

[5] 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, 1994.

[6] P. Inverardi, A. Wolf. “Formal specification and analysis of software architectures using the chemical abstract machine model.” IEEE TSE, 21 (4), 373–386, 1995.

[7] D. C. Luckham, J. Kenney, et al. “Specification and analysis of system architecture using rapide.” IEEE TSE 21 (4), 336–355, 1995.

[8] L. Lamport, “The temporal logic of actions.” ACM Transactions on Programming Languages and Systems 16 (3), 872–923.

[9] N. Medvidovic, R. Taylor, 2000. “A classification and comparison framework for software architecture description languages”. IEEE TSE 26 (1), 70–93, 2000.

[10] T. Murata, “Petri nets: properties, analysis and applications.” Proceedings of the IEEE 77 (4), 541–580, 1989.

[11] NuSMV, http://nusmv.fbk.eu/, last accessed, March 2013.

[12] D. Richardson, A. Wolf. “Software testing at the architectural level.” In: Proc. of the 2nd Intl. Soft. Architecture Workshop. pp. 68–71, 1996.

[13] L. Shan, and H. Zhu, “Generating Structually Complex Test Cases By Data Mutation: A Case Study of Testing An Automated Modelling Tool”. The Computer Journal, Vol. 52, No. 5, 2009.

[14] 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.

[15] D. Xu, D., K. E. Nygard, “Threat-Driven Modeling and Verification of Secure Software Using Aspect-Oriented Petri Nets”. IEEE TSE. 32(4), 265–278, 2006.

[16] D. Xu, J. Ding, “Prioritizing State-Based Aspect Tests”. Proc. of ICST 2010, pp. 265-274, Paris, France, 2010.

[17] D. Xu, R. A. Volz, T. R. Ioerger, J. Yen, “Modeling and Analyzing Multi-Agent Behaviors Using Predicate/Transition Nets”. International Journal of Software Engineering and Knowledge Engineering 13(1), 103–124, 2003.

[18] H. Zhu, and X. He, “A methodology of testing high-level petri nets”. Journal of Information and Software Technology. v44, pp. 473-489, 2002.

A Best Method to Synthesize Very Large K-th Order Systems without Reachability Analysis

Daniel Yuh Chao & T. H. Yu
Department of Management and Information Science
National Cheng Chi University, Taipei, Taiwan, ROC
e-mail: yuhyaw@gmail.com; Phone: 886-2-29387694; Fax: 886-2-29393754

***Abstract*—Lately, a revolutionary method to synthesize maximally permissive with fewest monitors has emerged. It relies on reachability analysis to find minimal sets of legal and forbidden markings. A number of linear constraints are constructed to forbid all forbidden markings in the set, while guaranteeing all legal markings reachable by solving an integer linear programming problem (ILPP). Due to the state explosion problem, it cannot handle very large systems. We propose earlier a method without reachability analysis and minimal siphon extraction; hence it is scalable to large systems. This paper illustrates such by applying the method to very large k-th order systems.**

Index Terms: Petri net, flexible manufacturing system (FMS), deadlock prevention.

I. Introduction

PETRI nets have been popular [1, 4, 10] in modelling parallel programs, flexible manufacturing systems (FMS), etc. The fierce competition toward resources (such as data locks, machines, robots, machines, etc) leads to deadlocks, thwarting the usage of multiple processors and concurrent programming paradigm.

Traditional optimal design [5,10] of a liveness- enforcing supervisor strives for efficient computation, least cost, and maximally permissiveness, but not concerning with minimal structure complexity of employing fewest monitors.

Later on, Chen *et al*. [8] pioneer a linear programming method (ILP) to allow a plant to forbid as many FBM (first met bad markings) as possible to achieve the minimal configuration or least structure complexity among all former approaches. Cordone *et al*. [13] speeds up the method in [8] by the strategy of branch-and-bound. Nazeem *et al*. [12] revise some linear constraints in the iteration-based heuristic by replacing the notorious big *M* with *n* and adding a small positive parameter ε.

We [7] discover that a single called unmarked pattern (UP) of distribution of tokens in unmarked siphons unify all different types of critical siphons. We propose a method to merge several monitors into a single one while not losing states. It solves a linear set of equations, resulting in some formula of some parameters of the monitors of the controlled net. The total time complexity is linear to the size of the net compared with the exponential one of the current most advanced approaches. It achieves the same best results in the literature while avoiding the time-consuming reachability analysis and complete siphon computation which does not scale well with the size of the nets.

This paper illustrates the above claim by applying the proposed method to a k-th order system (a special case of Gadara net employed by Nazeem *et al*. [11] in their approach), where MIP (mixed integer programming) [3] is employed and hence cannot handle very large Gadara net.

Daniel Yuh Chao & T. H. Yu are with the Department of Management and Information Systems, National ChengChi University, Taipei 116, Taiwan, Republic of China. (e-mail: yuhyaw@ gmail.com.)

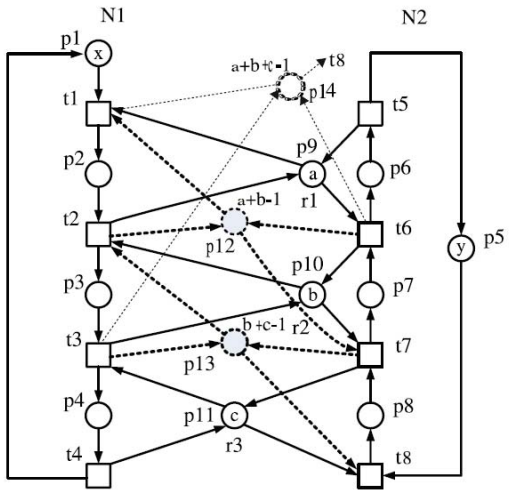


Figure 1: The controlled model of a 3-rd order system (when $a = b = c = 1$). $S_1 = \{p_3, p_6, p_9, p_{10}\}$, $S_2 = \{p_4, p_7, p_{10}, p_{11}\}$, $S_3 = \{p_4, p_6, p_9, p_{10}, p_{11}\}$, and $S_4 = \{p_{12}, p_{13}, p_3, p_7\}$.

Definition 1 *A k-th order system is a subclass of S^3PR* (systems of simple sequential process with resources) [9] *with k resource places $r_1, r_2, \ldots, r_k$ shared between two processes N_1 and N_2.*

$M_0(r_1) = M_0(r_2) = \ldots M_0(r_k) = 1$.

N_1 (resp. N_2) uses $r_1, r_2, \ldots, r_k$ (resp. $r_k, r_{k-1}, \ldots, r_1$) in that order.

Each of N_1 and N_2 is an elementary circuit.

The net in Fig. 1 is an example of controlled 3-rd order system. After removing all dashed objects (controlled arcs and places) from the net, it is a 3-rd order system. In the sequel, all the nets referred to are S^3PR. In [1], we show that an SMS (strict minimal siphons) in an S^3PR can be synthesized from a strongly connected circuit; such a circuit is called a *core* circuit.

Definition 2 *An elementary resource circuit is called a basic circuit, denoted by c_b. The siphon constructed from c_b is called a basic siphon. A n-compound circuit c is a circuit consisting of multiply interconnected elementary circuits c_{b_1}, c_{b_2} ,…, c_{b_n} extending between two processes and $c_{b_i} \cap c_{b_j} \neq \varnothing$ iff $j = i+1$, $i = 1, 2, \ldots, n-1$.* $c = c_{b_1} \circ c_{b_2} \circ \ldots \circ c_{b_{n-1}} \circ c_{b_n}$ *if $c_{b_i} \cap c_{b_j} = \{r_i\}$, $r_i \in P_R$; i.e., c_{b_i} and $c_{b_{i+1}}$ intersects at a resource place r_i. The SMS S synthesized from compound circuit c using the Handle-Construction Procedure in [1] is $S = S_1 \circ S_2 \circ \ldots \circ S_{n-1} \circ S_n$, where S_i is the basic siphon synthesized from c_{b_i}. Each siphon S^* such that $[S^*] \cap [B] \neq \varnothing$ is called an n-dependent one, where $B = S_1$ or S_n and $S^*_R = R(S_0)$ (the set of resource places in S_0). The set of n-dependent siphons is denoted as $\pounds(S_0)$. An SMS may contain both control and resource places; such an SMS is called a mixture siphon. If a n-dependent siphon S^* is a*

compound (resp. mixture) one, then S^* is called a n-compound (resp. n-mixture) siphon. The monitor for S^* (resp. S_i) is called an n-monitor (*resp. 1-monitor*).

II. Critical Siphons

We [14] propose to optimize the number of monitors (good states as well) if one adds monitors in the normal sequence of basic, compound, control, and other types of siphons. It is shown that among all 2-dependent siphons (depending on two component siphons), only one (called critical) siphon needs to be controlled by adding a monitor. This avoids redundant monitors and the unnecessary associated computational burden. Neither reachability graph nor minimal siphon needs to be computed achieving polynomial complexity-essential for large systems. As a result, there is no need to enumerate all siphons and the time complexity involved is polynomial.

We extend the result to n-dependent siphons with $n > 2$. It shows that in the set B of n-dependent siphons, there is only one emptiable (called critical) siphon (analyzed to be the one with the minimal number of tokens in the unmarked set of operations places) needs to be controlled while the rest of siphons are also controlled accordingly.

Definition 3 *Let $S_0 = S_1 \circ S_2 \circ \ldots \circ S_{n-1} \circ S_n$ be a compound siphon. The token distribution (called unmarked) pattern M is as follows: (1) For each singular place r_i, $M(r_i) = 1$, $M(H(r_i) \cap [S_0]) = 0$; and (2) For other r in S_0, $M(r) = 0$, $M(H(r) \cap [S_0] \cap {}^\bullet(V^\bullet)) = M_0(r)$,where $H(r)$ is the set of holder places of r (places that use r) and V is a monitor or control place. The unmarked n-dependent siphon S with the above unmarked pattern (UP) is called a critical siphon.*

Theorem 1 [7] *1) Once the critical siphon S for M in Def. 3 is controlled, so are the rest of siphons S' with $M([S']) \neq M([S])$ in $\pounds(S_0)$*, and

2) *when S is unmarked, $M([S]) = M_0(R(S_0)) - \theta$, where θ is the number of singular places and $R(S_0)$ the set of resource places in S_0.*

In Fig. 1, there are 3 SMS: $S_1 = \{p_9, p_{10}, p_3, p_6\}$, $S_2 = \{p_{10}, p_{11}, p_4, p_7\}$, $R(S_1) = \{p_9, p_{10}\}$, $R(S_2) = \{p_{10}, p_{11}\}$, $S_0 = S_3 = \{p_9, p_{10}, p_6, p_{11}, p_4\}$. $R(S_3) = \{p_9, p_{10}, p_{11}\} \Rightarrow R(S_3) = R(S_1) \cup R(S_2)$. S_1 and S_2 (resp. S_0) are basic (resp. compound) siphons; both are optimally controlled since $M(V_{S_1}) = M(p_{12}) = a + b - 1$ and $M(V_{S_2}) = M(p_{13}) = b + c - 1$. $M(S_0) = a + b + c$. S_3 is optimally-controlled (no need for control elements) *iff* $b = M(S_1 \cap S_2) = M(p_{10}) = 1$ explained as follows:

If $b > 1$, S_3 can become unmarked when the tokens at each resource place are trapped in $[S_3]$; i.e., $M(p_9) = M(p_{10}) = M(p_{11}) = 0$, $M(p_3) > 0$ (S_1 is controlled) and $M(p_7) > 0$ (S_2 is controlled), $M(p_2) = M_0(p_9)$, and $M(p_7) = M_0(p_{10})$. We need to add control elements for S_3 to be optimally-controlled. Note that in this case, all resource places are unmarked and $M_{max}([S]) = M_0(R(S)) - \theta$, with $\theta = 0$ consistent with Theorem 1.2. A monitor V is added so that $[V] = [S]$ and $M(V) = M(R(S)) - 1 = a + b + c - 1$. This token distribution pattern in $[S]$ is consistent with Theorem 1. The 2-control or 2-mixture siphon does not have such unmarked patterns and hence is not a critical one.

If $b = 1$, then $M(V_{S_1}) = M(p_{12}) = a$ and $M(V_{S_2}) = M(p_{13}) = c$ and the control siphon $S_4 = \{p_{12}, p_{13}, p_3, p_7\}$ ($[S_4] = \{p_2, p_8\}$) generated by circuit $[p_{12}t_7p_{13}t_2p_{12}]$ in Fig. 1 can be emptied when all tokens in $M(V_{S_1})$ and $M(V_{S_2})$ sink to p_2 and p_8, respectively. Thus, to avoid empty S_4, we need to add control elements for S_4. This precludes S_3 from becoming unmarked [i.e., $M(p_9) = M(p_{10}) = M(p_{11}) = 0$, $M(p_3) = 1$ or $M(p_7) = 1$], then S_1 or S_2 is unmarked against the fact that both are controlled.

Thus, S_3 can never become unmarked if $b = M(p_{10}) = 1$. Also, $M(S_1) = M(S_2) = 1$ and both S_1 and S_2 remain controlled since $M(p_{10}) = M_0(p_{10}) = 1$. All the rest of resource places are unmarked to have all their tokens trapped in $[S_3]$ and $M_{max}([S_3]) = M(R(S_3)) - 1 = M(R(S)) - \theta = a + c - 1$ with $\theta = 1$ consistent with Theorem 1. This token distribution pattern in $[S]$ is consistent with Theorem 1. and Def. 3. The 2-compound or 2-mixture siphon does not have such an unmarked pattern and hence is not a critical one.

On the other hand, if $b > 1$, one need not add control elements for S_4 to be controlled, since to empty S_4, a and $(b-1) > 0$ tokens of $M_0(V_{S_1}) = a + b - 1$ must go to p_2 and p_7 respectively. Hence, $M(p_7) = b - 1 > 0$ and $M(S_4) \geq M(p_7) > 0$ and S_4 can never be emptied.

Please refer to [14] for critical siphons that are neither control nor compound siphons.

III. Control Policy

This section reviews the control policy in [8]. Similar to [8], only the tokens in operation places are considered to obtain a PI to prevent an FBM from being reached since all places in $[S]$ are operation ones. By controlling the maximal amount of tokens in these operation places, S can never become empty of tokens. Following that in [8], N_A is defined as $N_A = \{i | p_i \in P_A\}$, where P_A is the set of operation places. Thus, the relations between different markings are simplified to study the number of tokens in these operation places.

To forbid an FBM M', the plant is enforced to satisfy the following P-invriant con straint:

$$(G(M) = \sum_{i \in N_A} l_i \cdot \mu_i) \leq \beta, \ \mu_i = M(p_i) \tag{1}$$

Where

$$\beta = \sum_{i \in N_A} l_i \cdot M'(p_i)) - 1 \tag{2}$$

Constraint (1) is called the forbidding condition.

For maximally permissive control purpose, all legal markings should be kept after a control place is added. To ensure that every live marking M' cannot be prevented from being reached, coefficients l_i ($i \in N_A$) should satisfy

$$\sum_{i \in N_A} l_i \cdot M'(p_i)) \leq \beta, \ \forall M' \in M_L \tag{3}$$

where M_L is the set of all legal markings. Constraint (3) determines the feasible values for coefficients l_i ($i \in N_A$). Thus, for an FBM M, if coeffcients l_i ($i \in N_A$) satisfy Constraint (3), a PI designed for Constraint (1) can guarantee the reachability of all the legal markings and forbid M. Therefore, a control place computed for the PI can ensure all the legal markings be reached. In this case, the control place is said to be optimal. By setting $\sum_{i \in N_A} l_i \cdot \mu_i) = \beta$, all M' such that $\sum_{i \in N_A} l_i \cdot \mu_i) > \beta$ are forbidden too. This allows us to solve equations rather than inequalities. Alternatively, one can assign a monitor to each critical siphon and merge monitors as many as possible.

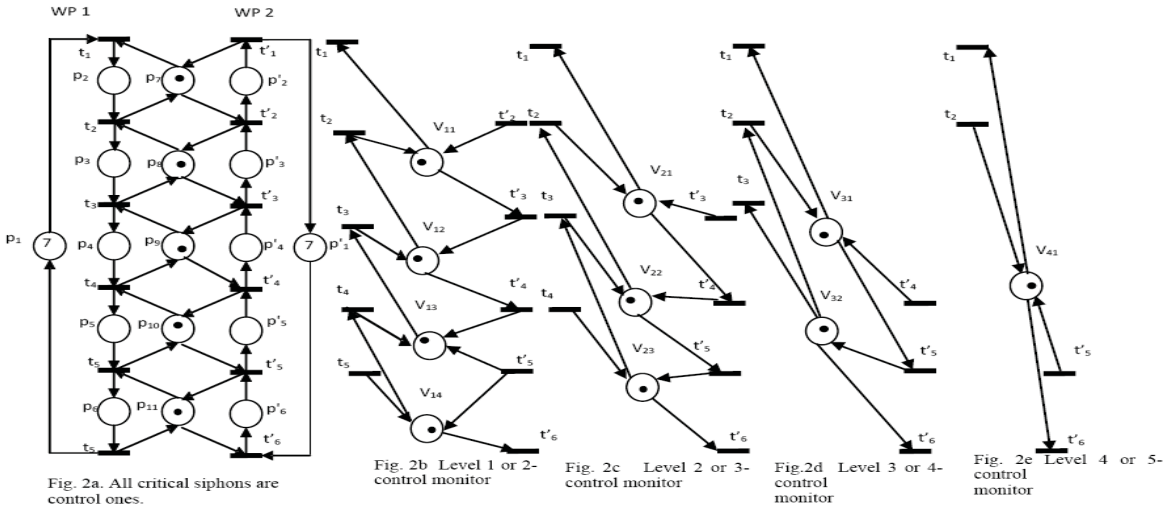


Figure 2: Petri net model of an FMS.

Chen *et al*. [8] pioneer a technique to reduce the computation burden by considering only a minimal covering set of legal markings and a minimal covered set of FBM via a vector covering approach. Similarly, one can select only one critical FBM $M = M^*_F(S)$ among those associated with a critical siphon S such that once M is forbidden, so are all the FBM related to S (denoted by $M_F(S_i)$). One can also choose one critical live marking $M^*_L(S_a, S_b)$ such that once $M^*_L(S_a, S_b)$ is not forbidden, so are all the legal markings related to S_a and S_b (denoted by $\pounds(S_a; S_b)$, which carries a different meaning than $\pounds(S_0)$, the set of set of n-dependent siphons derived from S_0 defined in Def. 2).

Theorem 2 *Let V_a and V_b be two monitors added to control S_a and S_b, respectively and $M^*_F(S_i)$* (resp. $M^*_L(S_a, S_b)$), *the critical FBM* (*resp. legal*) *defined above*. Let L ($l_i = 0$, $\forall p_i \notin [S_a] \cup [S_b]$) *be the solution to the following equations.*

$$L \cdot M^*_F(S_i) = \beta + 1 = k,\ i = a, b, \text{ and} \quad (4)$$

$$L \cdot M^*_L(S_a, S_b) = \beta = k - 1. \quad (5)$$

Then the PI

$$L \cdot M \le \beta,\ \forall M \in R(N, M_0), \quad (6)$$

does not forbid any live state and controls both S_a and S_b; that is, V_a and V_b can be merged into a single monitor.

This theorem allows to find the PI constraint by solving a set of linear first order equations.

Example 1: In Fig. 1, $S_1 = \{p_3, p_7, p_9, p_{10}\}$ ($[S_1] = \{p_2, p_6\}$), $S_2 = \{p_4, p_6, p_{10}, p_{11}\}$ ($[S_2] = \{p_3, p_5\}$), and Control siphon $S_3 = \{V_1(p_{12}), p_4, V_2(p_{13}), p_7\}$ ($[S_3] = \{p_2, p_5\}$) is synthesized from Control circuit $[t_2 p_{12} t_7 p_{13}]$ where all places are monitor ones. Control siphon S_3 is a critical one (since $S_1 \cap S_2 = \{p_{10}\}$ and $M_0(p_{10}) = 1$ based on the theory in [18]) in the sense that once S_3 is controlled, S_4 and the rest of siphons containing at least one of V_1 and V_2 is also controlled.

Considering merging V_1 and V_3 (monitors for S_1 and S_3, respectively) where $[S_1] \cap [S_3] = \{p_2\} \neq \varnothing$. $[S] = [S_1] \cup [S_3] = \{p_2, p_5, p_6\}$. Hence, the constraint to control both S_1 and S_3 is: $l_2 \cdot \mu_2 + l_5 \cdot \mu_5 + l_6 \cdot \mu_6 < \beta + 1 = k$, $l_i = 0$, $\forall p_i \notin [S]$. S_1 (resp. S_3) is unmarked at $M_1 = p_2 + p_6$ (resp. $M_3 = p_2 + p_5$), and $M^* = p_5 + p_6 = M_d - p_2$ is a live marking since $M^*(S_1) = M^*(S_3) = M^*(S) = 1$ where $M_d = p_5 + p_6 + p_2$ ($= M'$ in Eq. (2)) is an FBM to be forbidden, $M_d(S) = 0$. We have

$$l_2 + l_6 = k; \quad \text{(to forbid } M_3\text{), and } l_5 + l_6 = k - 1 \text{ (to not forbid } M^*\text{)}$$

Solving the above two equations, we have $l_2 = 2$; $l_5 = l_6 = 1$; $k = 3$, and the constraint that controls the two siphons is

$$2\mu_2 + \mu_5 + \mu_6 \le 2 \quad (7)$$

By Theorem 2 in [6], V_{S_1} and V_{S_2} cannot be merged (since $[S_1] \cap [S_2] = \varnothing$). Thus, a separate monitor must be added for S_2.

The resulting controlled model is shown in Fig. 1 and maximally permissive.

IV. Supervisor Control

First, it is easy to see that a k-th order system is a special case of α-S^3PR (defined below) and the initial markings of resource places is one (Fig. 2a).

Definition 3 [6] A*n α-S^3PR is an S^3PR where if any two basic circuits c_{b_1} and c_{b_2} intersect, they must intersect at a single resource place r*.

The net in Fig. 1 is α-S^3PR (also a k-th order system, k=3) defined above.

Theorem 3 [6] *The number of monitors for an α-S^3PR obtained using the methods in [10] is lower bounded by the number of basic siphons* .

Hence, a minimal configuration of a k-th order system has k monitors, where k is the total number of basic siphons.

All control siphons are emptiable. There are 4 ($k = 4$) basic siphons S_1, S_2, S_3, S_4 (Fig. 2a) with monitors V_{S_1} V_{S_2}, V_{S_3}, and V_{S_4}. There is a control circuit (Fig. 2b) containing every two adjacent monitor places; e.g., V_{S_2} and V_{S_3}. The corresponding 2-control siphons are critical ones. Monitors for these 2-control siphons are shown in Fig. 2c

Again, there is a control circuit (Fig. 2c) containing every two adjacent monitor places; e.g., $V_{22}[M_0(V_{22})=1]$ and $V_{23}[M_0(V_{23})=2]$. The resulting 3-control siphon is emptiable and critical since the unmarked pattern is $M(p_8) = M(p_{13}) = 0$ and $M(p_9) = M(p_{10}) = 1$.Monitor V_{32} is added with $M_0(V_{32})=2$ and $[V_{32}]=\{p_2, p'_6\}$ as shown in Fig. 2d. Similar conclusion applies to other level control or monitor places shown in Figs. 2d. and 2e, respectively.

In general, there are n basic siphons $S_1, S_2, \ldots, S_n$ (Fig. 2a), n-1 2-control siphons $V_{11} \circ V_{12}, V_{12} \circ V_{13}, \ldots, V_{1(n-1)} \circ V_{1(n)}$ (Fig. 2b), n-2 3-control siphons $V_{21} \circ V_{22}, V_{22} \circ V_{23}, \ldots, V_{2(n-2)} \circ V_{2(n-1)}$

(Fig. 2c), … 1 n-compound siphon $V_{(n-1)1} \circ V_{(n-1)2}$ (Fig. 2d), where V_{ij} is the jth monitor in Level i. The total number of emptiable siphons or monitors is also $n + (n-1) + (n-2) + \ldots + 1 = n(n+1)/2$. Based on Theorem 3, the minimal configuration employs n monitors.

These monitors can be grouped into the following *n sets* and each set can be merged into one monitor.

1. V_1, V_{11}, V_{12}, V_{13}, …, $V_{1(n-1)}$, $V_{1(n)}$ (n monitors),
2. V_2, V_{21}, V_{22}, V_{23}, …, $V_{2(n-1)}$, $V_{2(n)}$ (n-1 monitors),
3. V_3, V_{31}, V_{32}, V_{33}, …, $V_{3(n-1)}$, $V_{3(n)}$ (n-2 monitors),
4. …,
5. $V_{(n-1)(n-1)}$, $V_{(n-1)(n)}$ (2 monitors)
6. $V_{n,}$ (1 monitor)

We now consider merging the first set based on Theorem 2. The rest can be done similarly. First, the plant is enforced to satisfy the following P-invariant constraint:

$$(G(M)=\sum_{i\in NA_1} l_i \cdot \mu_i + \sum_{i\in NA_2} l'_i \cdot \mu'_i) \le \beta,\ \mu_i = M(p_i),\ \mu'_i = M(p'_i) \qquad (8)$$

where $N_{A_j} = \{i | p_i \in P_{A_j}\}$, $j = 1,2$. We are to solve l_i, l'_i , and β.

$$L \cdot FBM_1 = l_1 M_F(p_1) + l'_2 M_F(p'_2) = \beta,$$
……
$$L \cdot FBM_2 = l_1 M_F(p_1) + l'_3 M_F(p'_3) = \beta,$$
$$L \cdot FBM_n = l_1 M_F(p_1) + l'_{n+1} M_F(p'_{n+1}) = \beta, \text{ and}$$
$$L \cdot M_L = l'_2 M_L(p'_2) + l'_3 M_L(p'_3) + \ldots + l'_{n+1} M_L(p'_{n+1}) = \beta\text{-}1,$$

where $p_i \in N_1$ and $p'_i \in N_2$ belong to the holder set of a resource place $r \in P_R$. Setting $M_F(p_1) = M_F(p'_2) = M_F(p'_3) = \ldots = M_F(p'_{n+1}) = 1$ and $l_1 = a$, $l'_2 = b_2$, $l'_3 = b_3$, …, $l'_{n+1} = b_{n+1}$, we have

$$a + b_2 = \beta,$$
$$a + b_3 = \beta,$$
……
$$a + b_{n+1} = \beta, \text{ and}$$
$$b_2 + b_3 + \ldots + b_{n+1} = \beta - 1.$$

Solving the above equations, we have

$$b_2 = b_3 = \ldots = b_{n+1} = 1, \text{ and } a = \beta - 1 = n$$

The resulting constraint is

$$n\mu_1 + \mu'_2 + \mu'_3 + \ldots + \mu'_n + \mu'_{n+1} \le n + 1. \qquad (9)$$

When n = 2, we have

$$a + b_2 = \beta,$$
$$a + b_3 = \beta, \text{ and}$$
$$b_2 + b_3 = \beta - 1.$$

Solving the above equations, we have

$a = 2$ and $\beta = 3$ and the linear constraint is

$$2\mu_1 + \mu'_2 + \mu'_3 \le 3.$$

When n = 3, we have

$b_2 + b_3 + b_4 = \beta - 1$; b_4 is added to the left side compared with the case of $n = 3$. Hence, β is increased by one (to 4 from 3), so is a based on the equation $a + b_2 = \beta$.

Similarly, for the second set, we have

$$(n-1)\mu_2 + \mu'_3 + \mu'_4 + \ldots + \mu'_{n-1} + \mu'_n \le n.$$

Clearly, the time complexity involved is $O(n)$, where n is the total number of basic siphons. This is much faster than the exponential amount of time to solve the ILP problem for the current most advanced approaches.

V. CONCLUSIONS

We have illustrated a simple method to synthesize supervisors for very large k-th order system which overcome the state explosion problem involved in the reachability analysis. Our method first identifies all critical siphons (inferring from patterns M of unmarked siphons and the derived markings necessarily evolving into M) and the associated monitors. Second, we merge as many monitors as possible without losing any legal states. For k-th order systems, this amounts to solving a set of linear equation. The time complexity is linear to the basic circuits (or elementary circuits containing only resource places), much faster than the exponential time for solving integer linear programming problems.

REFERENCES

[1] D. Y. Chao, "Computation of elementary siphons in Petri nets for deadlock control," Computer Journal, (British Computer Society), vol. 49, no. 4, pp. 470-479, 2006.

[2] D. Y. Chao, "An incremental approach to extract minimal bad siphons," Journal of Information Science and Engineering, vol. 23, no. 1, pp. 203-214, Jan. 2007.

[3] D. Y. Chao, "Technical Note - MIP iteration -reductions for deadlock prevention of flexible manufacturing systems," International Journal of Advanced Manufacturing Technology, vol. 41, no. 3, pp. 343-346. 2009, doi:10.1007/ s00170-008-1473-x.

[4] Chao, D. Y., "Improved controllability test for dependent siphons in S3PR based on elementary siphons," Asian Journal of Control, vol. 12, no. 3, pp. 377-391, doi:10.1002/asjc.217, 2010.

[5] D. Y. Chao, "Improvement of suboptimal siphon- and FBM-based control model of a well-known S3PR," IEEE Transactions on Automation Science and Engineering, vol. 8, no. 2, pp. 404-411, 2011.

[6] Daniel Yuh Chao, "On the Lower Bounds of Monitor Solutions of Maximally Permissive Supervisors for A Subclass S3PR of Flexible Manufacturing Systems," Interna tional Journal of Systems Science, DOI:10.1080/00207721.2013.783946, 2013.

[7] Gaiyun Liu, Daniel Yuh Chao, and, Y. F, Fang, "A Control Policy for a Subclass of Petri Nets without Reachability Analysis," IET Control Theory & Applications, 2013, doi: 10.1049/iet-cta.2012.0426.

[8] Y. F. Chen and Z. W. Li, "Design of a maximally permissive liveness-enforcing supervisor with compressed supervisory structure for flexible manufacturing systems," Automatica, vol. 47, no. 5, pp. 1028-1034.

[9] J. Ezpeleta and L. Recalde, "A Deadlock Avoidance Approach for Non-sequential Resource Allocation Systems," IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans, vol. 11, pp. 173-184, 1995.

[10] H. Hu, M.C. Zhou, Z. W. Li, "Algebraic synthesis of timed supervisor for automated manufacturing systems using Petri nets," IEEE Transactions on Automation Science and Engineering, vol. 7, no. 3, pp. 549-557, July 2010.

[11] Yin Wang, Hongwei Liao, Ahmed Nazeem, Spyros A. Reveliotis, Terence Kelly, Scott A. Mahlke, and Stephane Lafortune, "Maximally permissive deadlock avoidance for multithreaded computer programs (Extended abstract)," IEEE Conference on Automation Science and Engineering, pp. 37-41 CASE 2009, Bangalore, India, 22-25 August, 2009.

[12] A. Nazeem, S. Reveliotis, "Designing maximally permissive deadlock avoidance policies for sequential resource allocation systems through classification theory," IEEE Conference on Automation Science and Engineering, pp. 405 - 412, CASE 2011, Tri este, Italy, Aug. 24-27, 2011.

[13] R. Cordone abd L. Piroddi, "Parsimonious monitor control of Petri net models of flexible manufacturing systems," IEEE Transactions on Systems, Man and Cybernetics: Systems, vol. 43, no. 1, pp. 215-221, 2013.

[14] Y. Y. Shih and D. Y. Chao, "Sequence of control in S3PMR," Computer Journal, vol. 53, no.10, pp. 1691-1703, 2010.

Combining multiple stress identification algorithms using combinatorial fusion

Yong Deng，Zhonghai Wu*
School of Electronic Engineering and Computer Science
Peking University, Beijing, China
dengyong@pku.edu.cn, wuzh@pku.edu.cn,
*corresponding author

D. Frank Hsu*
Department of Computer and Information Science
Fordham University, New York, NY 10023, USA
hsu@cis.fordham.edu

Abstract*—Sensor feature selection and combination algorithms are important in the identification of stress level for human activities and health. However, performance of each algorithm may depend on physiological sensors, sensing modalities and feature selection methods. Based on our previous work on feature selection and combination, we study combination of five sensor stress identification algorithms (SIA): C4.5, Naïve Bayes, Linear Discriminant Function, Support Vector Machine and K Nearest Neighbors across a variety of feature sets selected by C4.5, PCA, Correlation-based Feature Selection (CFS) and Diversity-based Feature Selection (DFS). Our experimental results demonstrate that combinatorial fusion is a viable method to improve identification of stress in human activities and health. Moreover, we observe that the improvement is stronger when the cognitive diversity between individual algorithms is bigger.(Abstract)*

Keywords- Physiological sensor; stress identification algorithm (SIA); combinatorial fusion, cognitive diversity; feature selection and combination; combining multiple SIAs formatting (key words)

I. INTRODUCTION

Stress has become more and more pervasive in people's daily life and it can have a negative impact on people's health. According to the latest stress survey by the American Psychological Association [2], more than half (56%) of the Americans reported that stress is a main source of their personal health problems. Also, more than 94% adults believe that stress can contribute to the development of major illnesses such as heart disease, depression and obesity, and that some types of stress can trigger heart attacks, arrhythmias and even sudden death, particularly in people who already have cardiovascular disease (92%).To effectively detecting the stress of human being in time provides a helpful way for people to better understand their stress patterns and provides physicians with more reliable data for intervention and stress management, which will be much useful for people's health.

In recent years, identifying the stress of human beings using multiple physiological sensors has been a hot research topic. Existing studies ([3, 4, 10, 11, 15, 22, 24]) have shown that psychosocial stress can be recognized by the physiological information of human being. The physiological information can be acquired by biological or physiological sensors, which usually include: ECG (electrocardiogram), GSR (galvanic skin response), EMG (electromyogram), and RESP (respiration).

Healey and Picard [11] have conducted studies in stress detection for real world driving tasks. In their experiment, the participants first wore 5 sensors: 1 ECG sensor, 1 EMG sensor, 2 GSR sensors (one for hand and the other for foot), and 1 RESP sensor. Then they will drive on a fixed route through downtown Boston covering three different driving conditions: rest, highway and city road. During their driving, the signals of the sensors were continually recorded. In the signal analysis period, a total of 22 features were extracted from the recorded signals, and Linear Discriminant Function (LDF) was used as the feature combination method to predict drivers' stress level during their driving [10]. Their work did not include the process of feature selection and combination.

The physiological signal data records used in Healey and Picard's experiment were partially published on the website PHYSIONET [19]. Although the driver dataset did not contain the complete data of all the drivers in Healey's experiment, the data allow others to further explore stress detection. Zhang, et al. [27] presented a systematic approach using a structurally learned Bayesian Network to fuse the sensor feature information and concluded that good correct rate can be acquired. Although a feature selection approach was mentioned, neither original data segments from which features were extracted are mentioned nor were the feature selection results shown in detail. In the work of [7], PCA method is applied to do feature selection. Five features are successfully selected from total 22 features and the sensor number is also reduced from 5 to 2.

Figure 1 shows a framework of multiple sensor fusion. First, features were extracted from the raw physiological sensor data using variant feature extraction techniques. Then most sensitive features are selected and then the selected features are combined to identify the stress level. Various feature selection and combination techniques may be used in this procedure and different decision results can be acquired accordingly. Then, various stress identification results are fused to improve the results of individual stress identification algorithm. In the paper, we focus on combining decision from various stress identification algorithms using a recently developed information fusion paradigm called combinatorial fusion analysis (CFA) ([12, 13]).

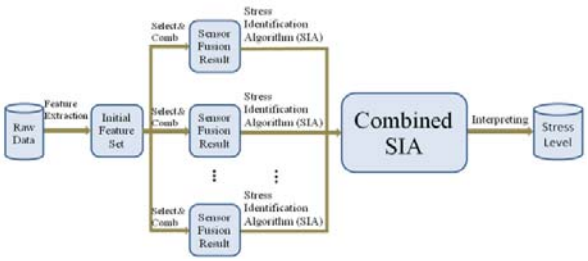


Figure 1. Procedure of multiple sensor combination

Combinatorial fusion analysis ([12, 13, 14]) treats each stress identification algorithm (SIA) as a scoring system A. Every scoring system A has a score function s_A, which assigns a score to each stress level, and a rank function r_A derived from s_A by sorting the score function values. The rank-score characteristic (RSC) function of A, f_A, is defined to be the composite function of s_A and the inverse function of r_A. By harnessing the asymmetry between score function and rank function, the diversity between two scoring systems A and B,$d(A,B)$, is defined to be the difference between the two RSC functions f_A and f_B.

The organization of this paper is as follows. In section II we briefly review preliminary work. These include combinatorial fusion, feature selection algorithms and results, and feature combination. Using combinatorial fusion to combine the decisions from various stress identification algorithms is presented in Section III. Our experiment results are described in Section IV. Finally, we provide conclusion and future work of the paper in Section V.

II. PRELIMINARY WORK

A. Combinatorial Fusion

Combinatorial Fusion Analysis (CFA) ([12, 13, 14]) provides a new paradigm of information fusion in analyzing the combination of multiple scoring systems (MSS). In this framework, each scoring system contains two functions: a score function and a rank function. Let D ($D = \{d_1, d_2, \dots, d_{n-1}, d_n\}$) be a set of candidates, such as sensor data, genes, documents, image, locations or classes. Assume A is a scoring system, which contains both a score function s_A and a rank function r_A on the set D. A Rank-Score Characteristic (RSC) function $f_A: N \to R$ can be computed as $f_A(i) = (S_A.r_A^{-1})(i) = S_A(r_A^{-1}(i))$, where N={1,2,…,n} and R is a set of real numbers ([13],[14]).

For a set of p scoring systems $A_1, A_2, \dots, A_p$ on the set D, two different approaches can be used to combine them: Score Combination (SC) and Rank Combination (RC) as the following:

$$s_{sc}(d) = (\sum_{i=1}^{p} s_{Ai}(d))/p, \quad (1)$$

$$s_{rc}(d) = (\sum_{i=1}^{p} r_{Ai}(d))/p, \quad (2)$$

where d is in D, and s_A and r_A are score function and rankfunction from D to R and N respectively.

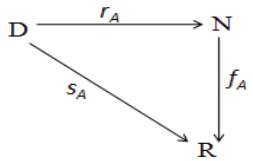


Figure 2. Rank-Score Characteristic (RSC) function[13,14]

For a pair of two scoring systems A and B, the diversity between A and B, $d(A,B)$, can be defined between their score functions, rank functions, or RSC functions as the following:
(a) $d(A,B) = d(s_A, s_B)$, s_A and s_B are score functions of A and B respectively;
(b)$d(A,B) = d(r_A, r_B)$, r_A and r_B are rank functions of A and B respectively;
(c)$d(A,B) = d(f_A, f_B)$, f_A and f_B are rank-score characteristic functions of A and B respectively.

Examples of the first two cases $d(s_A, s_B)$ and $d(r_A, r_B)$ are the Pearson correlation and rank correlation respectively. The third case $d(f_A, f_B)$ was defined as cognitive diversity and has been used in several application domains including information retrieval, virtual screening, target tracking, protein structure prediction, identification of degenerate motifs, and combining multiple ChIP-seq detection systems ([12, 13, 14, 16, 17, 18, 23, 26]).

B. Feature Extraction

Feature extraction has been performed by Healey et al ([10, 11]) and 22 features (See Table I) have been extracted from the driver stress dataset.

TABLE I. 22 features and their corresponding symbols [8]

Sensor	Feature Symbol (Feature Name)
EMG	A(EMG_mean)
Foot GSR	B(FGSR_mean); C(FGSR_std); (Fgsr_Freq); E(Fgsr_Mag); F(Fgsr_Dur); G(Fgsr_Area)
Hand GSR	H(HGSR_mean); I(HGSR_std); J(Hgsr_Freq); K(Hgsr_Mag); L(Hgsr_Dur); M(Hgsr_Area)
RESP	N(Resp_mean); O(Resp_std); P(Resp0~0.1); Q(Resp0.1~0.2); R(Resp0.2~0.3); S(Resp0.3~0.4)
ECG	T(HR_mean); U(HR_std); V(Ihr_LR)

C. Feature Selection algorithms and results

Feature selection aims to reduce the dimensionality of the input features and the number of sensors to wear and improve prediction correct rate. In this paper, we consider results from previous work on four feature selection algorithms: C4.5 decision tree, principal component analysis (PCA) [7], correlation-based feature selection (CFS) and diversity based feature selection (DFS) algorithms [8].

In reference [7], the decision tree is shown for driver stress condition generated by the C4.5 algorithm using the open source machine learning and data mining software WEKA. The five features remaining as the branch node of the tree are the feature selection result. The results of PCA feature selection method are also depicted. It can be seen that, the first component contributes about 80% of the original data set

Identify applicable sponsor/s here. *(sponsors)*

while the second and the third component contributes about 10% and 7% respectively. All together, the first five components can contribute to almost 100% of the whole original data set. Therefore, the strategy is to select the features which have the highest correlation with these five components.

Reference [8] shows the most significantly correlated features and the correlation coefficient between the features in our study. As shown, there are five groups of features (e.g., those extracted from RESP, FGSR, and HGSR) representing five sensing modalities, so that within each group, features are highly correlated with each other. In this case, maybe only one feature from each subclass should be considered in the evaluation. Reference [8] also presents the modalities as well as the intra&inter modality diversity for the features. The features are ranked in ascending order of the value of intra diversity within each modality. The number on the edge of two feature nodes represents the inter diversity of two features. For example, the inter diversity between feature A and feature E is 0.205.

Table II summarizes the initial feature selection results from this study. Both C4.5 and PCA approaches select five features while the features are different except FGSR_dur feature, which is labeled as "F".

TABLE II. Summary of Feature Selection Results [8]

Method	No. of Feature	Features Selected
C4.5	5	F,K,O,P,U
PCA	5	D,F,G,L,M
CFS1	5	A,E,J,P,T
CFS2	7	A,E,J,K,L,P,T
CFS3	9	A,E,J,K,L,M,P,R,T
CFS4	11	A,D,E,J,K,L,M,P,R,S,T
DFS1	5	A,E,M,P,T
DFS2	7	A,E,K,M,P,Q,T
DFS3	9	A,E,K,L,M,P,R,S,T
DFS4	11	A,D,E,K,L,M,P,Q,R,S,T

D. Feature Combination algorithms and results

Five feature combination algorithms are used: (a) C4.5, (b) Naïve Bayes, (c) Linear Discriminant Function, (d) SVM, and (e) k Nearest Neighbors.

Table III in Appendix presents feature combination results using leaving-one-out cross validation approach. It can be seen that Naïve Bayes performs best across most of the feature sets with only four exceptions that C4.5, SVM, k-NN and SVM perform best or equal to the best on the feature set (5 features) selected by C4.5 and by PCA, the feature set (7 features) selected by correlation, and the feature set (9 features) selected by diversity respectively. It is interesting to note that the feature combination algorithm C4.5 on the feature set (5 features) selected by C4.5 can achieve the highest result of 86.15%.

III. Combination of Stress Identification Algorithms

A. Multiple scoring systems

When a fusion method makes a decision for a group of feature data, it will calculate the probabilities of this data group belonging to different classes. The largest probability will be selected and its corresponding class will be regarded as the final combination result. We can view this combination probability distribution generation scheme as a scoring system. The probabilities that a fusion method calculate are the score values. Table III gives the corresponding formulas. For every testing case (a group of feature data), every scoring system will produce three probability scores, which are the probabilities the testing case belongs to three different stress level classes (low level, medium level and high level). The score formulas for C4.5, SVM, NB and k-NN are based on the probability distribution function Classifier::distributionForInstance(Instance inst) of Weka, a popular machine learning software.

TABLE III. Score formulas for 5 fusion methods

Fusion Method	Score Formula
C4.5	$p_{c_i} = n_i / n_{total}$ n_i is the number of cases belonging to class i on the resulting leaf node based on the trained tree model. n_{total} is the total number of cases belonging to the resulting leaf node based on the trained tree model.
SVM	$\forall i, j$, $\begin{cases} weight_i = weight_i + 1, if combination\ result\ \in C_i \\ weight_j = weight_j + 1, if\ combination\ result\ \in C_j \end{cases}$ $p_{c_i} = \frac{weight_i}{total\ class\ num}$
NB	$p(c_i \mid f_1, f_2, \dots, f_n) = \frac{p(c_i)p(f_1, f_2, \dots, f_n \mid c_i)}{p(f_1, f_2, \dots, f_n)}$
k-NN	$p_{c_i} =$ $\begin{cases} \frac{1}{num\ of\ training\ cases} + weight, if\ neighbo \in class\ i \\ \frac{1}{num\ of\ training\ cases} \quad , \quad otherwise \end{cases}$
LDF	$min\_g_c(\hat{y}) = min\{g_{c_1}(\hat{y}), g_{c_2}(\hat{y}), g_{c_3}(\hat{y})\};$ $p_i = (g_{c_i}(\hat{y}) - min\_g_c(\hat{y}))/(\sum_{i=1}^{3} g_{c_i}(\hat{y}))\ , i = 1,2,3;$

The diversity $d(A, B)$ between two algorithms A and B is defined by the cognitive diversity ([12, 13, 14]). $\mathrm{d}(\mathrm{A},\mathrm{B}) = \mathrm{d}(\mathrm{f_A}, \mathrm{f_B}) = \sqrt{\sum_{\mathrm{i}=1}^{\mathrm{n}} (\mathrm{f_A(i)} - \mathrm{f_B(i)})^2}$. n is the possible number of stress levels for the stress identification algorithm. i is the rank number. $f_A(i)$ and $f_B(i)$ are the score of rank number i for algorithm A and feature B respectively.

We can use formula (1) to do score combination directly. In order to do rank combination, firstly we will rank the three scores that an algorithm produces for a testing case. Our rank rules are as the following:

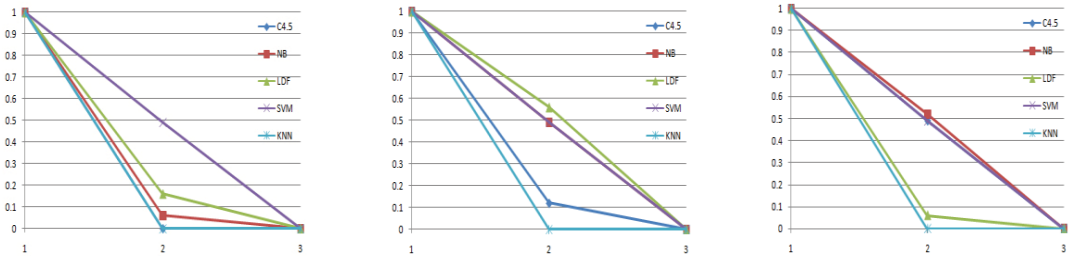
1) We sort the normalized n (in our test case, n=3 since there are totally 3 classes) scores in decreasing order. The rank value is assigned according to the decreasing value of the score.
2) For the equal scores, their corresponding rank values should be equal. We take the average of corresponding rank values.

For example, if the scores are (0.2, 0.1, 0.7), their corresponding rank values are (2, 3, 1). If the scores are (0.3, 0.3, 0.4) their corresponding rank values are (2.5, 2.5, 1). After the rank value assignment, we can then do rank combination

using formula (2). The score values for No. t (t=10, 20, 50) test case of feature set 5_C4.5 are shown in Table IV. The corresponding Rank Score Graph of Table IV is shown in Figure3.

TABLE IV. Score and Rank value for test case #t (t=10, 20, 50)

Fusion Method	Value Type	Test Case #10			Test Case #20			Test Case #50		
		Low Level	Middle Level	High Level	Low Level	Middle Level	High Level	Low Level	Middle Level	High Level
C4.5	Score	1.00	0.00	0.00	0	0.11	0.89	0.33	0.67	0.00
	Rank	1	2.5	2.5	3	2	1	2	1	3
NB	Score	0.14	0.10	0.76	0	0.33	0.67	0.08	0.58	0.34
	Rank	2	3	1	3	2	1	3	1	2
LDF	Score	0.00	0.88	0.12	0	0.36	0.64	0.05	0.00	0.95
	Rank	3	1	2	3	2	1	2	3	1
SVM	Score	0.00	0.67	0.33	0	0.67	0.33	0.00	0.33	0.67
	Rank	3	1	2	3	1	2	3	2	1
k-NN	Score	0.015	0.97	0.015	0.015	0.015	0.97	0.015	0.97	0.015
	Rank	2.5	1	2.5	2.5	2.5	1	2.5	1	2.5



Test Case 10 Test Case 20 Test Case 50

Figure 3. Rank Score Graph for test case #t (t=10, 20, 50)

B. *Combining stress identification algorithms using CFA*

We treat each of the five stress identification algorithms as a scoring system and perform 2-combination, 3-combination, 4-combination, and 5-combination using combinatorial fusion analysis. Score combination is to calculate the average score value of the corresponding class score values for the selected fusion approach. The class with the maximum combination score value is the final decision combination result. Rank combination is to calculate the average rank value of the corresponding class rank values for the selected fusion approach. The class with the minimum combination rank value is the final decision combination result. For example, if we select the results of C4.5 and NB in Table IV to do combination. The score combination result is: (Low, 0), (Middle, 0.22), (High, 0.78). The maximum score value is 0.78, which belongs to High level class. So score combination result is: High level. The rank combination result is: (Low, 3), (Middle, 2), (High, 1). The minimum rank value is 1, which belongs to High level class. So score combination result is: High level.

IV. EXPERIMENTAL RESULTS

A. *Combination results using CFA*

In Figure 4 (see Appendix), the details of combining stress identification algorithms using combinatorial fusion are presented. In these figures, the meaning of each of the five symbols is: A is C4.5, B is NB, C is LDF, D is SVM, and E is k-NN. For the 5-feature set selected by C4.5, the rank combination of A and B can result in the highest correct rate 92.31%. The score combination of A and E or A, B, C and D can also result in the highest correct rate 89.23%.

In Table VI (see Appendix), we can see that combing multiple stress identification results are better than results from individual algorithm as compared with Table V. The maximum decision result for all feature sets is larger than or equal to that of just feature combination. The best result in decision combination stage is 92.31%, which is obtained from the 5 features selected by C4.5 in the case of rank combination (Combining A and B).

B. *Comparison of Positive & Negative cases for 2-combinations*

Positive case is that the combination result is correct while at least one of the two individual prediction result is false. Negative case is that the combination result is false while at least one of the two individual prediction result is correct.

We analyze the positive cases as well as negative cases and their relation to the corresponding cognitive diversity in the combination of 2 sensor stress identification algorithms. In Figure 4 and Figure 5, we can see that overall the number of positive cases is larger than that of negative cases in each cognitive diversity band. The ratio of Positive#/Negative# increases according to the increasing of cognitive diversity.

C. *Discussion*

Overall, we see that combinatorial fusion of the five stress identification algorithms on the various feature sets can have best results equal to or higher than the best of each

individual algorithm except for two cases. Each of the two stress identification algorithms C4.5 and Naïve Bayes on the two feature sets (both 5 features) selected by C4.5 and by diversity performs better than any other combinatorial fusion by rank combination and score combination respectively. Figure 8 demonstrates that the best result may not be the combination of more or all individual systems.

Table IV exhibits that C4.5 and Naïve Bayes can acquire the best result of all the decision fusion when using rank combination. C4.5 and LDF can acquire the best result of all the decision results when using score combination. It is interesting to note that stress identification algorithm C4.5 plays an important role in combinatorial fusion on the feature sets of 22 features and those selected by C4.5 and PCA. On the other hand, algorithm NB plays similar vital role in combinatorial fusion on feature sets selected by correlation and by diversity. The ratio of Positive#/Negative# increases according to the increasing of cognitive diversity in case of 2-combination.

V. CONCLUSION AND REMARKS

In this paper, we showed how to use combinatorial fusion to fuse various stress identification algorithm results from physiological sensor information to detect people's stress levels. Our experimental results showed that combinatorial fusion provides a good method to combine sensor information at the decision level by combining multiple stress identification algorithms.

As discussed in section IV.C, combinatorial fusion on the five stress identification algorithms do achieve better results than each individual algorithm. As also discussed is that C4.5 does not play any role in the best results for combinatorial fusion on feature sets selected either by correlation or by diversity except in the case on the feature set (5 features) selected by correlation. On the other hand, algorithm NB is instrumental in the best results for 19 out of the total 22 feature sets (see last two column or Table VI). This is due to the fact that algorithm NB performs the best on 9 out of the 11 feature sets (see also last two column of Table III).

Comparing last two columns between Table III and Table VI, we see that combinatorial fusion produces better results than each individual algorithm in 8 out of 11 feature sets and has equal result as the best individual algorithm in the remaining 3 feature sets. These 3 feature sets are selected by PCA (5 features), diversity (DFS 1 with 5 features) and diversity (DFS 2 with 7 features) with individual best algorithms by SVM, NB and NB respectively. More significantly, the power of combinatorial fusion is reflected in the combination of C4.5 with each of k-NN and NB to achieve correct rate of 89.23% and 92.31% which is higher than the best correct rate of each individual algorithm 86.15% by algorithm C4.5. These improvements can be attributed to be the cognitive diversity as demonstrated in Figure 9 and Figure 10.

One of our long term goals is to estimate the performance of the combined algorithm P(** C) in terms of the average performance of individual stress algorithms P(to the BAR) and average diversity D(to the bar). In this regard, Chung et al [5,6] have obtained general results with respect to multiple classifier systems.

ACKNOWLEDGMENT

This research is supported by the State Key Program of National Natural Science of China (Grant No. 61232005) and National Science and Technology Major Project (No: 2012ZX03002022).

REFERENCES

[1] A. Akbas, Evaluation of the physiological data indicating the dynamic stress level of drivers. Scientific Research and Essays 6(2), (2011), pp. 430-439.

[2] APA (American Psychological Association), *Stress in America: Our Health at Risk,* **(2012)**.*URL:*

http://www.apa.org/news/press/releases/stress/index.aspx

[3] F. Angus, and J. Zhai, Front-end analog pre-processing for real time psychophysiological stress measurements, Proceedings of the 9th World Multi-Conference on Systematics, Cybernetics and Informatics (WMSCI), (2005), pp. 218-221.

[4] J. Bakker, M. Pechenizkiy and N. Sidorava, What's your current stress level? Detection of stress patterns from GSR sensor data, Proceedings of the11th IEEE International Conference on Data Mining Workshops.

[5] Y. S. Chung, D. F. Hsu, C. Y. Tang: On the Diversity-Performance Relationship for Majority Voting in Classifier Ensembles. MCS 2007: 407-420

[6] Y. S. Chung, D. F. Hsu, C.Y. Liu, C.Y. Tang: Performance evaluation of classifier ensembles in terms of diversity and performance of individual systems. Int. J. Pervasive Computing and Communications 6(4): 373-403 (2010)

[7] Y. Deng, Z. Wu, C. Chu, and T. Yang, Evaluating Feature Selection for Stress Identification. *International Conference of Information Reuse and Integration*, **(2012a)**, pp. 584-591.

[8] Y. Deng, D. F. Hsu, Z. Wu, C. Chu, Combining Multiple Sensor Features for Stress Detection using Combinatorial Fusion. *Journal of Interconnection Networks*, **(2012)**, Vol 13, Issue 3n04, DOI: 10.1142/S0219265912500089.

[9] R. Duda, P. Hart, and D. Stork, *Pattern Classification,* (2nd ed.), Wiley Inter-science **(2001)**.

[10] J. A. Healey, Wearable and automotive systems for affect recognition from physiology. Doctoral dissertation, Massachusetts Institute of Technology, MA **(2000)**.

[11] J. A. Healy, and R. W. Picard, Detecting stress during real-world driving tasks using physiological sensors. *IEEE Transaction on Intelligent Transportation System*, 6(2), **(2005)**, pp. 156-166.

[12] D. F. Hsu, and I. Taksa, Comparing rank and score combination methods for data fusion in information Retrieval, *Information Retrieval*, 8(3), **(2005)**, pp. 449-480.

[13] D. F. Hsu, Y. S. Chung and B. S. Kristal, Combinatorial fusion analysis: methods and practice of combining multiple scoring systems, in: H.H. Hsu (Ed.), *Advanced Data Mining Technologies in Bioinformatics*, Idea Group Inc. **(2006)**, pp. 32-62.

[14] D. F. Hsu, B. S. Kristal, and C. Schweikert, Rank-score characteristic (RSC) function and cognitive diversity, *Brain Informatics*, LNAI 6334, Springer, **(2010)**, pp. 42-54.

[15] E. Jovanov, A. O'Donnell Lords, D. Raskovic, P. G. Cox, R. Adhami, and F. Andrasik, Stress monitoring using a distributed wireless intelligent sensor system, *IEEE Engineering in Medicine and Biology Magazine*, **(2003)**, Vol. 22, pp. 49-55.

[16] K. L. Lin, C. Y. Lin, C. D. Huang, H. M. Chang, C. Y. Yang, C. T. Lin, C. Y. Tang, and D.F. Hsu, Feature combination criteria for improving accuracy in protein structure prediction, *IEEE Transaction on NanoBioscience*, 6, **(2007)**, pp. 186-196.

[17] D. M. Lyons, D. F. Hsu, Combining multiple scoring systems for target tracking using rank-score characteristics. *Information Fusion*, 10(2), **(2009)**, pp.124-136.

[18] C. H. Peng, J. T. Hsu, Y. S. Chung, Y. J. Lin, W. Y. Chow, D. F. Hsu, and C. Y. Tang, Identification of degenerate motifs using position restricted selection and hybrid ranking combination. *Nucleic acids research*, 34(22), (2006), pp. 6379-6391.

[19] PHYSIONET, Stress Recognition in Automobile Drivers *(drivedb)*, **(2010)**, URL: http://physionet.org/cgi-bin/atm/ATM/.

[20] I. Rish, An empirical study of the naive bayes classifier, *Proceedings of IJCAI-01 workshop on Empirical Methods in AI*, **(2001)**, pp. 41-46.

[21] S. Ruggier, Efficient C4.5.IEEE Transactions on Knowledge and Data Engineering, 14(2), **(2002)**, pp. 438-444.

[22] L. Salahuddin and D. Kim, Detection of acute stress by heart rate variability using a prototype mobile ECG Sensor, International Conference on Hybrid Information Technology, *Proceeding in IEEE CS*, **(2006)**,Vol. 2, pp. 453-459.

[23] C. Schweikert, S. Brown, Z. Tang, P. R. Smith and D. F. Hsu, Combining multiple ChIP-seq peak detection systems using combinatorial fusion, *BMC Genomics*, **(2012)**. (in press)

[24] F. T. Sun, C. Kuo, H. T. Cheng, S. Buthpitiya, P. Collins and M. Griss, Activity-aware Mental Stress Detection Using Physiological Sensors. *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, **(2012)**, Vol. 76, Part 5, pp. 211-230, DOI: 10.1007/978-3-642-29336-8_12.

[25] V. Vapnik, The Nature of Statistical Learning Theory, Springer Verlag, New York, **(1995)**.

[26] J. M. Yang, Y. F. Chen, T. W. Shen, B. S. Kristal and D. F. Hsu, Consensus scoring criteria for improving enrichment in virtual screen, *Journal of Chemical Information and Modeling*, **(2005)**,Vol. 45, no.4, pp. 1134-1146.

[27] L. Zhang, T. Tamminedi, A. Ganguli, G. Yosiphon, and J. Yadegar, Hierarchical multiple sensor fusion using structurally learned Bayesian network, *Proceedings of Wireless Health*, **(2010)**, pp. 174-183.

VI. APPENDIX

TABLE V. Feature combination results for C4.5, NB, LDF, SVM, and k-NN[8]

Feature Set	Feature Combination Results (%)						
	C4.5	NB	LDF	SVM	k-NN	Best result	Algorithm
22_Full	72.31	**75.38**	70.77	70.77	64.62	75.38	NB
5_C4.5	**86.15**	80	75.38	72.30	72.31	86.15	C4.5
5_PCA	58.46	67.69	64.62	**78.46**	72.31	78.46	SVM
CFS1 (5 features)	69.23	69.23	67.69	56.92	64.62	69.23	C4.5, NB
CFS2 (7 features)	63.08	**75.38**	66.15	67.69	75.38	75.38	NB, k-NN
CFS3 (9 features)	64.62	**78.46**	66.15	72.31	69.23	78.46	NB
CFS4 (11 features)	63.08	**75.38**	69.23	72.31	70.77	75.38	NB
DFS1 (5 features)	61.54	**73.85**	70.77	67.69	61.54	73.85	NB
DFS2 (7 features)	60	**72.31**	66.15	63.08	63.08	72.31	NB
DFS3 (9 features)	53.85	70.77	69.23	**73.85**	69.23	73.85	SVM
DFS4 (11 features)	66.15	**78.46**	73.85	70.77	70.77	78.46	NB
Average	64.616	**74.153**	68.922	69.538	68.924	74.153	NB

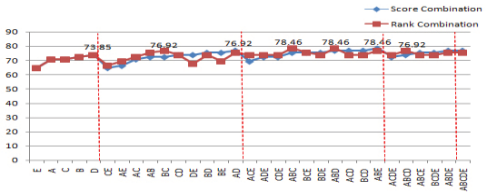


Figure 4a. CFA results on 22-feature set

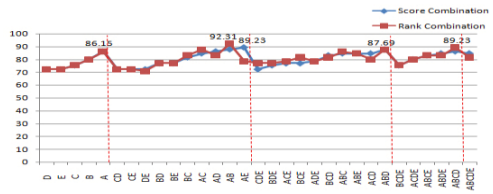


Figure 4b. CFA results on feature set selected by C4.5 with 5 features

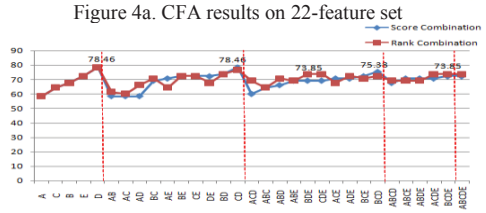
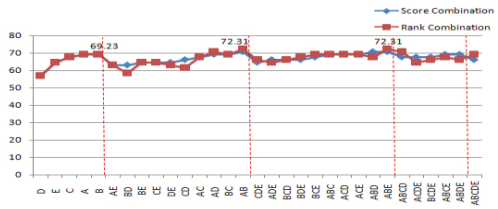
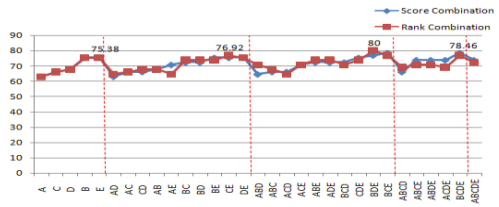


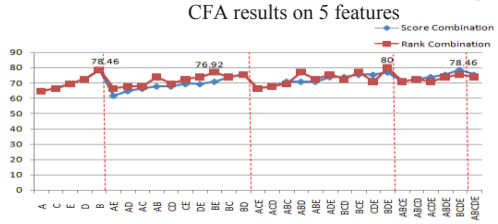
Figure 4c. CFA results on feature set selected by PCA with 5 features



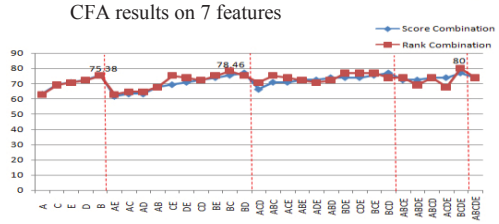
CFA results on 5 features



CFA results on 7 features

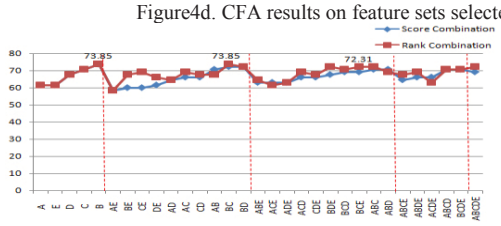


CFA results on 9 features

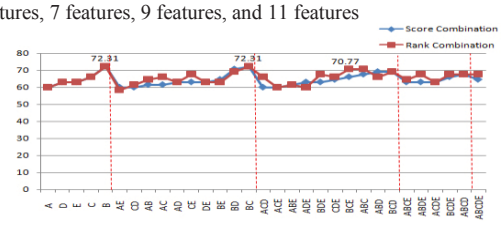


CFA results on 11 features

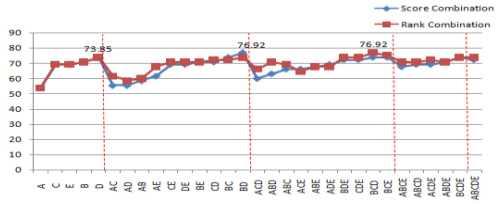
Figure4d. CFA results on feature sets selected by CFS with 5 features, 7 features, 9 features, and 11 features



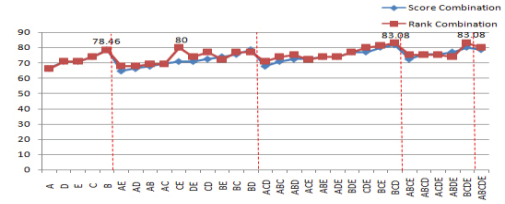
CFA results on 5 features



CFA results on 7 features



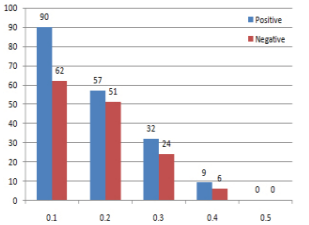
CFA results on 9 features



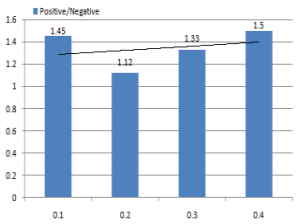
CFA results on 11 features

Figure 4e. CFA results on feature sets selected by DFS with 5 features, 7 features, 9 features, and 11 features

Figure 4. CFA results on the full 22-feature set (Figure 4a), 5-feature set selected by C4.5(Figure 4b) and by PCA(Figure 4c), and feature sets with 5,7,9 and 11 features respectively selected by CFS(Figure 4d) and by DFS(Figure 4e)

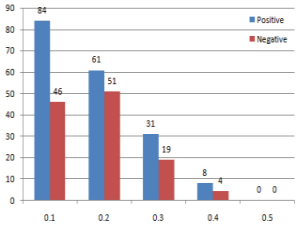


2-comb on full 22-feature set

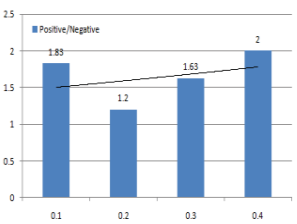


The Ratio of Positive#/Negative#

Figure 5a. 2-comb on full 22-feature set

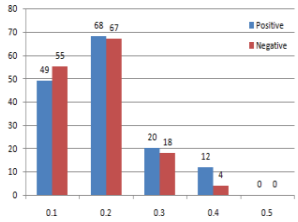


2-comb on C4.5 5-feature set

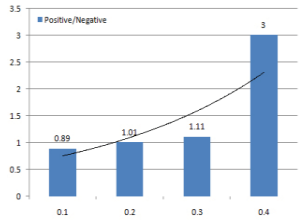


The Ratio of Positive#/Negative#

Figure 5b. 2-comb on C4.5 5-feature set

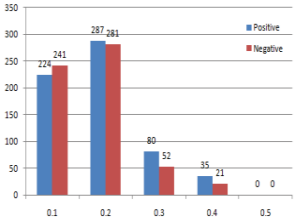


2-comb on PCA 5-feature set

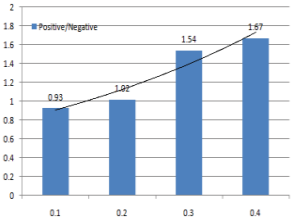


The Ratio of Positive#/Negative#

Figure 5c. 2-comb on PCA 5-feature set

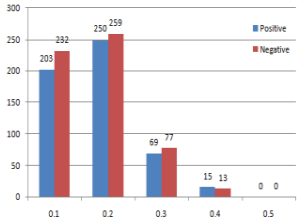


2-comb on CFS feature sets

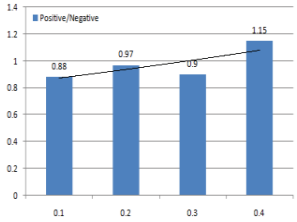


The Ratio of Positive#/Negative#

Figure 5d. 2-comb on CFS feature sets



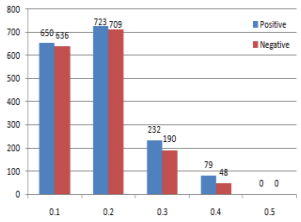
2-comb on DFS feature sets



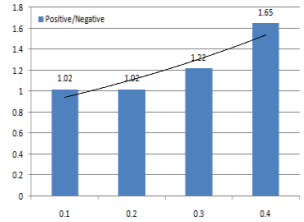
The Ratio of Positive#/Negative#

Figure 5e. 2-comb on DFS feature sets (*with 32 cases removed for the reason of 3 equal values)

Figure 5. Comparison of positive vs negative cases for 2-comb on full 22-feature set (Figure 5a), on C4.5 5-feature set (Figure 5b), on PCA 5-feature set (Figure 5c), on CFS feature sets (Figure 5d) and on DFS feature sets (Figure 5e). Cognitive diversity between tow systems is depicted as x-coordinate.



2-comb on all feature sets



The Ratio of Positive#/Negative#

Figure 6. Comparison of positive vs negative cases for 2-comb on all feature sets

TABLE VI. Combinatorial fusion results on t stress identification algorithms, t=2,3,4, and 5.

Feature Set	Comb Method	Ave. of 5 individual	Ave. of 2-comb	Ave. of 3-comb	Ave. of 4-comb	Ave. of 5-comb	Max Result	Comb Algorithms
Full (22 features)	Score	70.46	72.15	74.92	74.77	76.92	78.46	ABE
	Rank	70.46	72.00	75.23	74.77	75.38	78.46	ABC, ABD
C4.5 (5 features)	Score	77.23	80.00	80.46	81.85	84.62	89.23	AE
	Rank	77.23	79.38	81.23	82.15	81.54	92.31	AB
PCA (5 features)	Score	68.31	68.46	68.77	70.46	72.31	78.46	D, CD
	Rank	68.31	68.61	70.46	71.08	73.85	78.46	D
CFS1 (5 features)	Score	65.54	66.31	68.00	68.31	66.15	70.77	AB, ABD,ABE
	Rank	65.54	65.54	68.15	67.08	69.23	72.31	AB, ABE
CFS2 (7 features)	Score	69.54	70.46	71.54	73.23	73.85	78.46	BCDE
	Rank	69.54	70.46	72.31	71.38	72.31	80	BDE
CFS3 (9 features)	Score	70.15	68.62	72.15	74.15	75.38	78.46	B, BCDE
	Rank	70.15	71.69	72.77	72.62	73.85	80	BDE
CFS4 (11 features)	Score	70.15	69.38	72.62	73.85	73.85	76.92	BC,BCD,BCDE
	Rank	70.15	71.08	74	72.92	73.85	80	BCDE
DFS1 (5 features)	Score	67.08	65.23	66.92	67.69	69.23	73.85	B
	Rank	67.08	67.69	68.31	68.31	72.31	73.85	B,BC
DFS2 (7 features)	Score	64.92	64.00	64.46	64.62	64.62	72.31	B,BC
	Rank	64.92	64.92	65.85	66.15	67.69	72.31	B,BC
DFS3 (9 features)	Score	67.38	66.15	68.46	70.15	72.31	76.92	BD
	Rank	67.38	67.85	70.62	71.69	73.85	76.92	BCD
DFS4 (11 features)	Score	72	70.92	74.62	76	78.46	81.54	BCD
	Rank	72	73.08	76.15	76.62	80	83.08	BCD,BCDE

*A: C4.5, B: Naïve Bayes, C: Linear Discriminant Function, D: SVM, E: k Nearest Neighbors

eDOTS 2.0: A Pervasive Indoor Tracking System

Ryan Rybarczyk, Rajeev Raje, Mihran Tuceryan
Department of Computer and Information Science
Indiana University Purdue University Indianapolis
Indianapolis, Indiana USA
{rrybarcz, rraje, tuceryan}@cs.iupui.edu

***Abstract*— Designing tracking systems that cover large indoor areas and encompass different sensor modalities pose many significant challenges such as multi-sensor data fusion, coordinate system handoff and associated transformations. In this paper, we present the design and implementation of a prototypical system that effectively tackles these challenges. The proposed system is empirically validated in a laboratory setup and results of these experiments are also presented in this paper.**

Keywords-pervasive tracking, indoor tracking systems, multi-sensor data fusion, heterogenous tracking, sensor subset selection

I. INTRODUCTION

Indoor tracking is of critical importance in many futuristic application domains such as personalized health care setups in homes and assisted living centers. Typically, these application surroundings will contain deployments of multiple types of sensors (e.g., vision-based, Wi-Fi, etc.), each possessing different characteristics. Multiple types of sensors are necessary in these setups as each type may complement others, may provide a better coverage of a large indoor area, and overcome partial failure of sensors. Designing an effective pervasive indoor tracking system in such heterogeneous sensor-based setups requires tackling of following scientific challenges: a) the fusion of different types of location data obtained from various sensor modalities, b) a seamless handoff between different coordinate systems and associated transformations, and c) the dynamic nature of the indoor setup. This paper tackles the first two challenges, while the third challenge is considered as future work. Hence, specific contributions of this paper are: a) a development of effective techniques to tackle the first two challenges, and b) an empirical validation of these solutions by creating a prototypical system. The prototypical system is called the eDOTS (Enhanced Distributed Object System) 2.0 and it is an improved version of a preliminary prototype called eDOTS 1.0.

II. ENHANCING AN INDOOR TRACKING SYSTEM

The eDOTS 1.0 [9,17,18] was developed in an effort to provide an indoor tracking framework made up of one type of sensors, specifically, vision-based sensors. This framework utilized web cameras and the ARToolkit API [7,12,14] for the ability to track marker patterns. The eDOTS 1.0 made use of various data fusion techniques (e.g., averaging, Kalman filtering, etc.) to provide an estimate of an object's location from different location readings. Due to the stringent time sensitive requirements of data fusion, the underlying issue of clock synchronization was also addressed in the prototype. More information regarding the eDOTS 1.0 can be found in [9,17,18], however, its architecture is briefly described below to provide a basis for the discussion of the eDOTS 2.0.

The design of the eDOTS 1.0 is made up of three separate layers: Sensor, Middleware, and User Interface. The sensor layer is responsible for providing a software abstraction of the underlying physical sensor in the form of a sensor service. The Middleware layer contains discovery and the filter services that are responsible for locating available sensor services and communication between the sensor services and the user interface. The user interface layer is responsible for providing a graphical interface to the observer (i.e., entity interested in tracking an object of interest) as well as communicating with the middleware layer in order to retrieve data related to the tracking of an object. The design of the eDOTS 2.0 reused some components of the eDOTS 1.0 and enhanced the remaining components. Figure 1 indicates the components that were modified between version 1.0 and 2.0 of the eDOTS and Figure 2 presents the sequence of activities occurring in the eDOTS 2.0. A discussion of various components of eDOTS 2.0 is presented in next few subsections.

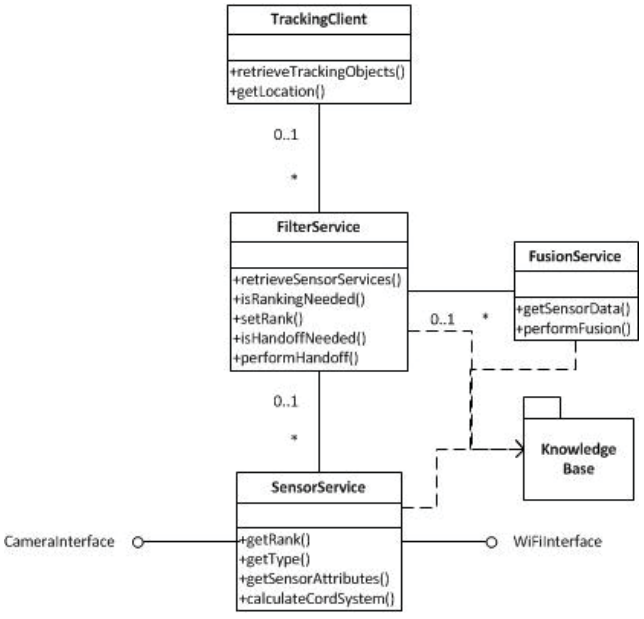


Figure 1. eDOTS 2.0 Enhancements

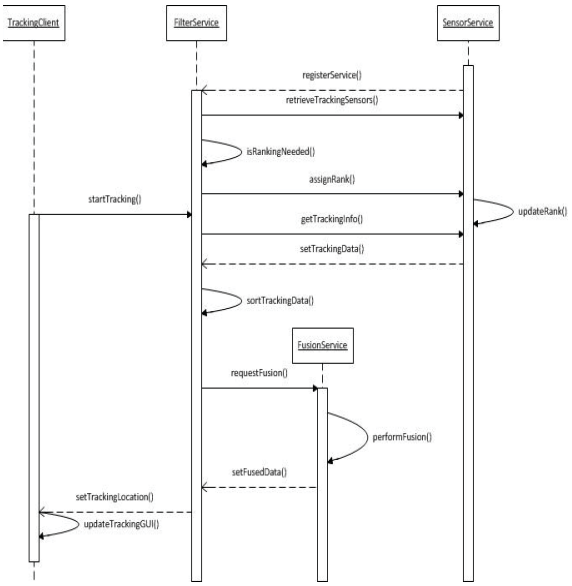


Figure 2. eDOTS 2.0 Tracking Overview

A. Multi-Sensor Data Fusion

Data fusion is defined as the process of combining a collection of data results in an effort to present a single representation of the data [21]. This process requires identifying and merging data readings from a set of sensors, within a given time frame, that are currently tracking an object. Unlike the homogeneous sensor situation, the data fusion in a multi-modal scenario has to tackle several formats (e.g., varying degrees of freedom associated with different classes of sensors). The data fusion process in the eDOTS 2.0 begins when the Tracking Client (see Figure 2) issues a request, on behalf of a user, for an object to be tracked. This request is handled by the Filter Service. The Filter Service is responsible for collecting data from various sensors that are able to locate the object, pass this data set to the Fusion Service, and then relay the fused result obtained from the Fusion Service back to the Tracking Client. This combining of data, carried out by the Fusion Service, can either be through simple averaging of the data estimations provided by different sensors or the use of more complex methods such as a Kalman-based technique [5,23]. This result provided to the Tracking Client by the Filter Service is in the form of a tuple representing the X, Y, and Z coordinates of the object and an associated time-stamp. This result then can be graphically displayed to the user.

One implicit challenge in the fusion process is the problem of subset selection – i.e., identifying a subset of sensors out of the ones that are able to track the object at a time instant. Once this subset is selected, the next task is to tackle the heterogeneity (e.g., THE NUMBER OF degrees of freedom) associated with the data readings of the selected sensors. There are many possible approaches to the subset selection problem – three alternatives are described below.

The first alternative is to take all of the data, regardless of the sensor modality, and send it to the Fusion Service. In this approach, the Fusion Service is responsible for deciding an appropriate technique to combine heterogeneous data. The second alternative is to separate the data by the sensor modality type prior to sending it to the Fusion Service. In this method, the Fusion Service can merge the data belonging to each modality separately. This technique allows a comparison between the readings provided by various classes of sensors. This pre-filtering activity is done by the Filter Service. The third alternative is to allow the Filter Service to only send those results that meet a specific Quality of Service (QoS) criterion as set by the Tracking Client. In this situation, the type of sensor modality is not directly examined but rather the selection of the data is based upon the QoS that the particular sensor provides. Similar to the first alternative, in this approach, the Fusion Service is again responsible for identifying a proper technique to merge heterogeneous data. The eDOTS 2.0 uses the third option (i.e., based on the QoS), as many application domains are susceptible to specific quality requirements and various different QoS attributes (e.g., accuracy, lag time, resolution, etc.) of sensors can be specified and used in the fusion process. Also, past experiments with the eDOTS 1.0 have identified the fusion process as the most time-consuming task if such a QoS-based pruning is not performed [17,18].

TABLE I. PARTIAL KNOWLEDGE BASE – SENSOR CLASS RANKING

Sensor Modality	***Rank***
Vision (1)	1
Vision (2)	2
Wi-Fi	3

The eDOTS 2.0 uses a concept of Knowledgebase (KB), shown in Figure 1, for storing information about sensor classes, their dependencies, and the topology of an environment. A starting point for this KB is to identify different classes of sensors that are potentially deployed in each environment. Each of these sensor classes is assigned a rank based on their typical QoS attributes. For example, as shown in Table I, Webcams with higher resolutions are assigned a rank of 1, while the ones with lower resolution are assigned a rank of 2. As the typical accuracy of Wi-Fi sensors is lesser than that of any vision-based sensor, the Wi-Fi class is assigned a rank of 3. Various other classes of sensors (e.g., RFID), when added to the setup, can be assigned different ranks similarly. When a sensor instance is deployed in the indoor setup, it communicates with the Filtering Service by indicating its QoS attributes. The Filtering Service then is responsible for assigning an appropriate membership and hence, a rank for each deployed instance – a simple algorithm, indicated in Figure 3, can be used to assign ranks to various sensor instances. The Filtering Service is also responsible for gathering the physical locations of deployed sensors and augmenting the KB with this information. The Filtering Service can also collect information associated with each sensor instance based on the results received from that instance. It can use this information to dynamically change the sensor class membership and hence, the rank, of each instance during the life cycle of the eDOTS 2.0.

Sensor Ranking Algorithm (Sensors, Ranking Attribute)

Input: Array of Sensors (A[]) Ranking Attribute (Attr)
Output: Array of Ranked Sensors (A[])

1. Check to see if each *A[i]* in *A* is currently ranked.
    1. If the sensor has not previously been ranked we need to establish a rank of *null*.
    2. Otherwise, continue to Step 2.
2. Check each sensor, *A[i]*, against the providing ranking attribute, *Attr*.
3. Assign a rank based upon the ranking attribute, *Attr*, to each sensor *A[i]*.
    1. If *A[i]* exceeds the *Attr* requirement then assign a rank of *1*.
    2. If *A[i]* meets the *Attr* requirement then assign a rank of *2*.
    3. Otherwise, assign a rank of *3*.
4. Repeat Steps 2 and 3 until all sensors have been ranked.
5. Make note of each sensor ranked and its associated rank in local array *S[]*.
6. Return array of ranked sensors, in ranked order.

Figure 3. Ranking Algorithm

The KB method described above is a simple first-attempt at addressing the issues associated with multi-modal tracking and it is very static. The problems associated with multi-modal tracking include handling of mismatched quality of service properties such as different levels of spatial resolutions, different or mismatched number of degrees of freedom, and different accuracy characteristics. For example, in vision-based trackers, even though the spatial resolution tends to be high, inside-out trackers and outside-in trackers have very different accuracy characteristics. The inside-out trackers (i.e., a moving camera looking at statically placed markers in the tracked space) typically are very accurate in orientation estimation but not so good at location estimation. The outside-in trackers (i.e., multiple cameras statically installed around the tracked space, tracking a moving marker) are, in contrast, very accurate in location estimation of the marker and not very accurate in orientation estimation. Wi-Fi based trackers lack the spatial resolution of vision-based trackers, but can be more pervasive. The challenge then is how to optimally utilize the subset selection and the fusion processes so as to remove the poor performance aspects of the particular trackers and have them complement each other to provide a more accurate estimation of the tracking parameters. In addition, it is desirable to do this in a dynamic fashion. For example, markers may become occluded in the camera views in vision-based trackers in which case, it may be desirable to include less accurate trackers such as Wi-Fi based trackers to bridge the gap. The noise characteristics may also vary with time, in which case, the use of more complex fusion methods such as Kalman filter or particle filter would be more useful [1]. Utilizing machine learning techniques may also help with the dynamic aspect of subset selection [19].

B. Coordinate System Transformations and Handoffs

As any indoor tracking system encompasses sensors that are geographically dispersed, each of these sensors may have entirely different coordinate systems which they are utilizing. For instance, in a typical office building there may be many rooms on each floor. Each room may consist of a sensor that may have an established coordinate system for its own environment. This coordinate system is local to that particular environment and thus, when an object moves from one room to another a handoff and associated transformation must occur between different sensors. To remedy this problem, the eDOTS 2.0 uses the concept of the Spatial Relation Graphs (SRG) as introduced in [16]. An example of when a handoff and coordinate system transformation would be needed is shown in Figure 4.

In addition to the need for handing off, a multi-modal sensor-based setup is prone to an additional overhead when attempting to provide a coordinate system handoff and transformation between the various sensors. This additional overhead is due to the fact that different sensor modalities may have different characteristics, such as different degrees of freedom, and thus, may need specialized adaptation before applying the SRG principles as discussed in [17,18]. An example of a situation in which such adaptation is required would be if an object leaves the vision-based environment and enters an adjacent environment that contains Wi-Fi-based sensors. This handoff is required both due to the difference between the two coordinate systems that each sensor type is using and the physical movement from one environment (such as a room) to another (such as an adjacent room). In the case of the vision-based environment, a single sensor can identify an object's location, whereas in the Wi-Fi environment, three access points are needed to identify an object's position based upon triangulation.

The eDOTS 2.0 borrows the concepts proposed in [17,18] for the purpose of tracking handoff and transformation. In the Wi-Fi and Vision infrastructure, as used in the eDOTS 2.0, the transformation must be adapted for inclusion of the Wi-Fi component. The adaptation involves determining the location of the Wi-Fi access points involved in the tracking process. This information is stored in the KB when the Wi-Fi Service registers itself for tracking. We can then use this information along with the Vision-based information to begin the transformation process. The Wi-Fi component only provides data for the X and Y axis in the tracking environment and thus, the transformation process in this instance only evaluates using these data readings. Since we now know the estimated location of the tracking objects position and the physical location of the various sensors, we can now begin the transformation process as described in [17,18]. This process of adapting to a multi-modal sensor environment brings about the challenges of determining the handoff point and coordinating the transformation. In the eDOTS 2.0 we address these issues by using the highest ranked sensor environment precedence over the lower in determining this handoff point, thus the highest ranked sensor environment will dominate tracking and handoff will only occur when that environment is no longer able to provide tracking estimates. The information and coordinate information used by this higher ranked sensor modality will then be used by the lower ranked modality to perform the coordinate transformation.

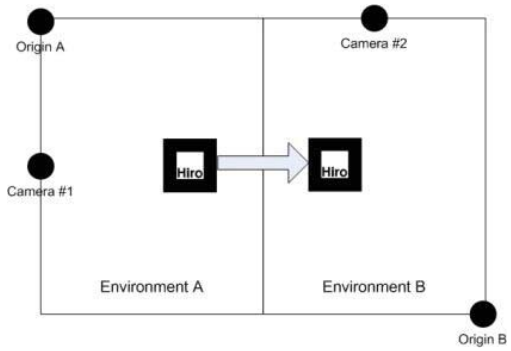


Figure 4. Coordinate Transformation & Handoff Example

III. EXPERIMENTATION

For an empirical validation, the eDOTS 2.0 is deployed in an experimental laboratory setup. The eDOTS 2.0 software is developed and run using the Eclipse IDE running Java 6. The software of the system consists of twenty-five classes and five interfaces. Software patterns were used in eDOTS 2.0 with the Adapter pattern [8] being used for the additional components involved with the Filter Service. JINI is used as the network architecture of the system for communication and discovery. The physical infrastructure includes ten Dell OptiPlex GX620 Pentium 4 machines running Windows XP SP3 with 1 GB of RAM. Each machine has two inexpensive webcams (made by Logitech and Micro Innovations) attached to it – one of the Webcams is of a better quality than the other. Additional equipment includes a HP Pavilion dv9000 Centrino Duo machine running Windows XP SP3 with 2 GB of RAM. These two devices allow for Wi-Fi tracking abilities for the eDOTS 2.0. Finally, the existing wireless infrastructure provided by Indiana University Purdue University Indianapolis (IUPUI) is used for the purpose of Wi-Fi-based tracking. This infrastructure consists of numerous access points mounted throughout the building. Physical measurements were manually taken of the locations of all stationary devices and the Wi-Fi access points. These measurements were then used to compare the accuracy of the results obtained from various experiments. This information was stored in the KB and could be accessed by the various sensors to determine their individual physical location within the setup. As these access points were physically present before the KB was developed, gathering this data and providing it in the KB was carried out as an offline activity. Through the availability of this information, it allowed for all estimations provided by the eDOTS 2.0 to be compared to the actual physical manual measurements.

Two patterns (such as the Hiro pattern shown in Figure 4), and a Wi-Fi enabled device were used as the objects to be tracked throughout the course of various experiments. The two tracking patterns were printed on 8.5 x 11 pieces of white paper and attached to a hard board. These objects were moved through the environment by an individual that entered and moved throughout the tracking environment before exiting. This sample movement was used during our experimentation and the path and types of movement were random.

A. *Experiments Related to Data Fusion In A Multi-Modality Environment*

Earlier fusion-related experiments using the eDOTS 1.0 are described in [9,17,18]. These earlier experiments with the eDOTS 1.0 used only Webcams, while the eDOTS 2.0 experimentation utilized the Webcams along with Wi-Fi sensors.

Experiment 1.1 consisted of moving the Wi-Fi enabled device throughout the lab and the adjoining hallway and tracking it only using the Wi-Fi sensors. This experiment was done over a period of five minutes and was conducted a total of ten different times. Each time the movement of the device was random. The estimated tracking error associated with an object's position when using the Wi-Fi sensors was expected to fall somewhere between 1 and 3 meters. This benchmark was taken from [17,18] with regards to the typical accuracy of current Wi-Fi based location tracking applications. The results from the Experiment 1.1 are shown in Table II. The average error in estimating the position of the tracked object, for both the X and Y axes, is calculated by comparing the physical and estimated positions, determining the error between the two, and then calculating the mean error of the collected data. These error values seem to be consistent with the accuracy of similar systems that provide Wi-Fi only location tracking such as that found in [17,18]. The average time required to compute the location of the tracked object using the Wi-Fi sensors was found to be 16 milliseconds. During the course of tracking the Wi-Fi sensor service interacted with a total of five different access points with three being required to perform the triangulation for the position estimate. In these experiments no data fusion was required since only a single set of tracking estimates were produced.

TABLE II. RESULTS OF EXPERIMENT 1.1 - AVERAGE ERROR (METERS)

X-Axis	*Y-Axis*
1.01	2.02

Experiment 1.2 was conducted to assess the ability of the eDOTS 2.0 to handle multi-modal tracking. As the Wi-Fi sensors only provide 2-dimensional readings, only these were considered while fusing them with the vision-based sensor readings. In this experiment simple averaging fusion was used. Table III shows these results obtained by fusing data from the Wi-Fi and the Vision sensors.

TABLE III. RESULTS OF EXPERIMENT 1.2 - AVERAGE ERROR (METERS)

X-Axis	*Y-Axis*
1.63	1.50

As shown in Table III, the Wi-Fi sensor dominates the overall estimation. This is consistent with the results described in Table II. Experiments 1.1 and 1.2 were both conducted without the invoking the ranking algorithm of the eDOTS 2.0. The overhead of this particular experiment was an additional 33 milliseconds. This overhead takes into account the additional time required to retrieve the Wi-Fi data from the tracking sensor as well the process of pruning

and streamlining the tracking data. This overhead was found to be caused in the Filter Service. No additional time overhead was found in the actual fusion process.

The goal of Experiment 1.3 was to evaluate if the accuracy in a multi-sensor environment could be improved through the use of the sensor ranking algorithm, which was described earlier. As there are two different classes of Webcams used in the setup, the better Webcam class (i.e., having a frame rate of 30 frames/sec and 640x480 resolution) was allocated rank 1, while the second Webcam class (i.e., having a frame rate of 15 frames/sec and 320x240 resolution) was allocated the rank 2. As the Wi-Fi class (based on the Experiments 1.1 and 1.2) provided a higher average tracking error than the Webcam class, the Wi-Fi class was given a rank of 3. Since this experiment focused solely on accuracy, timing constraints were not taken into consideration. Table IV shows the results of this experiment. It is evident from these results that the ranking algorithm improves the overall estimation when compared with the results of the Experiment 1.2. Such an observation is evident as the sensor selection process, when using the ranking algorithm, always selects the sensors with the higher rank and then uses their readings in the fusion process. In this specific experiment, the Wi-Fi sensors are not selected for the fusion process due to their lower rank and only the Webcams are selected.

TABLE IV. RESULTS OF EXPERIMENT 1.3 - AVERAGE ERROR (METERS)

X-Axis	*Y-Axis*
0.90	0.93

B. Experiments Related to the Coordinate System Handoff & Transformation

Experiment 2.1 was conducted to assess the coordinate systems handoff and associated transformations provided by the eDOTS 2.0. This handoff process involves switching from one coordinate system to another on the fly with a minimal impact seen by the Tracking Client. For Experiment 2.1, there were two possible avenues to take in order to demonstrate handoff – split the existing sensors in the lab setup into two separate "virtual" environments or to physically move the object outside of the lab to provide two physically different environments.

Initially, the lab setup was "virtually" separated into two different environments by virtue of their coordinate systems as shown in Figure 5. This meant classifying 10 of the Web cameras to be members of Environment #1 and remaining 10 Web cameras to be in Environment #2. This particular test did not include the Wi-Fi sensors in the tracking process.

Using the setup as shown in Figure 5 it was then necessary to begin the tracking process to validate the effectiveness of the eDOTS 2.0 to track an object accurately as it moves from one environment to another.

For Experiment 2.1, the accuracy of the measurements was compared with the physical measurements of the object. A final comparison was done by plotting of the data points using the graphical user interface shown in Figure 5.

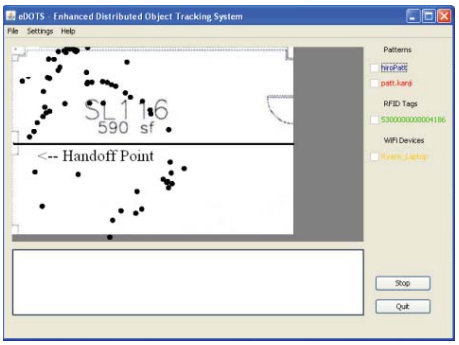


Figure 5. Environmental Setup and Coordinate System Handoff

Figure 5 shows the graphical representation of the object as it moves from one coordinate system to the next. In this experiment, the object was brought into the environment and moved in a circular pattern. This movement caused the object to cross the established boundary between the two coordinate systems, and thus forcing a handoff and transformation between the two coordinate systems. The dots plotted in Figure 5 show the track of the object as it is moved through the environment.

Experiment 2.2 was conducted in order to examine the feasibility of handoff between different classes of sensors. This was achieved by moving from an environment that made use of the vision-based sensors to an environment that made use of the Wi-Fi-based sensors. As a result, we found between 1 and 2 meters of error on both the X and Y axis when conducting this experiment. The results of this test were similar to that found with the Wi-Fi experiments discussed in Experiments 1.1 and 1.2, as the overhead created when transitioning from a more accurate Vision based environment to that of a Wi-Fi based environment is quite high and thus, the accuracy of estimation suffers.

In both experiments, the handoff and transformation from one system to another introduced an additional error while estimating the location of the tracked object. This was often the case from improper sensor readings and the different capabilities of the sensors used in the experimental setup. While improper sensor readings are not specific to a particular sensor modality we found that in the Wi-Fi scenario the use and discovery of various different access points could greatly affect the overall accuracy of the tracking data. In order to examine the accuracy of the handoff process, all of the estimated location data points were compared to their actual measurements. Once the accurate points were identified, the percentage of data points that were within the expected mean error range (as specified in [17,18]) were determined. The results of the handoff test are shown in Table V.

TABLE V. HANDOFF ACCURACY

Total Data Points	*Accurate Data Points*
86	73%

As shown in Table V, the accuracy of the handoff process was such that 73% of the total number of data points was

deemed to be accurate estimations of the object's location. Points were determined to be accurate if they fell within the bounds as indicated by the results of the Experiment 1.2. This degree of accuracy, or inaccuracy, may or may not be tolerated based upon the application domain. Future studies to compare both the data fusion and transformation are needed.

IV. Related Work

In recent past, many indoor tracking systems have been proposed, both homogenous [1,2,3,10,20,22,24] and heterogeneous [6] in nature. The eDOTS 2.0 differs from these exploratory systems in that it is designed with the goal of effectively tackling heterogeneity between sensor classes. We have also focused on the overall accuracy of the system while attempting to meet real-time constraints of 30 milliseconds. Extensive work has been done in the area of multi-sensor data fusion, such as [21]. In addition to this work, studies have also been conducted on the Federated Kalman Filter and its practical application [3] as it applies to multi-sensor environments. Klinker et al. at the Technical University of Munich have studied ubiquitous tracking in distributed environments [4,11,13,16]. Our work is an extension of this approach, as we have applied these various techniques for the purpose of tracking in an indoor multi-sensor environment. Finally, contributions have been made with respect to using SRG and their application in various different domains [15]. Our work has demonstrated the feasibility and effectiveness of using SRG for the purposes of coordinate system handoffs and associated transformations in multi-modal sensor-based indoor setups. Also, the selection of a subset of sensors, based on an associated rank determined by their QoS attributes, is a novel concept used by the eDOTS 2.0 for the purposes of indoor tracking.

V. Conclusion

This paper has discussed the design, implementation, and experimentation of an indoor tracking system. The contributions of this work are: a) an ability to handle multi-modal sensors, and b) a seamless handoffs and associated transformations. Future work includes an experimentation and analysis using additional sensor modalities, such as the RFID tags and sensors on mobile devices, and investigation of different algorithms for sensor selection and fault tolerance of sensors services.

References

[1] Arulampalam, M. S.; Maskell, S.; Gordon, N.,Clapp, T. A Tutorial on Particle Filters for Online Nonlinear/Non-Gaussian Bayesian Tracking IEEE Transactions on Signal Processing, 2002, 50, 174-188.

[2] Bagci F, Kluge F, Bagherzadeh N, Ungerer T. "LocSens - An Indoor Location Tracking System using Wireless Sensors." In: Computer Communications and Networks, 2008. ICCCN '08. Proceedings of 17th International Conference on , vol., no., pp.1-5, 3-7 (2008).

[3] Bahl P, Padmanabhan, V. (2000): "Radar: An In-Building RF-based User Location and Tracking System", Proceedings of INFOCOM 2000, IEEE Conference on Computer Communications, Tel Aviv, Israel.

[4] Bauer, M.; Bruegge, B.; Klinker, G.; MacWilliams, A.; Reicher, T.; Sandor, C., Wagner, M. An Architecture Concept for Ubiquitous Computing Aware Wearable Computers Proceedings of the International Conference on Distributed Computing Systems Workshops, IEEE Computer Society, 2002, 785-790.

[5] Carlson N. "Federated square root filter for decentralized parallel processors." Aerospace and Electronic Systems, IEEE Transactions on 26, no. 3 (1990): pp. 517-525.

[6] Ching-Sheng W, Li-Chieh C. "RFID & vision based indoor positioning and identification system," Communication Software and Networks (ICCSN), vol., no., pp. 506-510, (2011).

[7] Dorner R., "Accuracy in optical tracking with fiducial markers: an accuracy function for ARToolKit, Mixed and Augmented Reality," ISMAR 2004. Third IEEE and ACM International Symposium, 2004.

[8] Gamma E, Helm R, Johnson R, Vlissides J. "Design patterns: Elements of reusable object-oriented design." (1995): 1-30.

[9] Girish G, Rajeev R., Mihran T. "Designing and Experimenting with a Distributed Tracking System." ICPADS 2008: 64-7.

[10] Gustafsson F, Gunnarsson F. (2005): "Mobile Positioning Using Wireless Networks," IEEE Signal Processing Magazine, vol. 22, no. 4, pp. 41–53.

[11] Huber, M.; Pustka, D.; Keitler, P.; Echtler, F. & Klinker, G. A System Architecture for Ubiquitous Tracking Environments ISMAR '07: Proceedings of the 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality, IEEE Computer Society, 2007, 1-4.

[12] Kato H., Billinghurst M. "Marker Tracking and HMD Calibration for a video-based Augmented Reality Conferencing System," In Proceedings of the 2nd International Workshop on Augmented Reality (IWAR 99). October, San Francisco, USA.

[13] Keitler, P.; Pustka, D.; Huber, M.; Echtler, F. & Klinker, G. Dubois, E.; Gray, P. & Nigay, L. (Eds.) Management of Tracking for Mixed and Augmented Reality Systems The Engineering of Mixed Reality Systems, Springer London, 2010, 251-273.

[14] Malbezin, P, Piekarski, W, Thomas, B. "Measuring ARToolKit Accuracy in Long Distance Tracking Experiments," In ART02, 1st International Augmented Reality Toolkit Workshop; 2002.

[15] Nagpal, R, Shrobe, H, Bachrach, J. "Organizing a Global Coordinate System from Local Information on an Ad Hoc Sensor Network," 2003.

[16] Newman, J.; Wagner, M.; Bauer, M.; MacWilliams, A.; Pintaric, T.; Beyer, D.; Pustka, D.; Strasser, F.; Schmalstieg, D. & Klinker, G. Ubiquitous Tracking for Augmented Reality IEEE / ACM International Symposium on Mixed and Augmented Reality (ISMAR 2004), IEEE Computer Society, 2004, 192-201.

[17] Rybarczyk R, Raje R, Tuceryan M. "Enhancing a Distributed Tracking System." In: Proceedings of 3rd International Joint Conference on Information and Communication Technology (IJcICT-2011). Mumbai, India; (2011). 7 p.

[18] Rybarczyk R, Raje R, Tuceryan M. "A Heterogeneous Indoor Tracking System." Indiana University Purdue University Indianapolis – Computer Science Department; (2012). TR-CIS-0125-12.

[19] Tilak, O.; Mukhopadhyay, S.; Tuceryan, M., Raje, R. A Novel Reinforcement Learning Framework for Sensor Subset Selection 2010 International Conference on Networking, Sensing and Control (ICNSC), 2010, 95 -100.

[20] Varshavsky A. "The SkyLoc Floor Localization System." In: Proceedings of the IEEE PerCom, (2007).

[21] Varshney P. "Multisensor data fusion." Electronics & Communication Engineering Journal 9.6 (1997): pp. 245-253.

[22] Want, R, Hopper, A, Falcão, V, Gibbons, J. (1992): "The Active Badge Location System", ACM Transactions on Information Systems, vol. 10, no. 1, pp. 91–102.

[23] Welch, G., Bishop, G. An Introduction to the Kalman Filter Department of Computer Science, University of North Carolina at Chapel Hill, 1995.

[24] Xiang Z., Song S, Chen, J, Wang, H, Huang, J, Gao, X. (2004): "A Wireless LAN-Based Indoor Positioning Technology", Journal of Research Development, vol. 48, no. 5/6, pp. 617–626.

A context-aware approach on semantic trajectories

Caio Silva and M.A.R Dantas
Department of Informatics and Statistic (INE)
Federal University of Santa Catarina (UFSC)
88040-900 Florianpolis, SC Brazil
caio.cmsilva@gmail.com, mario.dantas@ufsc.br

***Abstract*—Context-sensitive systems can use geographical position at a specific time to extract knowledge and determine patterns in user trajectories. However, this data can be unreliable or inaccurate, thus undermining the trajectories pattern inference process. Some researches adopts a procedure to eliminate redundancy and conflicts through the utilization of a centralized inference server. Other studies adopt a post-processing procedure which is characterized by adding semantics to the spatial-temporal data. Both efforts consume unnecessary computational resources. Context data represents the state of an entity in a given time period, it provides information that allow to infer user behavior patterns with a set of semantics. In this research, it is proposed a differentiated context-aware approach to filter redundant and conflicting context information. The proposal aims to improve the inference process that determines patterns on user trajectories. The approach uses the processing power and storage capabilities of mobile devices to conduct operations to validate context data, thus eliminating useless information. Experimental results show an improvement both in the representation of spatial-temporal data and also on the semantic load that characterize the user trajectories.**

Keywords-context-awareness; semantic trajectories; spatial-temporal; pervasive computing;

I. INTRODUCTION

Nowadays, mobile devices and network sensors have considerable processing power, storage space and wireless communication capabilities. In addition to that, this equipment has diverse functionalities and interfaces as GPS, radio, TV, audio players, and digital cameras. These features have a strong link with the physical world, and the information provided by them help define the user profile.

Context-awareness permits mobile services to dynamically and efficiently adapt both to the current situation, such as current physical place and/or social activity, and to the challenging and highly variable deployment conditions typical of mobile environments (e.g. resources scarcity, unreliable and intermittent wireless connectivity) [1]. A context-awareness approach can, also, minimize power consumption, processing time and the amount of context data propagated over the network, providing more accuracy and adaptability to user applications.

Context is any information that can be used to characterize the situation of an entity such as person, and which is considered relevant to the interaction between a user and an application, including the user and applications themselves [2].

However, spatial-temporal data are normally represented as sample points, in the form *(tid,x,y,t)*, where *tid* represents the trajectory identifier and *x,y* correspond to a position in space at a certain time *t*. Trajectory sample points have very little or no semantics, and therefore it becomes very hard to discover interesting patterns from trajectories in different application domain [3].

Context data can hold diverse information about a user situation, depending on the context model used. The historical of context data can be used to infer user behavior patterns with a large semantic bias. On the other hand, context information may not be reliable or useful. Thus, an important point in context sensitivity is related to the information reliability, meaning that the quality of context should be ensured [4].

Accordingly, GPS information can be inaccurate or incorrect, depending on the amount of surrounding buildings, network connection, limitation of the device itself, among others. Under these circumstances, the GPS produces a large amount of redundant and conflicting data (e.g. when the vehicle stops or when there is an interference in the satellite signal). Such information are not required to be sent through the network to a inference server, the mobile device itself can take some responsibility on validating the data produced.

Some researches, as shown in [5], adopts a procedure to eliminate redundancy and conflicts through the utilization of a centralized inference server. Other studies, such as those found in [3], [6], [7] and [8], adopt a post-processing procedure which is characterized by "cleaning" the trajectory and adding semantics to the spatial-temporal data. Both efforts consume unnecessary computational resources.

In this paper a differentiated approach is presented. The research proposal considers the filter process of context data before sending them to the inference server. The information producer has the responsibility to remove redundant and conflicting data. In addition, to perform some validations based on user preferences. Consequently, the inference server can fetch user behavior patterns with more accuracy and fewer failures.

The remainder of this paper is organized as follows. Section II presents some basics concepts about trajectories and context-awareness. The proposed approach is described in section III, where there is redefinitions on trajectory concepts and pseudo

codes of procedures used. In section IV it is presented a set of experimental results. Finally, conclusions and discussions of future work are shown in section V.

II. BASIC CONCEPTS

This section presents the basic concepts involving trajectories and context-awareness, which represent the state-of-art and can provide a basis for a future comparison with our proposal.

A. Trajectory Concepts

Alvares [5] introduces a formal model to represent semantic trajectories, using stops and moves to integrate geographic information to trajectory sample points. In a subsequent work, Alvares [6] proposes a framework to preprocess trajectories for semantic data analysis and data mining. The formal definitions that characterize a semantic trajectory are presented below.

Let R denote the set of real numbers. The definitions are restricted to movements in the real plane R^2. Space-time space will be denoted R^2 x R, where the first two dimensions represent space and the latter represents time. Typically, it is agreed that x, y are variables ranging over spatial coordinates and t a variable that ranges more points in time.

Definition 1. *A sample trajectory is a list of space-time points* $((x_0 , y_0 , t_0), (x_1 , y_1 , t_1), \ldots ,(x_n , y_n , t_n))$ *where* $x_i , y_i, t_i \in R$ *for* $i = 0, \ldots , N$ *and* $t_0 < t_1 < \cdots < t_n$.

Definition 2. *A candidate stop C is a tuple* $(R_c , \Delta c)$*, where* R_c *is a* (*topologically closed*) *polygon in* R^2 *and* Δc *is a strictly positive real number. The set* R_c *is called the geometry of the candidate stop and* Δc *is called its minimum time duration.*

An application A is a finite set $\{C_1 = (R_{C1} , \Delta C_1), \ldots , C_N = (R_{Cn} , \Delta C_n)\}$ *of candidate stops with mutually non overlapping geometries* $R_{C1} , \ldots , R_{Cn}$.

Definition 3. *Let T be a trajectory and let A be an application as defined below.*

$$A = \{C_1 = (R_{C1} , \Delta C_1), \ldots , C_N = (R_{Cn} , \Delta C_n)\}$$

Suppose we have a subtrajectory of T $\{(x_i , y_i , t_i), (x_{i+1} , y_{i+1} , t_{i+1}), (x_{i+\mu} , y_{i+\mu} , t_{i+\mu})\}$*, where there is a* $(R_{Ck} , \Delta C_k)$ *in A such that* $\forall j \in [i, i + \mu] : (x_j , y_j) \in R_{Ck}$ *and* $t_{i+\mu} - t_i \geq \Delta C_k$ *, and this subtrajectory is maximal (with respect to these two conditions), then the tuple* $(R_{Ck} , t_i , t_{i+\mu})$ *is defined as a stop of T with respect to A.*

Definition 4. *A Semantic Trajectory S is a finite sequence* $\{I_1 , I_2 , \ldots , I_n\}$ *where* I_k *is a stop or a move* **.**

For the sake of finite representability, it is assumed that the space-time points (x_i , y_i , t_i), have rational coordinates. This will be the case in practice, since these points are typically the result of observations.

The set of definitions presented formally characterizes semantic trajectories. However, with this approach, it is necessary to post-process all trajectory data to add semantics. It would be better to work with useful data that already have intrinsic semantics. In next subsection we introduce some concepts about context-awareness and context data.

B. Context Concepts

Context is still a vague concept to identify the aspects the designer considers useful to model and describe the environment where a given service is to be deployed and executed [1]. There are several classifications of context in the literature such as those found in [2] [9] [10] and [11]. We adopted the context definition presented in [12], where context is a four-dimensional space composed by:

- Computing context: Deals with all those technical aspects related to computing capabilities and resources;
- Physical context: Groups all those aspects that represent real world and that are accessible by using sensors/resources deployed in the node surroundings;
- Time context: Captures the time dimension, such as time of a day, week, month, and season of the year, of any activity performed in the system (either real-world or computing);
- User context: contains high-level context aspects related to the social dimension of users, such as users profile, people nearby, and current social situation;

Thus, considering all the context dimensions mentioned, different context behaviors can be accomplished to adapt services to make them suitable for the end user and to conform current execution environment characteristics. For this purpose, quality of context data is a key issue, since it may impair the correctness of adaptations operations.

Quality of Context (QoC) is any information that describes the quality of the information being used as context information [13]. Models of quality of context and architectures have the role of establishing the flow of context information, from producer to consumer. In addition, these models define where the context parameters will be evaluated to ensure the desired quality of context.

Hence, similarly to context-awareness in itself, a well-accepted QoC definition is still missing. Several authors presented their own QoC framework, also introducing and using the same concepts with different names. However, despite these differences, a common thought can be highlighted: QoC is not requiring perfect context data, such as all data with the highest possible precision and up-to-dateness, but having and maintaining a correct estimation of the data quality [13].

Quality context data can be used in a great range of applications, and as well in trajectory pattern discovery. Our proposal goal is to use in a better fashion the information provider (mobile device/network sensors), sending only relevant information to the server and, consequently, optimizing the inference process that determine patterns on user trajectories. In addition, the approach also aims to improve the semantic structure of users trajectories through the use of context data.

III. PROPOSAL

The architecture on [14] seeks to explore aspects of processing and storaging of mobile devices to remove redundancies and conflicts. The information producer provides

pre-validated data to the server. This approach reduces the total data sent through network, and as a consequence is used less computational resources to infer behavior patterns. Besides that, the induction process can improve, since it uses consistent data that has a high load semantics. The architecture is shown in figure 1.

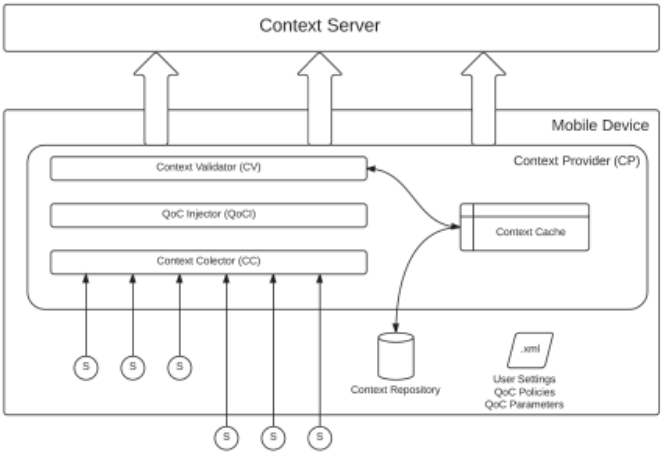


Figure 1. Quality-Aware Context Producer [14]

A. Components

This section presents the components of the proposed model describing their functionality and comprisement.

1) ***Context Collector:*** *Context Collector (CC) module is the interface between the sensors and the Quality-Aware Context Producer. This module has the responsibility to collect and aggregate context information provided by sensors, transforming context information into a context object.*

2) ***Quality of Context Injector:*** *Quality of Context Injector (QoCI) inject user defined quality parameters in the context object. When some of the required context parameters is not filled, QoCI invalidates and dismisses the context object.*

3) ***Context Validator:*** *Context Validator (CV) validates context data by removing redundancy and conflicts between context objects. The module uses the user-defined policies to invalidate context information. After ensuring quality on the new context object, CV query the cache collecting all context objects recently sent to find and remove redundancies. Moreover, CV try to identified conflicts in context objects, i.e. find inconsistencies between information provided by different context objects, and also remove them.*

4) ***Context Cache:*** *Context Cache module (CCa) stores runtime valid context objects. Furthermore, the module has the responsibility to communicate with the context server (inference server), sending valid context objects and requesting context information when necessary. CCa uses mobile device storage to save context information, thus reducing the number of requests to the server.*

In ubiquitous mobile environments, it may happen that the context producer is also a context consumer. Applications can adapt more quickly using the information stored in a context cache, and consequently fewer accesses are made on context server.

B. Context-Aware Semantic Trajectories

This section presents a redefinition of concepts that encompass semantic trajectories, in order to better exploit the benefits of using context data.

Definition 5. *A context data CT is a tuple* $\{(x_i , y_i), T, (a_i , b_i , c_i , \ldots)\}$, *where* (x_i , y_i) *is a spatial point (latitude and longitude), T is the time context and* $(a_i , b_i , c_i , ...)$ *is a set of environment attributes.*

Time context T and the set of environment attributes (ai, bi , ci , ...) are defined considering the needs of a application. The environment attributes are extracted from computing, physical and user context. The application context model will indicate which parameters are relevant to define a context data. The spatial point (x_i , y_i) is part of the physical context, however, as the focus of this paper is semantic trajectories, context data without spatial information are considered invalid.

Definition 6. *A stop S is a subset* $\{(x_i , y_i), t_c , t_s , (a_i, b_i, \ldots)\}$ *of a context data, where* (x_i , y_i) *is a spatial point,* t_c *represents the time which the data was captured,* t_s *indicate the time that the moving object stood at* (x_i , y_i), *and* $(a_i, b_i, \ldots)$ *are the semantic attributes that describe a particular stop.*

Unlike the definition presented in [5], in our approach any valid trajectory point became a stop, and t_c represents the extent of the stop. [5] define a stop as a polygon, this definition only makes sense because, nowadays, there are technological limitations in capturing spatial information. In the definition presented here, a stop is a point, and every point is a stop. In other words, a moving object stops everywhere it passes, but t_s may vary from a few milliseconds to several hours.

Definition 7. *A semantic trajectory ST is a finite sequence* $\{S_1, S_2 , S_3 , \ldots , S_n \}$ *where* S_i *is a stop.*

A *move* is considered a characteristic of the moving object, which can be inferred given at least two distinct stops from a trajectory. For the presented approach, *moves* do not characterizes a trajectory, since the locomotion characteristic belongs to moving objects.

C. *Algorithms*

In this section is presented the pseudo-code of algorithms used to remove redundancies and conflicts. The proposed algorithms are based on those proposed in [15].

```
<?xml version= "1.0"?>
<QoCPolicies>
  <Policy>
    <ParameterName>Latitude</ParameterName>
    <ParameterName>Longitude</ParameterName>

    <QualityParameter>
    <Name>Precision</Name>
    <And>
      <IsLessThan>1</IsLessThan>
      <IsNull>false</IsNull>
    </And>
   </QualityParameter>
 <Policy>

 <Policy>
     <ParameterName>DistanceDifference</
         ParameterName>
     <IsGreaterThan>200</IsLessThan>
     <IsNull>false</IsNull>
 <Policy>

 <Policy>
    <ParameterName>TimeDifference</
        ParameterName>
    <IsGreaterThan>10</IsGreaterThan>
 <Policy>
</QoCPolicies>
```

Figure 2. QoC Policies

```
void resolveConflict(ContextObject co1,
    ContextObject co2){
      p = getConflictPolicies(co1);
      validCo = applyPolicy(co1, co2, p);
      sendToContextCache(validCo);
}
```

Figure 3. Resolve conflict method

```
boolean equals(ContextObject co1, ContextObject
    co2){
  if(c01.getSpatialLocation().equals(co2.
      spatialLocation())){
     if(co1.getAllContexts().equals(co2.
         getAllContexts())){
        return true;
     }
     else
        resolveConflict(co1, co2);
  }
  return false;
}
```

Figure 4. Equals method

Equals method compare all context dimensions from two distinct context objects and verifies if these objects represent redundant information. *Remove Redundancy* compare the recently sensed context object with all context objects that are in cache. Once it is identified a redundancy, the algorithm get the time difference between the two context objects and adds it to the stopped time t_s of the valid context object. In addition, it eliminates the redundant object. Resolve conflict method is used when there is a indefinition on contexts comparison. The method utilizes a user-defined policy which characterizes context data conflicts and redundancies to resolve inconsistencies between context objects. The algorithms can be seen in figures 3, 4 and 5.

```
void removeRedundancy(ContextObject co){
   cacheObjects = getCtxCacheObjects();
   for i from 0 to cacheObjects.length{
      if(cacheObjects[i].equals(co)){
       t = getTimeDifference(cacheObjects[i], co);
       cacheObjects[i].addStoppedTime(t);
       delete(co)
      }
      else
        sendToContextCache(co);
   }
}
```

Figure 5. Remove redundancy method

IV. EXPERIMENTAL RESULTS

In this section, it is described the environment and experimental results related to our tests. As a study case, we used real GPS data collected from trajectories of a user walking and cycling. The goal was to identify patterns from several routes. Our focus is to ascertain the impact of removing redundancies and conflicts in trajectory data before being sent to the server. The experimental environment is presented on table 1.

The user was monitored during four marathons trainings and one long distance cycling. The GPS was configured to convey information at an interval of 4-second during the entire trajectory.

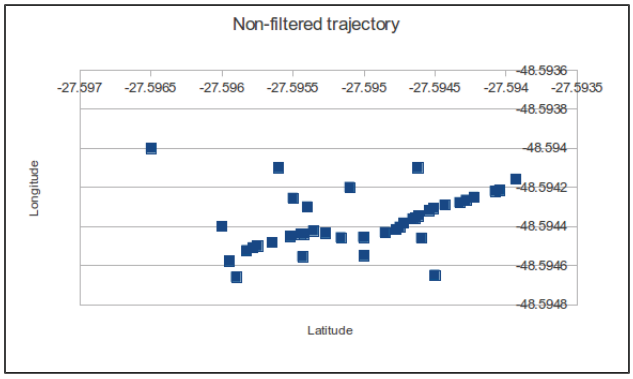


Figure 6. Non-Filtered trajectory: User walking

[14] presents a model that reads the user policies file to perform the validation process. Policies are defined through logic elements (or, and, not, =, ...). A context parameter can have a set of policies that are used by CV to remove existing conflicts. A user policy is defined by a parameter name, a set of

	Operational System	Processor	Memory	Storage	Database	Programming language	Environment
Context Server	Android 4.1	Quad-core 1.4Ghz	512MB	16GB	GvSig 0.3	Java 1.7	Android Emulator
Context Provider	Ubuntu 12.04	Intel i5 2.5GHz	4GB	1TB	PostGis 2.0	Java 1.7	Dell Vostro 3460

quality parameters and a set of logical expressions. The adopted policies that characterize conflicts and redundancies and that were used to perform the tests are presented below: (i) the accuracy of the GPS data is at most 10 meters, (ii) the distance difference between two consecutive GPS data must be more than 200 meters and (iii) the time difference between two consecutive GPS should be greater than 10 seconds. The user policies file used on tests is presented in figure 2.

The context parameters shown on figure 2 are extracted from the context model used by the application. The user policy schema is defined in [14].

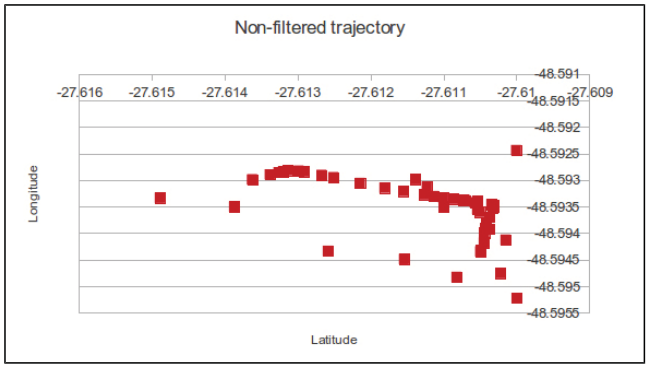


Figure 7. Non-Filtered trajectory: User cycling

Figure 6 e 7 shows data without any context filtering procedure. Unfiltered data represent trajectories that differ from the trajectory actually made by user. Both figures shows pieces of the trajectory with stops and signal loss. Figure 6 shows shows a part of the marathon trajectory which happened a stop on, and figure 7 shows a stretch of the trajectory where there was loss of the GPS signal. It is noticed that with unfiltered data is hard to see the route itself, the data set does not seem to represent a trajectory.

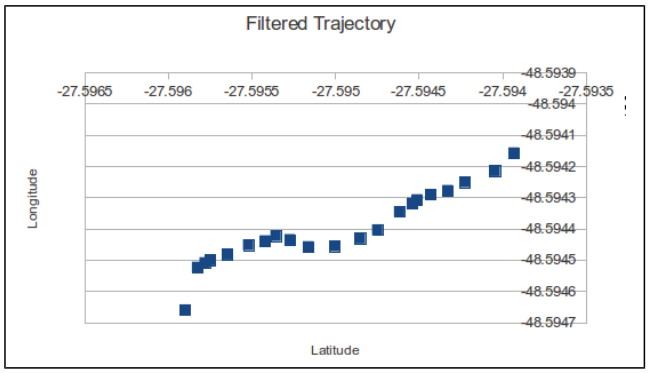


Figure 8. Filtered trajectory: User Walking

Figure 8 e 9 shows the same routes shown in figure 6 and 7, however, with filtered data. The amount of data representing the trajectory decreased by approximately 40%. The points outside the trajectory were largely eliminated. The representation of trajectories with filtered data provide a better reference for the actual trajectory taken by the user. Furthermore, the amount of data that is processed by the inference algorithms decreases.

For a better visibility, we chose not to represent the trajectories as a whole because, on average, each trajectory is represented by 4000 points. For this reason, we chose to evaluate an excerpt from the trajectory which had stops and signal loss situation, since these stretches of the trajectory concentrate a large number of redundant and conflicting data.

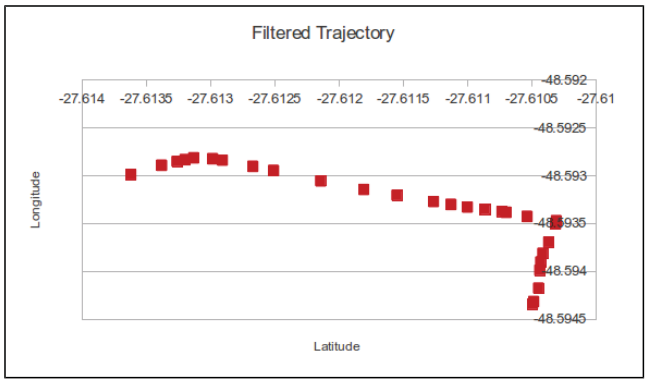


Figure 9. Filtered trajectory: User cycling

V. Conclusions and Future Work

In this paper it was presented a research work which main goal was to enhance the filter process of GPS data before sending them to the inference server context. Our hypothesis was based on the fact that the information producer has the responsibility to remove redundant and conflicting data in a more efficient fashion procedure. In addition, to perform some validations based on user preferences. Consequently, the inference server can fetch user behavior patterns with more accuracy and fewer failures. In other words, the study proposes a context-aware path analysis in semantics.

The approach was based on the fact that context data has intrinsic semantics and so can better represent the trajectory of the user. Filters of mobile devices and network sensors are responsible for all context objects, eliminating redundancies and conflicts before the data are sent to a server. Experimental results adopting a context-aware approach showed an improvement in trajectory representation, both in the

representation of spatial-temporal data as well as on the semantic load that characterize the user trajectories.

As a future work, we are planning to better exploit semantic aspects that involve context data in user trajectories. Accordingly, we intend to model behavioral aspects from users in order that the inference algorithms can, more efficiently, find user behavior patterns.

REFERENCES

[1] P. Bellavista, A. Corradi, M. Fanneli, and L. Foschini, "A survey of context data distribution for mobile ubiquitous systems," ACM Computing Surveys, vol. 45, no. 1, pp. 1–49, 2013.

[2] A. Dey, "Providing architectural support for building context- aware applications," Ph.D. dissertation, Georgia Institute of Technology, 2000.

[3] V. Bogorny and M. Wachowicz, "A framework for context- aware trajectory data mining," Data Mining for Business Applications, pp. 225–240, 2008.

[4] S. Kim and H. Lee, "The impact of organizational context and information technology on employee knowledge-sharing capabilities," Public Administration Review, vol. 66, no. 3, pp. 370–385, 2006.

[5] L. O. Alvares, V. Bogorny, B. Kuijpers, J. MACEDO, B. Moe- lans, and A. Vaisman, "A model for enriching trajectories with semantic geographical information," 2007.

[6] L. O. Alvares, G. Oliveira, C. A. Heuser, and V. Bogorny, "A framework for trajectory data preprocessing for data mining," in SEKE. Knowledge Systems Institute Graduate School, 2009, pp. 698–702.

[7] Z. Yan, D. Chakraborty, C. Parent, S. Spaccapietra, and K. Aberer, "Semitri: a framework for semantic annotation of heterogeneous trajectories," in Proceedings of the 14th International Conference on Extending Database Technology. ACM, 2011, pp. 259–270.

[8] V. Bogorny, B. Kuijpers, and L. O. Alvares, "St-dmql: A se- mantic trajectory data mining query language," International Journal of Geographical Information Science, vol. 23, no. 10, pp. 1245–1276, 2009.

[9] B. Schilit, N. Adams, and R. Want, "Context-aware computing applications," in Mobile Computing Systems and Applications, 1994. WMCSA 1994. First Workshop on. IEEE, 1994, pp. 85–90.

[10] A. Zimmermann, A. Lorenz, and R. Oppermann, "An operational definition of context," Modeling and using context, pp. 558–571, 2007.

[11] C. Bolchini, C. Curino, E. Quintarelli, and F. Schreiber, "Context information for knowledge reshaping," International Journal of Web Engineering and Technology, vol. 5, no. 1, pp. 88–103, 2009.

[12] G. Chen, D. Kotz et al., "A survey of context-aware mobile computing research," Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College, Tech. Rep., 2000.

[13] T. Buchholz, A. K¨ pper, and M. Schiffers, "Quality of conutext: What it is and why we need it," in Proceedings of the workshop of the HP OpenView University Association, 2003.

[14] C. Silva and M. Dantas, "Quality-aware context producer: A energy-efficient approach for context-sensitive systems on ubiquitous environment," submitted to the ISCC 2013.

[15] A. Manzoor, H. Truong, and S. Dustdar, "Quality aware context information aggregation system for pervasive environments," pp. 266–271, 2009.

Towards Quantifying Quality, Tactics and Architectural Patterns Interactions

Mohamad Kassab
The Pennsylvania State University
Malvern, PA, U.S.A
muk36@psu.edu

Ghizlane El Boussaidi
École de technologie supérieure
Montreal, QC, CANADA
ghizlane.elboussaidi@etsmtl.ca

***Abstract*—Architectural design involves choosing the best design solution that satisfies a set of requirements. During this process the architect has to assess and compare multiple, and possibly conflicting, criteria and decisions including quality attributes, architectural tactics and patterns. While architectural patterns embody high level design decisions, an architectural tactic is a design strategy that addresses a particular quality attribute. Tactics, in fact, serve as the meeting point between the quality attributes and the software architecture. To guide the architect in selecting the most appropriate architectural patterns and tactics, the interactions between quality attributes, tactics and patterns should be analyzed and quantified and the results should be considered as decision criteria within a quality-driven architectural design process. In this paper, we propose an approach for a quantitative evaluation of the support provided by a pattern for a given targeted set of quality attributes. Our approach incorporates the mathematical based trade-off technique: Analytical Hierarchy Process (AHP) to quantitatively deal with ambiguities, trade-offs, priorities and interdependencies among qualities, tactics and architectural patterns.**

Keywords- architectural patterns; architectural tactics; quality requirements; architectural design, AHP.

I. INTRODUCTION

When designing software architectures, an architect relies on a set of idiomatic patterns commonly named architectural styles or patterns. A software architectural pattern defines a family of systems in terms of a pattern of structural organization and behavior [1]. More specifically, an architectural pattern determines the vocabulary of components and connectors that can be used in instances of that pattern, together with a set of constraints on how they can be combined. Common architectural patterns include Layers, Pipes-Filters, Model View Controller (MVC), Broker and Client-Server. Architectural patterns have been described using different frameworks (see e.g. [1, 2, 3, 4, 5]).

A common framework to describing patterns includes the name, the problem, the structure and the dynamics of the solution, and the consequences of using a pattern expressed in terms of its benefits and its liabilities. Hence the selection of an architectural pattern is qualitatively driven by the properties it exhibits [6] and the architect's knowledge and experience with patterns.

While architectural patterns embody high level design decisions, an architectural tactic [7] is a design strategy that addresses a particular quality attribute. Tactics are a special type of operationalization that serves as the meeting point between the quality attributes and the software architecture. In [8], the authors define an architectural tactic as an architectural transformation that affects the parameters of an underlying quality attribute model. Implementing a certain tactic within a pattern may affect the pattern by modifying some of its components, adding some components and connectors, or replicating components and connectors [9].

Dealing with decisions concerning the selection of an architectural pattern for a software project or module involves trade-off of some sort. This is mainly because one of the primary goals of the architecture of a system is to create a system design to meet the required functionality, while satisfying the desired quality attributes. In fact, if we consider a system's architecture as a set of architectural decisions, the most significant ones concern the satisfaction of quality attributes [10]. Qualities, on the other hand, can often interact, in the sense that attempts to achieve one quality attribute can help or hinder the achievement of other quality attribute at particular software functionality. Such an interaction typically creates an extensive network of interdependencies among quality attributes which is not easy to trace or estimate [11].

In this paper, we propose an approach for a quantitative evaluation of the support provided by a pattern for a given targeted set of quality attributes. The main research question that we address in this paper is: "Given a set of software qualities to be implemented within a system, what will be the most accommodating architectural pattern to choose to design the system?" Our approach incorporates the mathematical based trade-off technique: Analytical Hierarchy Process (AHP) [12, 13] to quantitatively deal with ambiguities, trade-offs, priorities and interdependencies among qualities, tactics and architectural patterns.

The remaining of this paper is organized as follows: Section II provides the background on quality attributes, architectural tactics, patterns, and the AHP technique. In section III we start by the quantitative mapping of quality attributes into architectural tactics; using AHP technique, where we consider not only the positive contributions of tactics to implement the qualities but also the potential conflicts among qualities due to the introduced tactics. In section IV we relate the architectural

patterns to the tactics based on evaluations of the impact of incorporating the latter within the former. We then propose; in section V, the usage of the AHP technique again to quantify the relations between software qualities and architectural patterns using the results from sections III and IV. Section VI discusses related work and finally, Section VII concludes the paper.

II. BACKGROUND

A. Quality attributes, tactics and patterns

Quality is "the totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs" [14]. Software Quality is an essential and distinguishing attribute of the final product. Tactics on the other hand are measures taken to improve the quality attributes [7]. For example, introducing concurrency for a better resource management is a tactic to improve system's Performance. In [7], the authors list the common tactics for the qualities: Availability, Modifiability, Performance, Security, Testability and Usability. In this paper, we will illustrate our approach via considering the incorporation of Security and Performance qualities into a software system along with their derived tactics as proposed in [7] and shown in Table I.

TABLE I. TACTICS FOR SECURITY AND PERFORMANCE [7]

Security Tactics		Performance Tactics	
T1	Authenticate Users	T9	Increase Computation Efficiency
T2	Authorize Users	T10	Reduce Computational Overhead
T3	Limit Access	T11	Manage Event Rate
T4	Limit Exposure	T12	Control Frequency of Sampling
T5	Maintain Data Confidentiality	T13	Introduce Concurrency
T6	Intrusion Detection	T14	Maintain Multiple Copies
T7	Identification of Attacker	T15	Increase Available Resources
T8	Restoration	T16	Scheduling Policy

Typically, systems have multiple important quality attributes, and decisions made to satisfy a particular quality may help or hinder the achievement of another quality attribute. The best-known cases of conflicts occur when the choice of a tactic to implement certain quality attribute contributes negatively towards the achievement of another quality. For example, decisions to maximize the system Reusability and Maintainability through the usage of "Abstracting Common Services" tactic may come at the cost of the "Response Time".

Tactics are considered as the building blocks from which architectural patterns are composed [7]. While an architectural pattern is commonly defined by its components, their interactions and interrelationships and their semantic, its implementation includes a combination of tactics depending on the pattern's objectives. For example, the Broker pattern implements the Modifiability tactic "Use an Intermediary" by introducing the Broker component which acts as an intermediary between clients and servers to provide location-transparent service invocations [8]. Nevertheless patterns can impact individual tactics by making it easier or more difficult to implement them [9]. Indeed the changes due to a tactic's implementation within a pattern may at worst-break the pattern's structure and/or behavior. Therefore, architects must deal with tradeoffs, priorities and interdependencies between qualities, tactics and patterns before selecting some patterns and tactics that may achieve one quality and hinder another.

B. The AHP technique

The AHP technique [12, 13] was introduced by Thomas L. Saaty in the 1970s. AHP is a technique for modeling complex and multi-criteria problems and solving them using a pairwise comparison process. Simply described, AHP breaks down a complex and unstructured problems into a hierarchy of factors that can be judged according to their relative importance to determine which actions have the highest priority to achieve the desired goal.

The AHP process starts by a detailed definition of the problem; goals, all relevant factors and alternative actions are identified. The identified elements are then structured into a hierarchy of levels where goals are put at the highest level and alternative actions are put at the lowest level. Usually, an AHP hierarchy has at least three levels: the goal level, the criteria level, and the alternatives level. This hierarchy highlights relevant factors of the problem and their relationships to each other and to the system as a whole [13]. Once the hierarchy is built, involved stakeholders (i.e., decision makers) judge and specify priorities of the elements of the hierarchy. To establish the priorities of elements of the problem, a pairwise comparison process is used. This process starts at the top of the hierarchy by selecting an element (e.g., a goal) and then the elements of the level immediately below are compared in pairs against the selected element. A pairwise matrix is built for each element of the problem; this matrix reflects the relative importance of elements of a given level with respect to a property of the next higher level. Saaty proposed the scale [1...9] to rate the relative importance of one criterion over another (See Table II). Based on experience, a scale of 9 units is reasonable for humans to discriminate between preferences for two items [13].

TABLE II. PAIRWISE COMPARISON SCALE FOR AHP [13].

Intensity of judgment	Numerical Rating
Extreme Importance	9
Very Strong Importance	7
Strong Importance	5
Moderate Importance	3
Equal Importance	1
For compromise between the above values	2, 4, 6, and 8

In the next sections, we will illustrate our approach by analyzing four common architectural patterns (Pipes/Filters, Layered, MVC and Broker) for their accommodating to both Security and Performance quality attributes while building a credit-card system [11].

III. QUALITY – TACTICS INTERACTION

To evaluate the relations between qualities and tactics we will first establish qualitative relations between them using the five point scale that was proposed in [11]. That is if the impact of implementing the tactic is strongly in favor of satisfying the quality (make) then the interdependency relation is assigned

with double plus ("++") sign; and if the impact of implementing the tactic is moderately in favor of satisfying the quality (help) then the interdependency is assigned with a single plus ("+") sign. On the other hand, if the impact of implementing the tactic is strongly against satisfying the quality (break) then the interdependency is assigned with double minus ("--") sign; and if the impact of implementing the tactic is moderately against satisfying the quality (hurt) then the interdependency is assigned with a single minus ("-") sign. If implementing a tactic has no positive nor negative impact towards the quality in consideration, then the sign ("~") is being assigned.

In order to map these qualitative evaluations between qualities and tactics into quantitative ones, we incorporate the AHP technique. The steps towards applying the AHP technique on quality – tactics relations are:

We start exploring the Quality-Tactics interaction by the pairwise comparisons of qualities to be incorporated in the system under development. Table III represents the prioritization of the Security and Performance qualities in the credit-card system under development.

TABLE III. PAIRWISE COMPARISON MATRIX FOR SECURITY / PERFORMANCE IN A CREDIT-CARD SYSTEM.

	Security	**Performance**	**Weights**
Security	1	5	0.83
Performance	0.2	1	0.17

Next, we complete the comparisons of the tactics with respect to each quality in consideration (Security and Performance in our example). The qualitative knowledge described in [11] is transformed into quantitative values during the pairwise comparisons at this step. For each pairwise comparison, the impact of the first tactic with respect to the quality in consideration is to be compared against the impact of the second tactic with respect to the quality in consideration according to the scheme we propose in Table IV. For example, the pairwise comparison for (T9: Increase Computation Efficiency, T1: Authenticate Users) with respect to Performance is mapped to numerical value 5. This is because Increasing the Computation Efficiency helps satisfying the Performance quality (+); while Authentication hurts the satisfaction of Performance (As it hurts the response time of the system) (-). The pair (+, -) is then mapped to value 5 according to the proposed scheme.

TABLE IV. MAPPING THE QUALITATIVE IMPACTS FROM NFRS FRAMEWORK INTO THE AHP MEASUREMENT SCALES

Impact of the 1st tactic on a quality	**Impact of the 2nd tactic on a quality**	**Pairwise comparison value**
++	++	1
++	+	3
++	~	5
++	-	7
++	--	9
+	+	1
+	~	3
+	-	5
+	--	7
~	~	1
~	-	3
~	--	5
-	-	1
-	--	3
--	--	1

Table XI (Appendix I) presents the pairwise comparisons based on our analysis of the tactics with respect to Security quality attribute in a credit-card system. Table XII (Appendix I) shows the pairwise comparisons of the tactics with respect to Performance quality attributes in a credit-card system.

Finally, the results of the two analyses from Steps 1 and 2 are synthesized to compute the relative value of the tactics with respect to the goal of achieving both Security and Performance together. This is accomplished in our example by multiplying the weight of each Security tactic computed in Table XI by the value of "0.83" which represents Security weight as calculated from Table III (see the last column of Table XI); and the weight of each Performance tactic computed in Table XII by the value of "0.17" representing the Performance weight as calculated from Table III (see the last column of Table XII). Then the corresponding new weights for each tactic from the two Tables XI and XII are summed up. We end up with the synthesized weights for the tactics representing their relative values towards satisfying both qualities in accordance with their priorities in the credit-card system under development (Table V).

TABLE V. SYNTHESIZED WEIGHTS FOR TACTICS

Tactic	**Synthesized Weight**	**Tactic**	**Synthesized Weight**
T1	0.118	T9	0.045
T2	0.118	T10	0.045
T3	0.07	T11	0.045
T4	0.07	T12	0.045
T5	0.118	T13	0.04
T6	0.066	T14	0.022
T7	0.066	T15	0.022
T8	0.064	T16	0.043

IV. TACTICS – ARCHITECTURAL PATTERNS INTERACTION

To study the relationship between tactics and patterns, we analyze a pattern to evaluate the support it provides to help implementing a tactic. In [16], the authors identified six types of changes that a pattern's structure or behavior might undergo when a tactic is implemented within the pattern and they define a scale to rate these changes. In this paper, we adopt a similar five point scale from [16] to identify the impact magnitude of incorporating tactics within architectural patterns, where "++" presents "Good Fit" category, "+" presents "Minor Changes" required for the incorporation, "-" presents "Significant changes" required for the incorporation, "- -" presents "Poor Fit" category and "~" presents that the tactic and the pattern are basically orthogonal. The impact is defined as a function of number of the pattern's components impacted as listed in Table VI.

Tables VII and VIII summarize our findings on the impact of implementing Security and Performance tactics (using the scale described above) on Pipes-Filters, Layered, MVC and Broker patterns. For example, The Layers pattern structures an application by decomposing it into groups of subtasks where each group of subtasks is at a particular level of abstraction [3]. Because of Layers architecture organization, implementing the "Authenticate users" tactic (T1) is relatively easier than in Pipes-Filters. Indeed this tactic can be added as an additional layer on top of the existing ones; changes to the existing architecture are limited to its top layer and the new layer is within the structure of the pattern [16]. This is an example of (Add, in the pattern); and thus "+" was assigned as a sign. On other hand, as Layered Pattern supports layers of access, this limits the exposure by not placing all the logic and data into one layer. The sign "+ +" was assigned for the difficulty of incorporating Limit Exposure (T4) in Layered architecture. Our detailed analysis of the impact of implementing Security and Performance tactics on these four patterns is described in detail in [17].

TABLE VI. IMPACT MAGNITUDE AS A FUNCTION OF NUMBER OF PARTICIPANTS IMPACTED [16].

Change Type	Number of Changes	Impact Range
Implemented in	1	++ or +
Replicates	3 or less	++ or +
	More than 3	+ or ~
Add, in the pattern	3 or less	++ or +
	More than 3	+ or ~
Add, out of the pattern	3 or less	~ or -
	More than 3	- or --
Modify	3 or less	++ or -
	More than 3	~ or --

TABLE VII. IMPACT OF SECURITY TACTICS ON ANALYZED PATTERNS

	Pipes / Filters	Layered	MVC	Broker
T1	--	+	--	++
T2	--	--	--	--
T3	+	+	+	--
T4	++	++	--	-
T5	-	-	-	~
T6	~	~	~	+
T7	--	--	--	--
T8	--	-	~	++

TABLE VIII. IMPACT OF PERFORMANCE TACTICS ON ANALYZED PATTERNS

	Pipes / Filters	Layered	MVC	Broker
T9	-	-	+	+
T10	-	-	+	--
T11	-	-	+	--
T12	+	+	+	+
T13	++	+	+	~
T14	--	-	+	~
T15	~	~	~	~
T16	-	-	+	+

To complete the pairwise comparisons of the architectural patterns with respect to each identified tactic, the qualitative evaluation of patterns will be transformed into an ordinal scale according to a scheme similar to the one we proposed in Table IV. Table IX shows the pairwise comparison for the four patterns (Pipes/Filters, Layered, MVC and Broker) with respect to T1 (Authentication). For example, Broker scored "++" while Pipes / Filters scored "- -"; thus the pairwise comparison was mapped to 9. Due to the space constraints in this paper, we will not show the other 15 tables with respect to the other tactics.

TABLE IX. PAIRWISE COMPARISON FOR ARCHITECTURAL PATTERNS WITH RESPECT TO AUTHENTICATION

	Pipes / Filters	Layered	MVC	Brok-er	Weight	0.118 × Weight
Pipes / Filters	1	0.14	1	0.11	0.05	0.006
Layered	7	1	7	0.33	0.37	0.043
MVC	1	0.14	1	0.11	0.05	0.006
Broker	9	3	9	1	0.53	0.062

V. SYNTHESIZING THE IMPACT OF ARCHITECTURAL PATTERNS ON QUALITY ATTRIBUTES

We will now compute the quantitative impact of architectural patterns on quality attributes combining the data from the quality-tactic (section III) and tactic-pattern (section IV) relations. To do so, the results of the two analyses from the previous sections are synthesized to compute the relative weight of each architectural pattern with respect to the goal of achieving both Security and Performance qualities in the credit-card system. This is accomplished in our example by multiplying the weight of each architectural pattern in Table IX by the corresponding tactic weight calculated from Table V (as it is shown in the last column of Table IX); then the corresponding weights for each architectural pattern from all the 16 tables (i.e, corresponding to the 16 tactics) are summed up. We end up with the synthesized weights for the architectural patterns representing their value towards accommodating both Security and Performance qualities while considering the qualities' priorities; derived tactics and the interdependencies between the tactics and the qualities (See Table X).

TABLE X. SYNTHESIZED WEIGHTS OF ARCHITECTURAL PATTERNS

Pattern	Synthesized Weight
Pipes / Filters	0.22
Layered	0.22
MVC	0.26
Broker	0.3

The results from Table X reveal that when both Security and Performance are to be considered as the key qualities of the system, while prioritizing security over performance and considering all their tactics as criteria, the Broker is the most accommodating pattern for the credit-card system (it scored the highest among the four patterns). Pipes/Filters and Layered patterns are the least accommodating patterns for both Security and Performance qualities while implementing all

their tactics in the credit-card system (they scored the lowest among the four patterns).

VI. RELATED WORK

The need to deal with conflicting situations has become a critical area in requirements engineering. Several authors have detected the need to handle conflict resolution in requirements engineering. For example, in [18] J.Karlsson, C Wohlin, and B.Regnell have evaluated six different methods for prioritization software requirements. In their work, the authors have found that AHP technique to be the most promising method. In an industrial follow-up study they used the AHP to further investigate its applicability. They found that the process is demanding but worth the effort because of its ability to provide reliable results, promote knowledge transfer and create consensus among project members.

Many patterns have been proposed and described in the literature (e.g. [1, 2, 3, 4, 5]). However, patterns consequences descriptions are incomplete, not searchable or cross-referenced and, mostly qualitative [9]. They are often classified according to the specific classes of systems they enable to construct. Common approaches to architectural design (e.g., [7, 15]) propose a process to guide the architect in selecting architectural tactics and patterns that satisfy the most important quality attributes of a software system. However the architectural decisions such as the selection of a pattern is mainly based on a qualitative evaluation and the architect expertise and knowledge. Our work is closely related to the ones in [6] and [16]. Our analysis is partially based on the scale introduced in [16] where the authors propose a framework that relates patterns to tactics and quality attributes. The proposed framework concentrates on the interaction between tactics and patterns and especially how tactic implementations affect patterns. Our approach may be seen as complementary to this framework as we consider both the impact of a tactic on a pattern and the impact of the tactic on other quality attributes.

In [6], Bode and Riebisch present a quantitative evaluation of a set of selected architectural patterns regarding their support for the evolvability quality attribute. They refine the evolvability attribute into sub-characteristics and relate them to some properties that support good design. The selected patterns are used in a case study and the resulting design is assessed to determine the impact of these patterns on these properties. Our approach differs from Bode and Riebisch's approach in the way the analysis of the relationship between patterns and quality requirements was carried out; they use a particular case study and an evaluation by experts while our approach is based on the analysis of the generic structures and behavior of tactics and patterns. Besides, we consider more than one quality attribute in our analysis.

VII. CONCLUSION

In this paper, we proposed a quantitative approach to selecting architectural patterns starting from a subset of quality requirements. Our approach incorporates the mathematical based trade-off technique: Analytical Hierarchy Process (AHP) to quantitatively deal with ambiguities, trade-offs, priorities and interdependencies among qualities, tactics and architectural patterns. We illustrate the approach using four common architectural patterns and assessing their support for both Security and Performance quality attributes. Though this is a preliminary quantitative investigation of the architectural patterns when considering more than one quality attribute, we believe that it's a key step towards a quality-driven architectural design process.

In the future, we plan to refine the approach in three ways. First we would like to refine our analysis and results by considering sub-characteristics of quality attributes and design concerns (e.g. analyzing availability and confidentiality as sub-characteristics of Security). Second, the numerical value that is assigned to a pattern regarding its support for a quality attribute depends on the selected subset of tactics to achieve the targeted attribute. In this paper we derived this value by considering all the tactics related to an attribute. However, in the future we would like to give the opportunity to an architect to tune these values by considering or discarding alternative tactics. This will help to alleviate impacts of the pattern whose choice was driven by a core of quality attributes on the other attributes. Third, we will extend the proposed approach in the design of systems involving multiple architectural styles in which styles could not necessarily be evaluated in isolation from each other.

REFERENCES

[1] D. Garlan, M. Shaw, "An Introduction to Software Architecture", Technical Report, CMU, Pittsburgh, PA, USA, 1994.

[2] P. Avgeriou and U. Zdun, "Architectural Patterns Revisited– a Pattern Language", In Proceedings of EuroPLoP 2005, pp. 1- 39, 2005.

[3] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, "Pattern-Oriented Software Architecture: A System of Patterns", John Wiley & Sons, 1996.

[4] M. Shaw and D. Garlan, "Software Architecture: perspectives on an emerging discipline", Prentice Hall, 1996.

[5] E. Gamma, R. Helm, R. Johnson, and J.M. Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software systems", Addison-Wesley, 1994.

[6] S. Bode and M. Riebisch, "Impact Evaluation for Quality-Oriented Architectural Decisions Regarding Evolvability", The 4th European conference on Software architecture, pp.182-197, 2010.

[7] L. Bass, P. Clements, and R. Kazman, "Software architecture in practice", Addison-Wesley, 2003.

[8] F. Bachmann, L. Bass, and R. Nord, "Modifiability Tactics", Technical Report, SEI, CMU/SEI 2007-TR-002, September 2007.

[9] N. Harrison, P. Avgeriou, and U. Zdun, "On the Impact of Fault Tolerance Tactics on Architecture Patterns", In proceedings of 2nd International Workshop SERENE, 2010.

[10] J. Jansen, J. van der Ven , P. Avgeriou, and D. K. Hammer, "Tool Support for using Architectural Decisions", In proceedings of the 6th Working IEEE/IFIP WICSA'07, 2007.

[11] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, "Non-functional Requirements in Software Engineering", Kluwer Academic Publishing, 2000.

[12] T. L. Saaty, "The analytic hierarchy process", New York: McGraw-Hill, 1980.

[13] T. L. Saaty, "Decision making for leaders", Belmont, California: LifeTime Leaning Publications, 1985.

[14] International Standard ISO/IEC 9126-1. Software engineering – Product quality – Part 1: Quality model. ISO/IEC 9126-1:2001, 200.

[15] W. G. Wood, "A Practical Example of Applying Attribute-Driven Design (ADD)", Technical report CMU/SEI-2007-TR-005, 2007.

[16] N.B. Harrison and P. Avgeriou, P., "How do architecture patterns and tactics interact? A model and annotation", Journal of Systems and Software, vol. 83, Issue 10, pp. 1735-1758, 2010.

[17] M. Kassab, G. El Boussaidi, H. Mili, "A quantitative evaluation of the impact of architectural patterns on quality requirements", in the 9th ACIS SERA Conference, 2011.

[18] J. Karlsson, C. Wohlin, and B. Regnell, "An Evaluation of Methods for Prioritizing Software Requirements", Information and Software Technology, Vol. 39, No. 14-15, pp. 939-947, 1997-98.

Appendix I

TABLE XI. PAIRWISE COMPARISONS FOR SECURITY AND PERFORMANCE TACTICS WITH RESPECT TO SECURITY

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	Weight	0.83 × Weight
T1	1	1	3	3	1	3	3	3	5	5	5	5	5	7	7	5	0.14	0.11
T2	1	1	3	3	1	3	3	3	5	5	5	5	5	7	7	5	0.14	0.11
T3	0.33	0.33	1	1	0.33	1	1	1	3	3	3	3	3	5	5	3	0.08	0.06
T4	0.33	0.33	1	1	0.33	1	1	1	3	3	3	3	3	5	5	3	0.08	0.06
T5	1	1	3	3	1	3	3	3	5	5	5	5	5	7	7	5	0.14	0.11
T6	0.33	0.33	1	1	0.33	1	1	1	3	3	3	3	3	5	5	3	0.08	0.06
T7	0.33	0.33	1	1	0.33	1	1	1	3	3	3	3	3	5	5	3	0.08	0.06
T8	0.33	0.33	1	1	0.33	1	1	1	3	3	3	3	3	5	5	3	0.08	0.06
T9	0.2	0.2	0.33	0.33	0.2	0.33	0.33	0.33	1	1	1	1	1	3	3	1	0.03	0.026
T10	0.2	0.2	0.33	0.33	0.2	0.33	0.33	0.33	1	1	1	1	1	3	3	1	0.03	0.026
T11	0.2	0.2	0.33	0.33	0.2	0.33	0.33	0.33	1	1	1	1	1	3	3	1	0.03	0.026
T12	0.2	0.2	0.33	0.33	0.2	0.33	0.33	0.33	1	1	1	1	1	3	3	1	0.03	0.026
T13	0.2	0.2	0.33	0.33	0.2	0.33	0.33	0.33	0.33	0.33	0.33	0.33	1	3	3	1	0.025	0.021
T14	0.14	0.14	0.2	0.2	0.14	0.2	0.2	0.2	0.33	0.33	0.33	0.33	0.33	1	1	0.33	0.01	0.0099
T15	0.14	0.14	0.2	0.2	0.14	0.2	0.2	0.2	0.33	0.33	0.33	0.33	0.33	1	1	0.33	0.01	0.0099
T16	0.2	0.2	0.33	0.33	0.2	0.33	0.33	0.33	1	1	1	1	1	3	3	1	0.03	0.026

TABLE XII. PAIRWISE COMPARISONS FOR SECURITY AND PERFORMANCE TACTICS WITH RESPECT TO PERFORMANCE

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	Weight	0.17 × Weight
T1	1	1	0.33	0.33	1	1	1	3	0.14	0.14	0.14	0.14	0.14	0.33	0.33	0.2	0.02	0.003
T2	1	1	0.33	0.33	1	1	1	3	0.14	0.14	0.14	0.14	0.14	0.33	0.33	0.2	0.02	0.003
T3	3	3	1	1	3	3	3	3	0.33	0.33	0.33	0.33	0.33	0.2	0.2	0.2	0.04	0.008
T4	3	3	1	1	3	3	3	3	0.33	0.33	0.33	0.33	0.33	0.2	0.2	0.2	0.04	0.008
T5	1	1	0.33	0.33	1	1	1	3	0.14	0.14	0.14	0.14	0.14	0.33	0.33	0.2	0.02	0.003
T6	1	1	0.33	0.33	1	1	1	3	0.14	0.14	0.14	0.14	0.14	0.33	0.33	0.2	0.02	0.003
T7	1	1	0.33	0.33	1	1	1	3	0.14	0.14	0.14	0.14	0.14	0.33	0.33	0.2	0.02	0.003
T8	0.33	0.33	0.33	0.33	0.33	0.33	0.33	1	0.14	0.14	0.14	0.14	0.14	0.33	0.33	0.2	0.01	0.002
T9	7	7	3	3	7	7	7	7	1	1	1	1	1	1	1	1	0.11	0.019
T10	7	7	3	3	7	7	7	7	1	1	1	1	1	1	1	1	0.11	0.019
T11	7	7	3	3	7	7	7	7	1	1	1	1	1	1	1	1	0.11	0.019
T12	7	7	3	3	7	7	7	7	1	1	1	1	1	1	1	1	0.11	0.019
T13	7	7	3	3	7	7	7	7	1	1	1	1	1	1	1	1	0.11	0.019
T14	3	3	5	5	3	3	3	3	1	1	1	1	1	1	1	1	0.07	0.012
T15	3	3	5	5	3	3	3	3	1	1	1	1	1	1	1	1	0.07	0.012
T16	5	5	5	5	5	5	5	5	1	1	1	1	1	1	1	1	0.10	0.016

Metrics-based Detection of Similar Software

Paloma Oliveira[1,2]
[1]Department of Computer, IFMG
Formiga-MG, Brazil
paloma.oliveira@ifmg.edu.br

Hudson Borges[2], Marco Tulio Valente[2]
[2]Department of Computer Science, UFMG
Belo Horizonte – MG, Brazil
mtov@dcc.ufmg.br

Heitor Augustus Xavier Costa[3]
[3]Department of Computer, UFLA
Lavras – MG, Brazil
heitor@dcc.ufla.br

***Abstract* - This paper presents a quantitative approach to identify similarity among object-oriented systems. This approach has three major contributions: a) a mechanism to derive thresholds for a specific metric, considering different class profiles; b) a mechanism to obtain a subset of similar systems from a portfolio of systems according to one software metric and using well-known clustering techniques; and c) a mechanism to obtain subsets of similar systems according to a set of software metrics and using concepts of graph theory. In this paper, we also present a tool that supports our approach, called SQComp. Using SQComp, we evaluated our approach in a corpus of 103 open-source systems, comprising more than 16 MLOC. As a result, we were able to found several groups of systems with strong indications of similarity.**

Keywords - internal quality; software metrics; thresholds.

I. INTRODUCTION

Object-oriented programming is the dominant software development paradigm. However, after more than four decades of the inception of the main OO abstractions, there is a surprisingly modest quantitative knowledge on the internal structure of object-oriented software [2]. Particularly, we still lack thresholds to metrics when evaluating the internal quality of OO systems. In other works, many metrics have been proposed to quantify internal OO properties, such as, size, coupling, cohesion, information hiding, and complexity [3]. However, we still lack a solid body of knowledge to effectively support the use such metrics to assess software quality.

Although we have many OO methodologies, design principles, and heuristics, we typically cannot answer some simple questions, such as, "How many lines of code (LOC) does the typical class have?" or "Is there such a thing as a 'typical class'?". What we would really like to know about software is "Is it good?", i. e., does it have quality attributes, such as, high modifiability, high reusability, high testability, or low maintenance costs [2].

There are some recent studies claiming that many software metrics follow heavy-tailed distributions [2, 10]. If this is true, it would have important implications on the types of empirical studies that are possible. One issue is the fact that heavy-tailed distributions do not have finite mean and variance. If this is the case, the central limit theorem cannot be applied; so, the sample mean and variance cannot be used as estimates of the population mean and variance. Therefore, making any conclusions on sample means and variances without fully understanding the distribution is questionable at best.

Software can be analyzed according to several characteristics, such as, static structure, dynamic execution, available functionality, and lifecycle evolution. This paper focuses on similarity of static structure, which includes many different measures of source code. More specifically, we present a quantitative approach based on a set of 19 software metrics to assess the internal similarity of object-oriented software. This set of metrics was selected because they are related to relevant software quality factors, such as, cohesion, coupling, and information hiding. This study involved 103 software from the Qualitas Corpus [10].

The paper is organized as follows. Section II describes the background work and concepts. Section III presents our approach. Section IV describes a case study carried out to evaluate our approach. Section V discusses related work. Concluding remarks are presented in Section VI.

II. BACKGROUND

The practical approach to verify similarity among object-oriented systems presented in this paper uses concepts like heavy-tailed distribution, clustering techniques, and graph cliques. These concepts are briefly described in this section.

A. Heavy-tailed distribution

The analysis of the interaction among modules of a system has been target of many researchers. Some studies indicate that systems, in general, are in conformity with heavy-tailed distribution. A characteristic of this distribution type is the frequency of high and low values being low and high, respectively, for the random variable. In such distribution, the mean value is not representative, and so, there is no typical value for random variables [5, 6, 9, 11].

There are several distributions with features of heavy-tailed. In this paper, the EasyFit[1] tool is used to fit the data. For each metric, we considered the values extracted for the classes of a given system as heavy-tailed when the "best-fit" distribution returned by EasyFit is Power Law, Weibull, LogNormal, Cauchy, Pareto or Exponential [11].

B. Clustering

Clustering is a technique for grouping of similar elements of a sample to form mutually exclusive clusters [12]. There are different methods of grouping, but the best known is the Kmeans algorithm. This method carries out cluster analysis though the partition of `n` observations (elements) into `k` clusters. These observations are distributed in clusters according to closeness to the mean value of each cluster, i.e., clusters are formed iteratively by rearranging the elements to minimize the cluster center (average).

[1] http://www.mathwave.com/products/easyfit.html.

C. Clique

A clique in an undirected graph, G = (V, E), is a complete sub graph of G, G' = (V', E'), such that V' ⊆ V, E' ⊆ E, and exists an edge between any two vertices in V' [4]. A maximal clique cannot be extended by including an additional vertex.

III. IDENTIFYING SIMILARITY OF OBJECT-ORIENTED SYSTEMS

The approach presented in this paper aims to contribute for establishing internal quality for object-oriented systems. Basically, thresholds of metrics for class profiles of a system are designed according to best practices. Such thresholds can be used as indicators of internal quality. For example, suppose two systems, S_1 and S_2. Suppose that S_1 was developed in accordance with the best principles/concepts of Software Engineering. Moreover, it was developed by a team of skilled developers, with high experience. Thus, the probability that S_1 has a high degree of internal quality is high. Suppose also that using the approach proposed in this paper, we identify that S_2 and S_1 are similar. Thereby, we claim that this similarity is a strong indication that S_2 also has high levels of internal quality.

Our approach has three principal mechanisms: A) A mechanism to obtain thresholds to specify software metrics, according to different profiles of classes; B) A mechanism to obtain set of similar systems from a system repository using clustering techniques and accordingly to thresholds of a specific metric; and C) A mechanism to obtain sets of similar systems, according to the thresholds of a set of metrics.

A. Obtaining Thresholds for Class Profiles

Defining threshold for software metrics at class level is not a trivial task [1, 5]. For example, to obtain a threshold to the LOC metric is complex. The main reason is that LOC follows a heavy-tailed distribution. In other words, systems have many classes with few LOC (e.g., less than a hundred lines) and few classes with many LOC (e.g., in some cases, thousands of lines). Therefore, the mean of LOC is unrepresentative, which makes it more challenging to analyze these systems.

To overcome this difficulty, we used clustering – specifically the Kmeans algorithm – to partition the classes into groups (G_1,...,G_n), automatically. The groups denote the classes' location in different positions in the curve of a heavy-tailed distribution. For example, suppose three groups, G_1, G_2, and G_3, and a metric, M_1. The classes in G_1, G_2, and G_3 have high, medium, and low values for M_1, respectively. These values represent the Euclidean distance between the value of M_1 for a class in a given group and the threshold of this group. The threshold for M_1 for each group is obtained by considering the mean value of M_1.

B. Obtaining a Set of Similar Systems for One Metric

This mechanism is used to determine sets (clusters) of systems with indications of similarity for a specific metric. Suppose a set of systems, S, and a metric, M_1. The thresholds for class profiles of the systems in S are calculated for M_1 and the groups G_1,..., G_n are generated (using the procedure described in the previous section). For example, suppose two systems, S_1 and S_2, one metric, M_1, and one profile, P. For example, both systems can have three profiles of classes ($P=3$): i) G_1: classes with high value for M_1; ii) G_2: classes with medium value for M_1; and iii) G_3: classes with low value for M_1.

In this mechanism we use the Kmeans algorithm again to obtain groups of similar systems according to the profile of their classes. In other words, classes of the systems in S are clustering according to values of M_1 in the groups generated. The determination of the number of clusters is not trivial when the Kmeans algorithm is used. Basically, we should run the algorithm several times to different amounts of clusters until the internal cohesion of the clusters exceeds 90%. Clusters of similar systems in relation to the values of M_1 are the output of this algorithm.

C. Obtaining a set of similar systems for a set of metrics

This mechanism is an extension of the Mechanism B. It is used to determine sets (clusters) of systems with similarity indications regarding a set of metrics. It combines two or more thresholds of distinct metrics as similarity criterion.

Suppose a set of systems, S, and a set of metrics, M. The Mechanism B should be used for each metric in M. Systems with similarity indications for metrics in M are represented by a graph. The nodes are the systems and the edges represent the minimum amount of metrics with similarity indications between two systems. For example, suppose two systems, S_i and S_j. Suppose also that the number of metrics for detecting similarity indications between S_i and S_j is $Q_{m(i,j)}$, and the minimum number of metrics that indicates that two systems have similarity is Q_{min}. There is an edge between S_i and S_j, if $Q_{m(i,j)} \geq Q_{min}$. Systems with similarity indications for the same set of metrics are represented by cliques. The resulting graph can have several cliques, which characterizes systems with similarity.

IV. CASE STUDY: *QUALITAS CORPUS*

We use the Qualitas Corpus (version 20101126r), which is a repository with 103 open-source Java-based systems, specially created for empirical research in software engineering [10]. These systems have around 115K classes and more than 16 MLOC.

We used 19 metrics related to classes for measuring important factors to software quality, such as, size, complexity, encapsulation, and cohesion. These metrics are: Number of Lines of Code (LoC); ii) Number of Attributes (NoA); iii) Number of Public Attributes (PubA); iv) Number of Private Attributes (PriA); v) Number of Inherited Attributes (IA); vi) Number of Methods (NoM); vii) Number of Public Methods (PubM); viii) Number of Private Methods (PriM); ix) Number of Inherited Methods (IM); x) Fan-Out; xi) Fan-In; xii) Coupling between Object Class (CBO); xiii) Number of Children (NoC); xiv) Total Number of Children (TNoC); xv) Response for a Class (RFC); xvi) Weighted Methods per Class (WMC); xvii) Number of Methods Overriden (NoMO); xviii) Lack of Cohesion in Methods (LCoM); and xix) Hierarchy Nesting Level (HNL).

A. *SQComp - A Computational Support for Comparative Evaluation of Internal Software Quality*

A tool was developed for providing a graphical user interface and to automate the proposed approach. This tool, called SQComp (Comparative Evaluation of Internal Software Quality), uses the Moose platform to compute the metrics values for each class of each system [8]. To cluster and to find cliques SQComp we used the R Platform[2].

B. *Methodology and Results:*

Mechanism A: Obtaining Thresholds for class profiles

Five groups, G_1, G_2, G_3, G_4, and G_5, were defined to represent threshold for class profiles. These groups correspond to classes with small values for metrics, classes with values for metrics ranging from small to medium, classes with medium values for metrics, classes with values for metrics ranging from medium to high, and classes with high values for metrics, respectively.

To illustrate, the thresholds of LoC regarding the groups of classes in the FindBugs system are presented in Table I. The calculated thresholds, the number of classes, and percentage of classes in each group are shown on the second, third, and fourth rows, respectively. Therefore, there is a better understanding of the class profiles of the FindBugs system related to number of lines of code. Approximately, 73.84% of classes are small (26.2 lines of code); on the other hand, 0.25% of classes are large (23331.7 lines of code). The thresholds of LoC for five other systems are presented in Table II.

TABLE I. THRESHOLDS FOR CLASS PROFILES IN THE FINDBUGS SYSTEM

	G_1	G_2	G_3	G_4	G_5
Number of LoC	26.2	135.2	359.3	787.5	2331.7
Number of Classes	878	226	63	19	3
Percentage of Classes (%)	73.84	19.00	5.30	1.60	0.25

TABLE II. THRESHOLDS FOR LOC IN TEN SYSTEMS

Systems	G_1	G_2	G_3	G_4	G_5
Ant	19.64	92.18	217.49	448.91	916.92
Aoi	48.50	213.37	523.74	1130.36	2635.20
Argouml	26.04	163.69	478.88	1813.00	9122.00
Aspectj	56.38	382.73	1078.24	2719.89	6332.00
Cayenne	17.11	82.16	197.59	400.77	973.33

Mechanism B: *Obtaining a set of similar systems for one metric*

The number of clusters of similar systems is obtained by the kmeans(X, k) function of the R system, where X is equal to 5 (groups) and k is number of clusters. This function was run several times for different number of clusters until we obtained, at least, 90% of internal cohesion of the clusters (as suggested in Section IV-B). For example, considering LoC, the value for k more appropriate was 5. This means that the systems ere classified into five groups considering similarity indications for LoC.

Most systems (33 systems ≈ 32.04%) were classified in the cluster #3. These systems present the following values for LoC: G_1 = 23.77; G_2 = 116.87, G_3 = 286.43; G_4 = 619.17; and G_5 = 1,525.36. In the other hand, three systems were classified in the cluster #2 and have following values for LOC: G_1 = 39.17; G_2 = 226.43; G_3 = 625.47; G_4 = 1,887.83; and G_5 = 6,147.33.

Mechanism C: *Obtaining a set of similar systems for a set of metrics*

In our approach, the Mechanism B relies on a single metric and the Mechanism C uses a set of metrics. Therefore, we used the Mechanism B for each metric that SQComp provides (19 metrics). Besides, we decided to use k = 5 (kmeans(X, k) function), because the internal cohesion was approximately 90%. Finally, we obtained a 103 x 103 matrix, where each cell has the number of times that two systems were classified in the same cluster (similar metric). A subset of this matrix with only 10 systems is shown in Table III.

TABLE III. NUMBER OF SIMILAR METRICS AMONG SYSTEMS

	Ant	Aoi	ArgoUML	AspectJ	Cayene
Ant	19	1	3	1	9
Aoi	1	19	9	10	2
Argouml	3	9	19	9	1
AspectJ	1	10	9	19	0
Cayenne	9	2	1	0	19

As it can be seen, the intersection cell between the Ant and Cayenne has the value nine, meaning that these systems have similarity indications for the same set of metrics (nine metrics). Cells in the main diagonal have the value 19, because each system is compared with itself. We decide to set Q_{min} = 9, because it represents half the number of metrics. An adjacency matrix was then used to represent pair of systems with $Q_{min} \geq 9$, where 1 and 0 represent the presence and absence of an edge, respectively. The graph corresponding to this adjacency matrix is shown in Figure . In this figure, there is one clique with three vertices (AspectJ, AOI, and ArgoUML) and one clique with two vertices (ant and cayenne).

Evaluating 103 systems. We conducted some experiments by changing Q_{min}. In one of these experiments, we used Q_{min} = 12 and we find out a maximum clique with three vertices (Quilt; JUnit; JFinDateMath) and similar metrics were LoC, IA, NoM, PubM, FAN-OUT, FAN-IN, CBO, NoMO, TNoC, WMC, RFC, and LCoM. Besides this experiment, two experiments are shown in Table IV. We can noticed that smaller the set of similar metrics (Q_{min}), the higher is the number of similar systems.

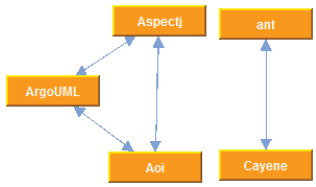


Figure 1 - Adjacency Matrix in Table III

Discussion: Software quality can be measured by external and internal factors [7]. Our approach evaluates internal software quality by considering a set of metrics that includes several features of object-oriented systems. Table IV shows the systems with indications of high similarity in their internal structure, considering coupling, cohesion, and complexity. However, the approach is not able to explain this similarity,

[2] http://www.R-project.org/

which may be attributed to development processes, programming patterns, same programmers' team, or just for being a coincidence.

TABLE IV. THREE EXPERIMENTS: MAXIMUM CLIQUE, SYSTEMS, AND SIMILAR METRICS

Q_{mim}	Systems (Vertices)	Similar Metrics (Edges)
12	Quilt; JUnit; JFinDateMath	LoC; IA; NoM; PubM; FAN-OUT; FAN-IN; CBO; NoMO; TNoC; WMC; RFC; LCoM
9	Quilt; JUnit; FitJava; JFinDateMath	LoC; NoM; PubM; CBO; NoMO; TNoC; WMC; RFC; LCoM
7	JUnit; CheckStyle; JGraph; Quilt; JFinDateMath	NoM; PubM; NoMO; FAN-OUT; CBO; RFC; LCoM

V. RELATED WORKS

In this section, we discuss work related with our approach. Baxter et. al analyzed 17 metrics in 56 Java systems for verifying their internal structure [2]. The authors reported that most metrics follow power-laws. Louridas et al. analyzed coupling metrics using 11 systems developed in multiple languages (C, Perl, Ruby, and Java) [6]. The authors concluded that most metrics are in conformity with heavy-tailed distribution, independently of programming language. Studies conducted by Taube-Schock et al. confirms such results, but for coupling metrics [9].

In the context of thresholds, a study of Alves et. al [1] used 100 systems to obtain thresholds with the aim to classify them. This study analyzed the quantile function for a set of systems and attributed weight by LOC for calculating the threshold for each metric. Ferreira et. al [5] has proposed thresholds for 6 metrics, considering 40 systems. As result, systems were classified into 3 categories: Good, Regular, and Poor.

We can observe that there is not well-established thresholds for software metrics that reflect real practices and projects of development. In general, studies tend to recommend absolute thresholds, such as, "the class is good if it has at most 20 methods" [5]. However, several studies indicate that software metrics follow a heavy-tail distribution [2,5,6,9]. In other words, software practice seems to reveal that is inevitable to have software components with very high values of metrics (the tail of the distribution). In this context, our approach aims to provide thresholds respecting the heavy-tailed behavior common in software metrics distributions.

VI. CONCLUSION

A approach to verify similarity among object-oriented systems was presented in this paper. The approach includes three mechanisms: i) obtaining thresholds for class profiles; ii) obtaining a set of similar systems for a single metric; and iii) obtaining a set of similar systems for a set of metrics.

The proposed approach helps to check the existence of classes of distinct profiles in the systems analyzed in terms of size, coupling, cohesion etc. This reinforces the result of some authors that software metrics follows a heavy-tailed distribution. The use of clustering is an interesting alternative, because the mean of the elements in a cluster is more representative and can be used as threshold for different class profiles in the same system.

As future work, we suggest to use the approach in other systems repositories and to discover why systems have a specific similarity. We also aim to investigate the existence of causality between the proposed thresholds and external software quality metrics, such as number of bugs [13] and number of warnings raised by bug finding tools [14,15].

Acknowledgments Our research is supported by CAPES, FAPEMIG, and CNPq.

REFERENCES

[1] Alves, T.; Ypma, C.; Visser, J. (2010). Deriving metric thresholds from benchmark data. In 26th *Int. Con. on Software Maintenance*, pp. 1–10.

[2] Baxter, G.; Frean, M.; Noble, J.; Rickerby, M.; Smith, H.; Visser, M.; Melton, H.; Tempero, E. (2006). Understanding the shape of Java software. *In* 21th *Int. Conf. on Object Oriented Programming, Systems, Languages and Applications*. pp. 397-412.

[3] Chidamber, S..; Kemerer, C. (1994). A metrics suite for object oriented design. *IEEE Trans. on Software Engineering.* 20(6). pp.476–493.

[4] Cormen, T.; Leiserson, C.; Rivest, R.; Cliford C. (2009). Introduction to Algorithms. 3rd edition. *MIT Press*.

[5] Ferreira, K.; Bigonha, M.; Bigonha, R.; Mendes, L.; Almeida, H. (2011). Identifying Thresholds for Object-Oriented Software Metrics. *The Journal of Systems and Software*. 85(2). pp.244-257.

[6] Louridas, P.; Spinellis, D.; Vlachos, V. (2008) Power Laws in Software. *ACM Trans. on Software Engineering and Methodology.* 18(1). pp.1-26.

[7] Meyer, B. (2000). Object-oriented Software Construction. Prentice-Hall.

[8] Oscar, N.; Stéphane, D.; Tudor, G. (2005). The Story of Moose: An Agile Reengineering Environment. *In: European Sof. Engineering Conference*. pp. 1-10.

[9] Taube-Schock, C.; Walker, R.; Witten, I.(2011). Can We Avoid High Coupling". *In. 25th European Conf. on Object-Oriented Programming*. pp. 204-228

[10] Tempero, E.; Anslow, C.; Dietrich, J.; Han, T.; Li, J.; Lumpe, M.; Melton, H.; Noble, J. (2010). The Qualitas Corpus: A Curated Collection of Java Code for Empirical Studies. *In: Asia-Pacific Software Engineering Conference.* pp. 336-345.

[11] Sergey Foss, Dmitry Korshunov, and Stan Zachary. An Introduction to Heavy-Tailed and Subexponential Distributions. Springer-Verlag, 2011.

[12] Tufféry, S. (2011) Association Analysis, in Data Mining and Statistics for Decision Making. *John Wiley & Sons*.

[13] Cesar Couto; Christofer Silva; Marco Tulio Valente; Roberto Bigonha; Nicolas Anquetil. (2012) Uncovering Causal Relationships between Software Metrics and Bugs. 16th European Conf. on Software Maintenance and Reengineering, pp. 223-232.

[14] Joao Eduardo Araujo; Silvio Souza; Marco Tulio Valente. (2011) Study on the Relevance of the Warnings Reported by Java Bug Finding Tools. IET Software, v. 5, n. 4, pp. 366-374.

[15] Cesar Couto; Joao Eduardo Araujo; Christofer Silva; Marco Tulio Valente. (2013) Static Correspondence and Correlation between Field Defects and Warnings Reported by a Bug Finding Tool. Software Quality Journal, v. 21, n. 2, pp. 241-257, Springer.

A Checklist for Evaluation of Reference Architectures of Embedded Systems

José Filipe Marreiros Santos*, Milena Guessi*, Matthias Galster†, Daniel Feitosa* and Elisa Yumi Nakagawa*
*Department of Computer Systems
University of São Paulo - USP
PO Box 668, 13560-970, São Carlos, SP, Brazil
Email: joesantos@usp.br, {milena, fdaniel, elisa}@icmc.usp.br

† University of Canterbury
Department of Computer Science and Software Engineering
Private Bag 4800, Christchurch 8140, New Zealand
Email: mgalster@ieee.org

***Abstract*—Embedded systems are computers designed to perform specialized tasks. Examples of embedded systems include printers, consoles and televisions. The software that controls embedded systems usually present critical requirements, since, many times, their failure may result in human harm or environmental damage. Therefore, the design of such software requires a quality driven approach. In software engineering, reference architectures are reusable software engineering artifacts introduced to facilitate the design of software architectures of a given domain. The adoption of reference architectures in embedded systems design offers advantages that could help improve their quality. To assure that the reference architecture presents all required information and address all concerns, it is important to have means of evaluating it, but available evaluation methods for reference architecture require adaptation and may have limitations. In this context, this work introduces a checklist for evaluation of reference architectures of embedded systems. We elaborate on a web based tool that could support the checklist application. To evaluate this checklist, we considered the opinion of experts in software architecture and reference architecture. Also, we successfully applied the checklist in an academic reference architecture project. We expect that this work contributes to the evaluation of reference architectures of embedded systems. Finally, we intend that this work could open interesting, new research perspectives in this direction.**

I. Introduction

Embedded systems can be understood as devices that include programmable computers but are not intended to be general-purpose computers themselves [1]. Usually, their development requires great effort due to stringent requirements that may exist. This includes not only technical requirements, such as adaptability, maintainability, safety, security, and dependability, but also business requirements, like smaller time-to-market and lower cost [1].

In another context, reference architectures refer to software architectures that encompass the knowledge about how to design concrete architectures of systems in a given application domain. Therefore, they must address all the concerns and the software elements that support development of systems [3] with a proper description and organization.

Given the diversity of requirements and the safety criticality of embedded systems, their design usually demands a quality driven approach. This can be achieved through the use of reference architectures. In fact, their adoption for the design of embedded systems can bring many benefits such as integrability improvement, design decisions reuse and basic software standardization [2].

One example of reference architecture of embedded systems is AUTOSAR (AUTomotive Open System ARchitecture),[1] a reference architecture for systems embedded in cars. AUTOSAR was created by some of the main manufacturers of car parts (e.g. BMW, Bosch, Toyota, Continental, Daimler, Ford, GM, Peugeot, Volkswagen) aiming to standardize interfaces and stimulate innovation.

Considering that reference architectures will the basis for the construction of a set of systems, their evaluation requires special attention. According to Barcelos and Travassos, evaluation methods for software architectures may present issues, such as scope limitations, dependence on the evaluator's knowledge, dependence on particular representation techniques or high costs. All these, can be overcame by the use of checklists [7]. In addition, although there are many evaluation methods for software architectures, they cannot be used without adaptations for reference architectures [5]. For instance, Graaf et al. tailored SAAM to evaluate an industry reference architecture [6].

The objective of this paper is to present a checklist for evaluation of reference architectures of embedded systems.

[1] Available at http://www.autosar.org/ (Accessed on 03/10/2013)

We believe that the checklist is a simple and flexible method, which can be used alone or as a pre-evaluation that may support other methods to discover mistakes at an early stage of the reference architecture life cycle. We also propose a web based tool to support the checklist application. Finally, we identify future and important research lines based on our results.

This paper is organized as follows. In Section II we present the checklist and elaborate on a tool support for its application. In Section III we present the results of a qualitative evaluation of the checklist which consulted experts in software architecture and reference architecture. Finally, in Section V we summarize our contributions and discuss perspectives for future work.

II. Checklist for Reference architectures of Embedded Systems

The checklist construction was based on literature available on embedded systems, reference architectures, and software architecture. At first, we adapted the checklist for evaluation of software architectures presented in the work of Clements et al. [8] for reference architectures. Then, we extended the checklist with questions about quality attributes presented in the work of Rozanski et al. [9]. These quality attributes were determined by a systematic review presented by Guessi et al. [11]. Finally, the checklist was refined to cover the topics treated by Nakagawa et al. in their RAModel [10], which stated the information that should be presented in a reference architecture.

We kept the stakeholder considerations originally made by Clements et al. [8], which divided the questions among seven respondents: architects, domain experts, developers, analysts, software manager, integrators, testers and quality assurance stakeholders. Architects are the software engineers that are responsible for designing the reference architecture. The domain experts are, in this case, the ones with knowledge on embedded systems. Developers are the professionals responsible for implementing derived application, that is, applications which the architecture is derived from the reference architecture. Analysts are in charge of the documentation i.e. requirements engineering and UML diagrams. Software managers make the strategic decisions of the project, such as resources and time management. Integrators are responsible for the system consistency and adaptation. Testers are in charge of finding defects on the documentations and the developed application. Finally, quality assurance stakeholders verify the project conformance with the quality requirements. In addition, nine concerns about embedded systems are considered: availability, interoperability, maintainability, performance, security, reliability, scalability, protection, usability.

Regarding standardization, ISO 42010 specifies how architectural descriptions are expressed and organized. It also specifies viewpoints, frameworks and description languages for use in architectural descriptions [4]. Conformance with this standard is one of the topics covered by the checklist originally presented by Clements et al [8].

The checklist proposed in this paper is structured into four stages, namely: (i) General Information, (ii) Raising Discussions, (iii) General Analysis Conclusion and (iv) Embedded Systems Specifics. These stages are divided into one or more sets of questions.

Table I presents the topics by stage and Table II presents a subset of questions from the checklist.

The first stage is divided into two sets of questions. The first set includes questions about general information, e.g. overview data, viewpoints, views and models, while the second set contains questions about the reference architecture construction e.g. if quality attributes were considered or if the stakeholders are defined. The stakeholders to respond to this stage are solely the architects. This stage has 25 questions, nine in the first set and 16, in the second.

The second stage has only one set. It intends to raise discussions and improve rationale understanding. It has 52 questions, seven for all the stakeholders, 12 for the architects, six to be answered by the domain experts, six for the project manager, four for the developers, three for integrators, three for the testers, nine for the quality assurance managers and finally, two questions targeting analysts.

The third stage has only one set about general analysis conclusion, e.g. if the reference architecture fits its purpose. It has three questions. These questions target all stakeholders.

Finally, the fourth stage has two sets. There are 11 questions in the first set about embedded hardware and three about embedded software. Both sets for domain experts.

The checklist was built to have a broad coverage. Therefore, some questions might not be suitable for a specific project. For example, some questions may be directed for stakeholders that are unknown. In these cases, the questions not relevant in a particular project can be simply dismissed.

We believe that the checklist application may produce better results during the elaboration of the reference architecture. At this point, discovered mistakes are easier to identify and fix.

A. Supporting Tool

Currently, FERA-ES Web (Framework for evaluation of reference architectures of embedded systems), an open source web application designed to support the checklist is under development. The goal of this tool is to provide an online collaborative environment to control the evaluation and help the results management.

The evaluation team will be able to create new projects and use the checklist, either as we proposed, or customized.

TABLE I
TOPICS AND NUMBER OF QUESTIONS IN EACH STAGE

Stage	Set	Questions	Subjects covered
1	1	9	overview information, viewpoints, views, models, stakeholders and concerns
	2	16	legal regulation, ISO 42010 conformance, design and development issues, domain specific issues, conformance with other artifacts, quality attributes
2	1	52	specific questions to stakeholders
3	1	3	conclusion of the general analysis
4	1	11	embedded hardware
	2	3	embedded software

TABLE II
EXAMPLES OF QUESTION FROM THE CHECKLIST

Stage	Set	Respondent	Question
1	1	Architect	Does the reference architecture presents: overview information, release date, version, owner (e.g. organization), change history, short description, scope, domain terminology, open decisions and supporting material?
	2	Architect	Does the detail level favors the reference architecture understanding?
		Architect	Does each view correctly represent its viewpoint?
2	1	All Stakeholders	Do the selected viewpoints frame the concerns of all stakeholders (including domain-specific stakeholders)?
		All Stakeholders	For each viewpoint, are its models clear and well-defined? Do the models provide enough information for determining whether the concerns framed by the viewpoint have been satisfied?
		Architect	Can you show how you produced the list of RA stakeholders and their concerns?
		Domain Expert	Are the domain goals the system must satisfy clearly articulated and prioritized?
		Domain Expert	Is there traceability between the domain goals and the requirements?
		Manager	Does the RA description allow an estimate of the effort for implementing it?
		Manager	Does the reference architecture description show what parts can be implemented using OTS or OSS components?
		Developer	Can you identify the allowed and prohibited dependencies between parts of the RA?
		Integrator	Do you understand the adaptation points of the RA?
		QA stakeholder	Is there a process to ensure conformance with quality attribute requirements?
		Tester	For each partition of the RA, can you determine what is needed (e.g., data, special hardware, other units) to test it?
		Analyst	If the RA is part of a life cycle or process that includes a procurement decision, does the RA contain the appropriate information to support the procurement process?
3	1	Architect	Is the current document complete in the sense that all information is documented? If not, are there placeholders for what has yet to be documented along with descriptions of what still needs to be worked out?
4	1	Domain Expert	How the power supply related decisions affect the reference architecture?
	2	Domain Expert	How embedded operating system considerations affect the reference architecture?

This checklist will be divided by stakeholders, that is, a respondent will see only the questions that must be answered by him/her. Checklist control will allow deadlines, editing questions and specification of respondents. Considering the respondent perspective, it will allow changes and reviews in their answers. FERA-ES Web will also work as a questions repository. Groups interested in contributing will be able to submit new questions, which can be made availabe for all users. The last feature included is the collaboration support. Collaboration will be provided by comments in each question. This feature is essential for the applicability of FERA-ES Web since the checklist requires stakeholders to discuss and collaborate with each other.

III. EVALUATION REPORT

To evaluate the checklist, we consulted experts. They provided insights about the checklist during the checklist construction, and later an evaluation was conducted with the help of a team of experts from our research group.

These experts have been involved with software architecture, reference architecture, and embedded system in their research. Each one of them answered a questionnaire in which each respondent answered about benefits they suspect that the use of the checklist will offer, and also, to elaborated on checklist characteristics.

The collected answers showed that the checklist questions are clear and the support material along the checklist is enough for the evaluation team to understand all the technical terms. Results also pointed out that the checklist has a broad coverage. In addition, the experts suspect that this checklist can help improve the quality of the documentation of the reference architecture. One of the benefits cited by them was flexibility, i.e., the checklist can be easily tailored to fit in a project scope.

Regarding the downsides and disadvantages of using the checklist, the experts pointed out that some questions target stakeholders that may not be known at an early stage of the reference architecture life cycle. However, these questions could be discarded with no harm. Another pointed downside was the difficulty to gather all stakeholders to apply the checklist. FERA-ES Web should be able to overcome this limitation by making these meetings asynchronous.

In addition to the experts evaluation, the checklist was applied to evaluate the SiMuS reference architecture for multi-robotic service systems. The goal of this reference architecture is to support the architectural stage of the development of service robots that will perform tasks as a group [12]. The checklist was distributed among three evaluators, each of which received a specific set of questions based on their qualifications. The evaluation team did not answer all the questions because some of them were not considered suitable for the level of abstraction adopted by the reference architecture. They conducted this evaluation in separate, that is, the evaluators did not collaborate with each other to understand the reference architecture and answer the checklist. Despite the lack of collaboration, they had no doubts regarding the reference architecture or the checklist.

The results of this evaluation pointed out mistakes in the reference architecture. SiMuS description did not express the variation parts and how they would affect fixed parts. The results also pointed out that the SiMuS included no information about the relationship between views and stakeholders. On the other hand, the results showed that the abstraction level was adequate for the reference architecture purposes. Finally, the inspection showed that the SiMuS was ready for publishing but still needed reviews before adoption.

IV. Discussion

Results of our work point out that the checklist is a flexible and direct approach for evaluate reference architectures of embedded systems. The amount of questions is relatively small and the checklist requires minimum effort to be understood and applied. The checklist also has a broad coverage, that is, it treats all the concerns of reference architectures of embedded systems.

Considering knowledge arisen from this work, it is possible to identify interesting and important research lines that can be investigated in future work. For example, the applicability of FERA-ES Web for the checklist application. This tool could help the evaluation team to have a better control over the checklist application and a good understanding of the results. Another possibility is the adaptation of the checklist to fit in other references architecture domain. This adaptation can be accomplished by the replacement of the fourth stage, which holds questions about the embedded systems domain. Thus, the change to other domain should not affect the remaining stages and, therefore, should allow the application of the checklist to other domains.

Regarding limitation of this work, the checklist has only been tested in a research environment, and a validation in industry could help demonstrate the checklist validity, applicability and pertinence. Also, the evaluation did not use a quantitative evaluation approach. Such evaluation would give a better idea of the improvement achieved by this checklist in evaluation of reference architectures of embedded systems.

V. Conclusions

This work presented a checklist for evaluation of reference architectures of embedded systems and demonstrated its applicability. As main result, we showed that the checklist is adequate for use, as considered by the experts, and also demonstrated in the checklist application on SiMuS. We also presented three future work possibilities: The checklist could be used to validate FERA-ES, which is currently under development, to assist its application. It could be adapted to another domain to prove its flexibility. Another possibility is to conduct a case study on a reference architecture used in the industry. As a result we expect to contribute to the quality of reference architectures of embedded systems.

VI. Acknowledgments

This work is supported by Brazilian funding agencies: FAPESP, CNPq and Capes.

References

[1] W. Wolf, *Computers as Components Principle of Embedded Computing System Design*. Morgan Kaufman, 2008.

[2] U. Eklund, O. Askerdal, J. Granholm, A. Alminger, and J. Axelsson, "Experience of introducing reference architectures in the development of automotive electronic systems," in *International workshop on Software engineering for automotive systems*, ser. SEAS '05. New York, NY, USA: ACM, 2005, pp. 1–6.

[3] E. Y. Nakagawa, P. O. Antonino, and M. Becker, "Reference architecture and product line architecture: A subtle but critical difference," in *ECSA*, 2011, pp. 207–211.

[4] ISO/IEC/(IEEE), "ISO/IEC 42010 (IEEE Std) 1471-2000 : Systems and Software engineering - Recomended practice for architectural description of software-intensive systems," 07 2007.

[5] S. Angelov, J. J. M. Trienekens, and P. W. P. J. Grefen, "Towards a method for the evaluation of reference architectures: Experiences from a case," in *ECSA*, 2008, pp. 225–240.

[6] B. Graaf, H. W. van Dijk, and A. van Deursen, "Evaluating an embedded software reference architecture – industrial experience report," in *CSMR*, 2005, pp. 354–363.

[7] R. F. Barcelos and G. H. Travassos, "Arqcheck: Uma abordagem para inspeção de documentos arquiteturais baseada em checklist," in *V Simpósio Brasileiro de Qualidade de Software, SBQS 2006*, 2006, pp. 175 – 188.

[8] 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*, Addison-Wesley, Ed. Addison-Wesley, 2011.

[9] N. Rozanski and E. Woods, *Software Systems Architecture - Working with stakeholders using viewpoints and perspectives*, Addis, Ed. Addison-Wesley Professional, 2011.

[10] E. Y. Nakagawa, F. Oquendo, and M. Becker, "Ramodel: A reference model for reference architectures," in *WICSA/ECSA*, 2012, pp. 297–301.

[11] M. Guessi, E. Y. Nakagawa, F. Oquendo, and J. C. Maldonado, "Architectural description of embedded systems: a systematic review," in *ISARCS*, 2012, pp. 31–40.

[12] D. Feitosa and E. Y. Nakagawa, "Simus - uma arquitetura de referência para sistemas multirrobóticos de serviço." Master's thesis, University of São Paulo, 2013.

Empirical Evidence on Developer's Commit Activity for Open-Source Software Projects

Sihai Lin[1], Yutao Ma[2*], Jianxun Chen[1]
1. College of Computer Science and Technology, Wuhan University of Science and Technology, Wuhan 430065, China
2. State Key Lab of Software Engineering, Wuhan University, Wuhan 430072, China
*e-mail: ytma@whu.edu.cn

***Abstract*—The manner of development is an important factor for the success of open-source software (OSS). Through mining the information of developer's commits, researchers within the community of software engineering can investigate evolutionary aspects of OSS projects and analyze developer's behaviors and collaboration. In this paper we conducted statistical analyses on commit activity for four OSS projects, and found that (1) the commit size in terms of new definitions roughly follows a power-law distribution, and exhibits self-similarity in the temporal dimension; (2) there are five common zones for the distribution of commit activity across various releases in terms of our indicator, and there exists an interesting "deadline effects" in the last zone (i.e. so-called rushing deadline); and (3) developers do prefer to fix bugs in the stage of rushing deadline, perhaps due to deadline pressure. These findings may provide a new insight into schedule planning, resource allocation and quality assurance of OSS projects.**

Keywords-open-source softare; commit; power law; release; self-similarity

I. INTRODUCTION

Over the last two decades, open-source software (OSS) has widely been used by individuals, companies, universities and governments all over the world. For example, as one of the prime examples of OSS, the Apache HTTP Server became the most popular web server software in use in April 1996, and as of December 2012 it was estimated to serve 63.7% of all active websites on the Internet [1]. Usually, the first perceived advantage of OSS is to lower software costs and simplify license management. Hence, more and more enterprises began to use OSS systems instead of commercial software to reduce IT budgets. A Zenoss survey published in 2010 revealed that 98% of the survey respondents indicated usage of open-source systems in their enterprises [2]. According to a report of the Standish Group, the adoption of OSS models has resulted in savings of about $60 billion per year to consumers [3].

OSS's great potential to achieve success depends largely on the manner of development. An OSS system or project is typically built and maintained by a community of volunteer programmers or developers with a decentralized self-organizing organization structure. Since they are distributed among different geographic regions in the world and work for one or more OSS projects at different times, this demands for software tools that can assist their collaborative software development and facilitate source code management and conflict resolution. As we know, revision control systems such as Concurrent Versions System (CVS) and later Subversion (SVN) and Git are well-known examples of such software tools, which could help developers to centrally manage source code files and the changes to those files in the repository of an OSS project.

Apache SVN is such an OSS which can maintain current and historical versions of various types of files in a repository, and has been deemed as a compatible successor to CVS. Every OSS project (on the SourceForge, Google Code, etc.) has a history log for files' each change and the related message ever committed to its SVN repository. Thus, through mining OSS developer's commits, on one hand, researchers can investigate evolutionary aspects of an OSS system as well as its components; on the other hand, this could provide us a statistical way to analyze developer's individual behaviors and teamwork, which might prove useful in quantifying and understanding the dynamics of human behavior on a collective scale [4]. Furthermore, such empirical evidence may offer new insights into schedule planning, resource allocation and quality assurance of an OSS system [5].

Although previous studies focused on commit size distribution [6,7,8], commit classification [6,9,10], developer's contribution and expertise estimation [11,12,13], and task assignment [11,14], as far as we know, it is surprising that there are few representative statistical analyses on commit activity for OSS projects across a certain number of successive releases. To gain a deeper understanding of how an OSS project evolves, in this paper we will analyze 4 projects on the Apache.org in an attempt to answer the following questions: Are there any general rules within the duration between two adjacent releases (e.g. 2.1 and 2.2) that would be found in the whole lifecycle? And, does such an activity within the community of OSS developers have any characteristics similar to those found in professional software companies?

So, the goal of this paper is to explore what distinct feature commit activity possesses and how such an activity is distributed over different releases. Based on case studies, our research could provide empirical evidence for the evolutionary rules of OSS projects. Moreover, this would offer dedicated guidance for OSS developers to draw up project schedule and release a new version better. The main contributions of this paper are listed as follows.

(1) Investigating general statistical laws of commit activity for OSS projects in the whole lifecycle as well as within the duration between two adjacent releases;

(2) Identifying 5 common zones when analyzing the distribution of commit activity across various releases in the temporal dimension;

(3) Exploring the correlation between commit activity within the above-mentioned zones and software bugs fixing (and other types of development activities).

The remainder of this paper is structured as follows**.** Section II introduces related work. Section III explains the analysis method we followed, including research issues and data processing. In Section IV, we present the results of our sample of 4 OSS projects. Finally, Section V concludes this paper and puts forward future work.

II. RELATED WORK

For an OSS project, developers check out a working copy from its SVN repository, update files in the local workspace with the latest ones from the repository, and commit their changes to those local files to the repository. Commit is an important activity for OSS development, so there are a growing number of studies that offer some interesting findings on developer's commits to OSS projects during software lifecycle.

Commit size distribution describes the probability that a given commit is of a particular size [8]. Hattori *et al.* found that the commit size in terms of the number of files follows a Pareto distribution [6], while Arafat *et al.* discovered that the commit size in terms of source lines of code (SLOC) follows a power-law distribution [7]. Such a distribution in terms of lines of code (LOC) is confirmed to be best described by a generalized Pareto model [8] similar to the finding in [7]. The commit size distribution with a long tail shows that developers might conduct large-size commits, though they are less likely to occur. Compared with the previous work, in this paper we will redefine commit size in terms of other indicators, and analyze their statistical distributions.

Because there is no recognized characterization of commit activity, its categorization is still vague. Hattori *et al.* proposed a classification framework in two dimensions (i.e. commit size and the comment of a commit) [6] to relate commits to certain types of activities such as code management and development. In [9], the research examined the version histories of 9 OSS systems to characterize a typical commit according to the number of files, the number of LOC, and the number of hunks committed together, and the results showed that the size categories of commits can be an indicator for the types of maintenance activities being performed. In order to build a categorization of commit types, Dragan *et al.* put forward an automated method by means of the meta-data for the stereotype information of methods added or deleted in a commit [10]. In this paper we plan to explore some characteristic rules recurred in various releases of an OSS project according to the proposed categorization [6].

Besides the above-mentioned studies, recent related research work focused on measuring developer contributions to an OSS project, and on recommending appropriate developers to tackle the changes to a specific file. Gousios *et al.* utilized the LOC that a developer commits to a SVN repository as a basic metric for development effort estimation [11]; Schuler *et al.* mined usage expertise from version archives to recommend experts for specific files [12], because developers who changed a file most often have the probable implementation expertise; considering that a developer who has actually contributed changes to specific files in the past will probably be a good choice of persons for their current or future changes, researchers proposed some new approaches [13,14] to assigning a task that performs software changes to a particular file to an appropriate developer or a ranked list of developers recommended. Actually, this paper has nothing to do with the studies in this field.

III. ANALYSIS METHODOLOGY

A. Research Issues

Like commercial software, OSS always tends to evolve through successive releases with incremental development. In a new release, the system will be updated by adding new functionality, removing redundant components or changing the existing ones, to deliver a set of required functionality or non-functionality such as performance, security and reliability. Changes to specific files committed by developers become an integral part of the system when a new release is delivered, and no other changes will be allowed after the delivery. Then, subsequent changes to files in the system's SVN repository are going to be integrated into future releases. Because there are rarely previous studies taking release into account, in this paper we want to explore the following questions.

Q1: Are there any general rules of commit activity recurred both within the duration between two adjacent releases and in the whole lifecycle? In particular, does the distribution of commit size in terms of new indicators have some form of self-similarity in the temporal dimension?

Q2: Are there any common zones for the distribution of commit activity across various releases? That is to say, we want to detect whether there are any similar zones in which commit activities are active or inactive in terms of commit size.

Q3: If we do find such zones, are there any relationships between certain types of development activities and them? In other words, we are especially concerned whether these zones are related to bug fixing, code refactoring, and other activities.

B. Data Collection

Our methodology is based on case studies, so we selected four OSS projects written in Java on the Apache.org: Apache POI, Tomcat, Struts2, and Derby. The purpose of Apache POI is 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). Struts2 is an elegant, extensible framework for creating enterprise-ready Java web applications, and it uses and extends the Java Servlet API to encourage developers to adopt a model–view–controller (MVC) architecture. Derby is a rational database management

system (RDBMS) developed by the ASF that can be embedded in Java programs and used for online transaction processing.

TABLE I. OSS PROJECTS ANALYZED

Project	Description	Date	Class	Commits
POI	APIs for manipulating file formats	2012-10-12	2,438	8,588
Tomcat	Servlet container		1,980	14,481
Struts2	Framework for web applications		1,521	9,999
Derby	RDBMS		2,974	21,529

Table I shows the brief introduction to the projects analyzed, including the date we conducted our experiments, the number of class files and the total number of commits analyzed. These projects from different application domains were chosen to be experimental subjects based upon that they have been active for at least 2 years. For each project, we retrieved commit history of class files (from their respective main trucks of SVN repository) and release history for the whole system (from their respective websites) till the analysis date, and built sets of commit data within the duration between two adjacent releases according to release history. Moreover, we mined and collected bug information of these projects from Bugzilla and JIRA issues (http://www.atlassian.com/software/jira/overview).

C. Data Processing

As for previous work about the distribution of commit size [6,7,8], the size was defined in terms of the number of files, LOC and other metrics. Here, we first present two new definitions of commit size as follow.

Definition 1. After a class file was created in a SVN repository, *RC* is defined as the number of revisions to the class till its deletion. If a developer commits a change to a specific class from his/her local workspace, the value of *RC* will increase by 1.

Definition 2. From the perspective of a project, *CN* is defined as the total number of commits per time unit (e.g. one day, one week, or one month). It indicates the level of activity of a project during a given unit of time.

To answer the first question presented in the subsection III.A, we examined the distribution of commit size in terms of *RC* and *CN*. Power function, exponential function, polynomial function and logarithmic function were utilized to fit release-level (i.e. the duration between two adjacent releases) and lifecycle-level data sets so as to indentify the best fitting curve and its corresponding function expression. Note that we used a cumulative distribution function (CDF) to reduce noise levels during the estimation of the scaling exponent of power function with the method introduced in [15]. It represents the frequency-of-occurrence of *m* with value greater than or equal to a given number,

$$P_{\geq}(m) = \sum_{m' \geq m} P(m') \approx \int P(m')dm' \sim m^{-r+1} (P(m) \sim m^{-r}). \quad (1)$$

In order to answer the second question, we identified common zones for the distribution of commit activity across various releases according to two parameters, namely the duration between two adjacent releases and *CN*. By normalizing both the former and the latter, their different values for the four OSS projects analyzed could be compared on a notionally common scale. The formulae of such normalization for these two parameters are described as follow.

$$tu' = f(tu) = \frac{tu}{D} = \frac{tu}{\sum tu}, \quad (2)$$

where D is the duration between two adjacent releases, and tu represents the ith time unit within D.

$$cn' = g(cn) = \frac{cn - cn_{\min}}{cn_{\max} - cn_{\min}}, \quad (3)$$

where cn is the total number of commits during the ith time unit, and $cn_{\max}$ and $cn_{\min}$ are the maximum and the minimum value of cn within D, respectively.

However, for the projects analyzed, the distribution of tu' is uneven for different pairs of adjacent releases. Hence, we used a simple partitioning method to cluster neighboring normalized time units across various releases. Then, the normalized total number of commits would be converted to another form of expression, which is the sum of cn' within a new unified time unit for all releases of the 4 projects in question.

$$cn'' = h(cn') = \sum_{tu' \in c} cn', \quad (4)$$

where c is the unified time unit, and its value is equal to the unified normalized D (whose value is 1 in this paper) divided by the number of equal parts K (e.g. 100).

In addition, we analyzed the log message of commits within the identified common zones by using Tf-idf (term frequency–inverse document frequency) to answer the third question. Tf-idf is a numerical statistic which reflects how important a word is to a document in a collection or corpus [16].

$$w_{ij} = tf_{ij} \cdot \log_2 \frac{N}{n}, \quad (5)$$

where w_{ij} is the weight of term T_j in document Doc_i, tf_{ij} denotes the frequency of term T_j in document Doc_i, N indicates the number of documents in corpus, and n is the number of documents where term T_j occurs at least once. The document Doc_i is the set of all log messages of commits within D_i. Hence, we attempt to know what kind of activities developers could do within those common zones in terms of the terms in log messages of commits.

IV. RESULTS AND DISCUSSION

A. Experimental results

Besides the lifecycle of these four OSS projects, we analyzed 11, 10, 20 and 14 releases of POI, Tomcat, Structs2 and Derby, respectively. Due to space limitations, Table II shows the fitting statistics of a small number of samples, including lifecycle-level and two randomly-selected release-level cases. The power exponents for all release-level cases are

presented in Figure 1, where X axis denotes D_i and Y axis represents power exponent. The results show that all cases for *RC* roughly follow a power-law distribution, suggesting an obvious self-similarity. The observation indicates that most of classes are modified a few times, whereas the revisions to a small number of classes are very large within either the duration between two adjacent releases or the whole lifecycle. So, we will pay more attention to these frequently-modified classes in the evolutionary process of projects.

Then, we analyzed the distribution of commit size in terms of *CN* by using the same fitting method, where the time unit is set as day and week. L and R denote lifecycle and release, respectively; D and W represent time unit (i.e. one day and one week). Due to space limitations, Table III only shows the fitting statistics of a small number of samples. For each project, release-level cases are analyzed in terms of the same duration between any two adjacent releases used for *RC*. The results show that all cases for *CN* roughly follow a power-law distribution, suggesting there is also an obvious self-similarity between lifecycle and releases. The observation indicates that the total number of commits in most of time units is small, whereas few time units contribute a large number of commits to its SVN repository.

TABLE II. FITTING CURVES FOR *RC*

Sample	Logarithmic	Polynomial	Exponential	Power
POI (lifecycle)	$y = -472.1\ln(x) + 1527.100$ $R^2=0.645$	$y = 0.987x^2 - 72.919x + 1112.100$ $R^2=0.473$	$y = 474.06e^{-0.109x}$ $R^2=0.878$	$y = 8154.4x^{-1.730}$ $R^2=0.968$
POI (3.5beta4-beta5)	$y = 0.908\ln(0.166x)$ $R^2=0.723$	$y = 0.031x^2 - 0.397x + 1.135$ $R^2=0.795$	$y = 4.107e^{-1.414x}$ $R^2=0.889$	$y = 1.002x^{-2.236}$ $R^2=0.999$
POI (3.5beta5-beta6)	$y = -0.343\ln(0.153x)$ $R^2=0.657$	$y = 0.023x^2 - 0.319x + 0.986$ $R^2=0.707$	$y = 5.664e^{-1.735x}$ $R^2=0.913$	$y = x^{-2.624}$ $R^2=0.999$
Tomcat (lifecycle)	$y = -353\ln(x) + 1396.600$ $R^2=0.712$	$y = 0.161x^2 - 25.095x + 820.030$ $R^2=0.478$	$y = 437.76e^{-0.049x}$ $R^2=0.903$	$y = 11264x^{-2.011}$ $R^2=0.946$
Tomcat (7.0.5beta-7.0.6)	$y = -0.237\ln(0.092x)$ $R^2=0.598$	$y = 0.003x^2 - 0.096x + 0.593$ $R^2=0.526$	$y = 3.075e^{-1.132x}$ $R^2=0.908$	$y = 0.999x^{-1.826}$ $R^2=0.978$
Tomcat (7.0.12-7.0.14)	$y = -0.239\ln(0.080x)$ $R^2=0.680$	$y = 0.004x^2 - 0.101x + 0.625$ $R^2=0.585$	$y = 2.572\ e^{-0.956x}$ $R^2=0.921$	$y = 1.003x^{-1.648}$ $R^2=0.995$
Struts2 (lifecycle)	$y = -418.6\ln(x) + 1426.4$ $R^2=0.838$	$y = 0.500x^2 - 48.791x + 970.16$ $R^2=0.644$	$y = 668.65e^{-1.107x}$ $R^2=0.895$	$y = 13708x^{-1.956}$ $R^2=0.918$
Struts2 (2.0.6-2.0.8)	$y = -0.744\ln(0.127x)$ $R^2=0.856$	$y = 0.178x^2 - 0.397x + 1.135$ $R^2=0.803$	$y = 3.659e^{-0.623x}$ $R^2=0.917$	$y = 0.973x^{-2.224}$ $R^2=0.969$
Struts2 (2.0.9-2.0.11)	$y = -0.277\ln(0.153x)$ $R^2=0.878$	$y = 0.095x^2 - 0.544x + 1.872$ $R^2=0.799$	$y = 2.676e^{-0.899x}$ $R^2=0.925$	$y = x^{-1.876}$ $R^2=0.989$
Derby (lifecycle)	$y = -513.2\ln(x) + 2036.2$ $R^2=0.727$	$y = 0.134x^2 - 28.427x + 1097$ $R^2=0.656$	$y = 448.97e^{-0.042x}$ $R^2=0.806$	$y = 43448x^{-1.884}$ $R^2=0.978$
Derby (10.1.2.1-10.1.3.1)	$y = -0.768\ln(0.433x)$ $R^2=0.748$	$y = 1.094x^2 - 0.709x + 0.863$ $R^2=0.622$	$y = 2.494e^{-0.181x}$ $R^2=0.923$	$y = 1.753x^{-1.963}$ $R^2=0.989$
Derby (10.5.1.1-10.5.3.0)	$y = -0.665\ln(0.565x)$ $R^2=0.767$	$y = 1.345x^2 - 0.168x + 1.213$ $R^2=0.749$	$y = 3.875e^{-1.923x}$ $R^2=0.908$	$y = 1.877x^{-1.281}$ $R^2=0.998$

TABLE III. FITTING CURVES FOR *CN*

Sample	Logarithmic	R^2	Polynomial	R^2	Exponential	R^2	Power	R^2
POI (L-D)	$y=-65.04\ln(2.764x)$	0.791	$y=0.002x^2-0.959x+81$	0.877	$y=370.3e^{-0.078x}$	0.929	$y=1.002x^{-2.236}$	0.999
POI (R-D)	$y=-5.870\ln(2.023x)$	0.820	$y=0.002x^2-0.234x+23.54$	0.895	$y=45.23e^{-0.096x}$	0.908	$y=3.374x^{-1.485}$	0.979
POI (L-W)	$y=-27.5\ln(2.345x)$	0.899	$y=0.001x^2-1.045x+98.86$	0.826	$y=127.6e^{-0.024x}$	0.908	$y=443.7x^{-0.631}$	0.987
POI (R-W)	$y=-1.342\ln(2.177x)$	0.899	$y=0.002x^2-0.763x+11.32$	0.876	$y=23.87e^{-0.075x}$	0.922	$y=46.482x^{-0.130}$	0.989
Tomcat (L-D)	$y=-57.81\ln(2.847x)$	0.851	$y=0.006x^2-2.137x+124.2$	0.653	$y=302.8e^{-0.080x}$	0.912	$y=3441x^{-1.36}$	0.991
Tomcat (R-D)	$y=-17.81\ln(1.192x)$	0.843	$y=0.007x^2-1.182x+23.18$	0.687	$y=28.27e^{-0.085x}$	0.902	$y=201.7x^{-1.281}$	0.993
Tomcat (L-W)	$y=-15.17\ln(0.874x)$	0.902	$y=0.001x^2-0.429x+53.48$	0.897	$y=60.55e^{-0.013x}$	0.928	$y=309.7x^{-0.619}$	0.989
Tomcat (R-W)	$y=-8.473\ln(1.817x)$	0.877	$y=0.001x^2-0.478x+13.47$	0.875	$y=6.945e^{-0.002x}$	0.931	$y=32.79x^{-0.615}$	0.989
Struts2 (L-D)	$y=-72.04\ln(2.544x)$	0.798	$y=0.021x^2-0.433x+90$	0.865	$y=289.3e^{-0.032x}$	0.912	$y=45.85x^{-1.192}$	0.989
Struts2 (R-D)	$y=-32.17\ln(0.817x)$	0.768	$y=0.014x^2-2.192x+13.58$	0.797	$y=56.45e^{-0.092x}$	0.879	$y=310.65x^{-1.018}$	0.976
Struts2 (L-W)	$y=-33.5\ln(1.945x)$	0.889	$y=0.013x^2-1.421x+120.61$	0.866	$y=134.6e^{-0.024x}$	0.919	$y=376.7x^{-0.473}$	0.997
Struts2 (R-W)	$y=-10.28\ln(1.234x)$	0.788	$y=0.006x^2-0.826x+10.14$	0.878	$y=16.181e^{-0.018x}$	0.920	$y=47.35x^{-0.386}$	0.974
Derby (L-D)	$y=-104.23\ln(2.413x)$	0.867	$y=0.065x^2-1.871x+89.2$	0.848	$y=327.8e^{-0.092x}$	0.911	$y=761.2x^{-1.181}$	0.981
Derby (R-D)	$y=-57.38\ln(1.385x)$	0.787	$y=0.017x^2-1.192x+43.46$	0.864	$y=35.67e^{-0.074x}$	0.899	$y=413.5x^{-1.474}$	0.959
Derby (L-W)	$y=-37.65\ln(0.801x)$	0.799	$y=0.008x^2-0.454x+33.48$	0.834	$y=148.55e^{-0.011x}$	0.918	$y=276.7x^{-1.261}$	0.987
Derby (R-W)	$y=-25.23\ln(1.983x)$	0.857	$y=0.007x^2-0.499x+27.28$	0.895	$y=8.967e^{-0.008x}$	0.911	$y=56.43x^{-0.373}$	0.988

According to the results of the distribution of commit size in terms of *CN*, we further processed release-level data (whose time unit is one day) by normalizing and clustering to identify common zones across various releases. The unified *D* was evenly divided into 100 units, and *CN* was re-calculated based on the formula (4). The result for 55 releases of the 4 projects is illustrated by Figure 2, where X axis denotes time line, and Y axis represents normalized *CN*. Taking into consideration the observation that *CN* follows a power-law distribution, we set a threshold to filter out trivial values of data points.

It is obvious from Figure 2 that commits within the zones I and III are less active than those of other zones. On the other hand, it is surprising that commits within the zone V are also active; furthermore, the average of commits within this zone is slightly larger than those of other active zones. This implies that with deadline approaching developers are still busy in preparing a new release, and the phenomenon what we found is similar to software engineers' working overtime before the release of a new version to be delivered in professional companies. That is to say, although the date of a new release of an OSS project is not strictly fixed, within the zone V developers are still rushing out a new release which should be delivered in time, which is known as "deadline effects" [17].

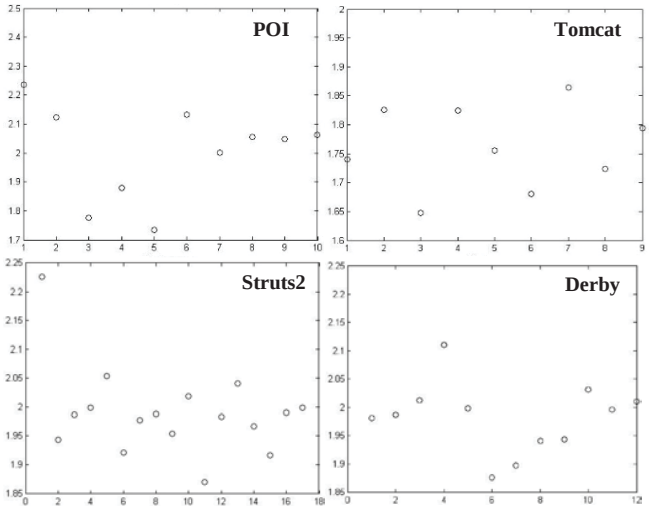


Figure 1. Scatter plot of power exponents for release-level cases.

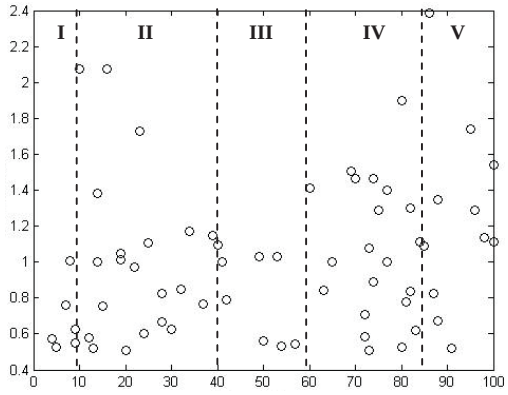


Figure 2. Scatter plot of commits after normalizing and clustering.

Hence, according to the common zones identified in Figure 2, the duration between two adjacent releases (or so-called the period of a new release) could include 5 stages: preparation, active development, interim, active development, and rushing deadline. In the process of iterative and incremental development of software systems, the functional requirements for a new release should be validated and verified by software testing. In order to assure quality, OSS developers often locate and fix software bugs with the support of bug/issue tracking software such as Bugzilla and JIRA issues. Then, we want to explore the distribution of the activity of bug fixing over different stages, and to confirm whether developers tend to fix bugs in the stage of rushing deadline.

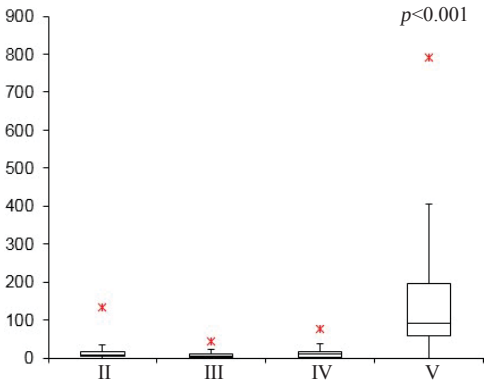


Figure 3. Box-and-whisker plot of bug fixing within various zones.

Because a box plot (also known as a box-and-whisker plot) can display differences between populations without making any assumptions of the underlying statistical distribution, we made use of it to analyze the data of commit log messages about bug fixing for the four projects in question. The result is displayed in Figure 3, where X axis represents four zones (i.e. II, III, IV and V) in Figure 2, and Y axis denotes the number of bugs fixed within the zone. The plot is interpreted as follows: the bottom and top of the box are the 25th and 75th percentile (the lower and upper quartiles, respectively), and the band near the middle of the box is the 50th percentile (the median); the "red star" is the outlier; and *p*-value attached to the plot expresses the probability that the observed difference in the number of bugs fixed among four zones is expected by chance.

Obviously, the difference in the number of bugs that have been fixed among zones is significant over the usual criterion of 99% confidence, which is calculated by using the Kruskal-Wallis test. Statistically significant differences illustrate an interesting finding for OSS project developers who prefer to fix bugs in the stage of rushing deadline, perhaps due to deadline pressure. But for active zones, the difference does not seem so significant. Besides bug fixing, what kinds of common activities developers would perform within these zones?

TABLE IV. TERMS WITH HIGH TF-IDF WEIGHT WITHIN VARIOUS ZONES

POI	Tomcat	Struts2	Derby
avoid	show	misname	compute
record	**import**	rename	contribute
clash	scan	revert	backup
typecast	trim	spend	alter
eliminate	replace	change	attach
mod	complete	build	return
apply	extend	redesign	boot
alleviate	sync	report	prepare
import	suggest	inject	grant
centralize	break	**import**	**import**

Then, Tf-idf (see the formula (5)) was used to analyze the problem. Because it is related to the kind of development activities, we focus on verbs that have high weight in commit log messages. After some highly-frequent terms were filtered

out, a ranked list of terms could be obtained with a simple program. The result of top-10 terms with high weight is listed in Table IV. Considering that these projects analyzed are domain-specific and maintained by different developers, there are no verbs except "import" recurred in all projects, which demands for an elaborate investigation on the relationship between development activities and these common zones.

B. *Threats to validation*

Although we tried to diversify the characteristics of projects by carefully choosing four different OSS projects, some of our findings may not be generalized to other projects. In addition, these findings may not be suitable for industrial software systems, since OSS projects have particular characteristics different from those of commercial software.

In fact, an active OSS project evolves over time. The whole lifecycle presented in this paper is an approximate estimation of the real entire lifecycle. Even so, we argue that the power-law distribution of *RC* and *CN* for the period of development in the future still holds because of the well-known "Matthew effect". The number of partition units *K* for the unified duration between two adjacent releases may influence the common zones that have been identified in this paper. So, we will seek to detect more accurate range and boundary of these common zones by analyzing more OSS projects. Another issue concerns the usage of log messages as the information of bug fixing, because there is lack of standardization for commit log messages. The method depending on the frequency of keywords to analyze the bug information in our study may be slightly biased.

V. CONCLUSION AND FUTURE WORK

Developers from all over the world can work together to develop an OSS project with the support of revision control systems such as CVS and SVN. Commit is an important activity of such OSS development, which attracted increasing attention from researchers within the community of software engineering. In this paper we conducted statistical analyses on commit activity for 4 OSS projects across a certain number of successive releases, and uncovered the following findings:

(1) The commit size in terms of *RC* and *CN* roughly follows a power-law distribution, within both the duration between any two adjacent releases and the whole lifecycle, implying an obvious self-similarity in the temporal dimension;

(2) We analyzed 55 releases of these 4 projects in question in terms of *CN* at the scale of one day, identified 5 common zones for the distribution of commit activity, namely preparation, active development, interim, active development and rushing deadline, and found an interesting "deadline effects" in the stage of rushing deadline, though the date of a new release of these projects is not strictly fixed;

(3) We mined commit log messages and bug information from Bugzilla and JIRA, and found statistically significant differences in the number of bug fixed among 4 common zones, suggesting that developers do prefer to fix bugs in the stage of rushing deadline, perhaps due to deadline pressure.

The future work is to detect more accurate range and boundary of these common zones by analyzing more OSS projects, and to relate the characterization of these common zones to more types of development and maintenance activities to facilitate evaluating the spread of bugs.

ACKNOWLEDGEMENT

This work is supported by the National Natural Science Foundation of China under grant Nos. 61272111 and 61100017.

REFERENCES

[1] http://en.wikipedia.org/wiki/Apache_HTTP_Server

[2] Zenoss Inc., "2010 Open Source Systems Management Survey," http://community.zenoss.org/servlet/JiveServlet/download/38-3009/ OpenSourceManagement.pdf, 2010.

[3] Standish Group International, "The Trends in Open Source," http://www.marketwire.com/press-release/free-open-source-software-is-costing-vendors-60-billion-new-standish-group-international-844462.htm, 2008.

[4] A.-L. Barabási, "The origin of bursts and heavy tails in human dynamics," Nature, 435(7039): 207–211, 2005.

[5] B. O'Sullivan, "Making sense of revision-control systems," Communications of the ACM, 52(9): 56–62, 2009.

[6] L. Hattori and M. Lanza, "On the nature of commits," In Proceedings of the 4th International ERCIM Workshop on Software Evolution and Evolvability at the 23rd IEEE/ACM International Conference on Automated Software Engineering, Italy, 2008, pp. 63–71.

[7] O. Arafat, and D. Riehle, "The Commit Size Distribution of Open Source Software," In Proceedings of the 42nd Hawaii International Conference on Systems Science, USA, 2009, pp. 1–8.

[8] C. Kolassa, D. Riehle, M. A. Salim, "A Model of the Commit Size Distribution of Open Source," In Proceedings of the 39th International Conference on Current Trends in Theory and Practice of Computer Science, Czech Republic, 2013, pp. 52–66.

[9] A. Alali, H. Kagdi, J. I. Maletic, "What's a Typical Commit? A Characterization of Open Source Software Repositories," In Proceedings of the 16th IEEE International Conference on Program Comprehension, The Netherlands, 2008, pp. 182–191.

[10] [N. Dragan, M. L. Collard, M. Hammad, J. I. Maletic, "Using stereotypes to help characterize commits," In Proceedings of the 27th IEEE International Conference on Software Maintenance, USA, 2011, pp. 520–523.

[11] G. Gousios, E. Kalliamvakou, D. Spinellis, "Measuring developer contribution from software repository data," In Proceedings of the 2008 International Working Conference on Mining Software Repositories, Germany, 2008, pp. 129–132.

[12] D. Schuler and T. Zimmermann, "Mining usage expertise from version archives," In Proceedings of the 2008 International Workshop on Mining Software Repositories, USA, 2008, pp. 121–124.

[13] H. H. Kagdi, M. Hammad, J. I. Maletic, "Who can help me with this source code change?" In Proceedings of the 24th IEEE International Conference on Software Maintenance, China, 2008, pp. 157–166.

[14] H. Kagdi, M. Gethers, D. Poshyvanyk, M. Hammad, "Assigning change requests to software developers," Journal of Software: Evolution and Process, 24(1): 3–33, 2012.

[15] A. Clauset, C. R. Shalizi, M. Newman, "Power-law distributions in empirical data," SIAM Review, 51(4): 661–703, 2009.

[16] http://en.wikipedia.org/wiki/Tf%E2%80%93idf

[17] C. Fershtman, D. J. Seidmann, "Deadline effects and inefficient delay in bargaining with endogenous commitment," Journal of Economic Theory, 60(2): 306–321, 1993.

The Impact of Confirmation Bias on the Release-based Defect Prediction of Developer Groups

Gul Calikli and Ayse Bener
Ryerson University, Data Science Laboratory,
Department of Mechanical and Industrial Engineering
Toronto, CANADA
{gcalikli, ayse.bener}@ryerson.ca

***Abstract*—During software development life cycle (SDLC), source codes are created and updated by groups of one or more developers. Information about the defect rates introduced by developer groups for the current release of a software product might guide project managers to form developer groups in order to manage defect rates for the next releases. In this research, we use partial least squares regression (PLSR) and principal component regression (PCR) to model the relation between defect rates and a specific cognitive aspect of developers, namely *confirmation bias.* In order to empirically estimate the performance of our model, we use datasets from three industrial software projects.**

Keywords-software engineering; defect rates; software psychology; confirmation bias; release management.

I. INTRODUCTION

In software development, managers need to make decisions under uncertainty to allocate valuable resources effectively. Prediction models are oracles to guide project managers in taking these critical decisions. In empirical software engineering literature, there are various decision-making models that aim to predict pre-release defects so that exhaustive testing is prevented and more defects are detected in shorter times [1], [2]. This results in both increased efficiency of the software testing phase and timely delivery of the software product to the market. There are also models that can be useful in the early detection of post-release defects before the software is released to the market [3].

Learning based prediction models usually take static code attributes (i.e. McCabe and Halstead metrics) as input to train the algorithms. There are also prediction models that use other metric types, such as design metrics that are extracted from requirement documents, development history metrics (i.e. churn metrics) [1], [4]. Combinations of different metric types were also used to enhance performance of prediction models. For instance, Nagappan and Ball [5] combined dependency and churn metrics to predict post-release faults in the binary files of Windows Server 2003. Misirli-Tosun et al. [6] used network and churn metrics in order to build defect predictors for different defect categories. Jiang et.al used combination of static code and design metrics that were extracted from requirements documents [4]. Turhan et al. [7] reduced the probability of false alarms by supplementing static code metrics by their call graph based ranking (CGBR) framework

In order to enrich the content of input data, the above-mentioned studies focused primarily on product and process attributes. However, the thought processes of people may have a significant impact on software defect density, as software is developed and tested by people [8], [9]. Therefore, it is highly likely that further progress in defect prediction performance may be achieved by taking into account cognitive aspects of people. Developers use some heuristics to solve the problems they encounter during SDLC and these heuristics may lead to cognitive biases, which are defined as the deviations of human mind from the laws of mathematics and logic. In this research, we focus on a specific cognitive bias type called *confirmation bias*. There are empirical studies showing the existence of confirmation bias among software engineers and programmers [8].

The prediction model that we build in this research is different than the ones in the literature in the following aspects:

- Instead of predicting defect-prone parts of software, our model predicts the defect rates of developer groups.
- Majority of existing prediction models use product and process related metrics. There are few models that employ people-related metrics [10], [11], [12]. However, they do not take into account people's thought processes. Our model uses information regarding developers' *confirmation biases*. In our previous work [13], we used confirmation biases to build models to predict defect prone parts of software.
- Our proposed model performs "release-based" prediction. The model cumulatively learns from the past releases to predict the next release. Most existing prediction models in the literature perform 10 fold cross validation regardless of the release information, since they employ public data sets.

The rest of the paper is organized as follows. Section 2 gives background on prediction models in software engineering and confirmation bias. Section 3 describes our empirical work consisting of data, methodology, metric collection, and prediction model. Section 4 discusses the results of the study. We discuss threats to validity in Section 5. Section 6 concludes and mentions potential future directions.

II. Related Background

A. Prediction Models to Guide SDLC

Models for assessing software failure risk, in terms of predicting defects in a specific module or function, have been proposed in previous research [1], [2], [14], [15]. Software testing is the most popular defect detection method, however, when the size of projects grow in terms of both lines of code and effort spent, testing becomes more computationally expensive. Popular ways of detecting software defects prior to testing phase are expert judgments and manual code reviews. Although code reviews are accepted to be an effective solution they are labor intensive [5]. Therefore, research in this area has focused on finding cost-effective defect detection/ prediction methods [1] .

Software development is one domain in which remarkably effective defect prediction models have been generated by using learning-based models and data mining methods [1-7], [14], [15] These models took into account product (static code metrics, repository metrics, etc.), and process (organizational factors, experience of people, etc.) attributes. However, in software development projects *people* (developers, testers, analysts) are the most important pillar, but very difficult to model. In this research, we focus on a specific cognitive aspect of people, namely *confirmation bias.*

B. Confirmation Bias in Cognitive Psychology

In cognitive psychology, confirmation bias is defined as the tendency of people to seek for evidence that could verify their hypotheses rather than seeking for evidence that could falsify them. Wason used the term "confirmation bias" for the first time, in his rule discovery task [16] and later in his selection task [17].

The experimental procedure of Wason's selection task can be explained as follows: Initially, the subject is given a record sheet on which the triple "2 4 6" is written and (s)he is told that "2 4 6" conforms to this rule. In order to discover the rule, the subject is asked to write down triples together with the reasons of his/her choice on the record sheet. After each instance, the examiner tells the subject whether the instance conforms to the rule or not. The subject can announce the rule only when (s)he is highly confident. If the subject fails to discover the rule at the first attempt, (s)he can continue giving instances together with reasons for his/her choice. This procedure continues iteratively until either the subject discovers the rule or (s)he wishes to give up. However, if the subject cannot discover the rule in 45 min, the experimenter aborts the procedure.

On the other hand, in Wason's selection task, the subject is given four cards, where each card has a letter on one side and a number on the other side. These four cards are placed on a table showing D, K, 3, 7, respectively. Given the rule "Every card that has a D on one side has a 3 on the other side", the subject is asked which card(s) must be turned over to find out whether the rule is true or false.

C. Confirmation Bias in Relation to Software Development

Due to confirmation bias, developers may perform only the unit tests that make their program work rather than breaking the code. This may lead to an increase in software defect density. In literature, there are empirical studies showing the existence of confirmation bias among software developers [8], [9].

There are similarities between Wason's rule discovery task and functional (black-box) testing that are performed by software developers to test the functional units of their codes during unit testing [8]. According to the findings of Wason, during the rule discovery task, subjects have a tendency to select many triples (i.e., test cases) that are consistent with their hypotheses and few tests that are inconsistent with them. Similarly, program testers may select many test cases consistent with the program specifications (positive tests) and a few that are inconsistent with them (negative tests). Moreover, the number of possible test cases is either infinite or too large to be tested within a limited amount of time. Consequently, a strategic approach must be followed that covers both positive and negative test cases while trying to make the code fail during testing in order to find as many defects as possible.

Wason's selection task measures the extent of subject's logical reasoning skills. During unit testing when covering possible scenarios, logical reasoning is required. Moreover, testing the correctness of conditional statements in the source code during white-box testing also requires logical reasoning skills. Detailed explanation of the analogy between Wason's selection task and white box testing by Stacy and MacMillian can be found in [9].

III. Methodology to Quantify Confirmation bias

In order to perform empirical analysis, it was necessary to quantify/ measure confirmation biases of software engineers. For this purpose, we developed a methodology to define a confirmation bias metrics set and to extract metrics values. Details of our methodology to define and extract confirmation bias metrics can be found in our previous work [13] and it mainly consists of the following steps:

A. Preparation of the Confirmation Bias Test

Confirmation bias test consists of written questions and an interactive question. Interactive question is Wason's rule discovery task itself [16], while written test is based on Wason's selection task [17]. Written test consists of 7 abstract and 16 thematic questions, 9 of which have software development/testing theme. Abstract questions require logical reasoning skills to be answered correctly, while real life experience and/or memory cueing [19] can help to answer thematic questions correctly. Both abstract and thematic questions were prepared based on the psychological experiments conducted to show the existence of confirmation bias among people [16], [17], [19]. Initially, we administered confirmation bias test to a pilot group consisting of 28 Computer Engineering PhD candidates, half of whom had at least two years of commercial software product development experience. After pilot study, we administered the test to software engineers in various large-scale companies and Small Medium Enterprises (SMEs). So far, we have administered the

test to 199 software engineers from 4 large-scale companies and 3 SMEs.

B. Preparation of the Confirmation Bias Metrics Set

The initial metric suite was formed concurrently with the preparation of the interactive question and the set of written questions. Statistical analysis and feature selection techniques helped to eliminate metrics that displayed a lower level of significance in the measurement of confirmation bias. Our metrics set evolved as our research progressed [20], [21], [22] and our final set of metrics are summarized in [13].

IV. Empirical Study

A. Datasets

In this study, we used datasets from three different projects as shown in Table 1. In order to build our prediction model, we took into account only the source code files whose development activities can be traced through the version control system, since project managers needed information about the performance of developer groups who maintain/develop these files.

TABLE I. Properties of Datasets

Datasets	Number of Active Files	Number of Releases [a]	Defect Density	Number of Developers
ERP	3199	7	0.07	6
Telecom1	826	4	0.11	10
Telecom2	1828	6	0.11	10

a. Defect density is the ratio of the number of defective files to the number of active files.

Dataset ERP belongs to a project group that consists of six developers who are employees of the largest ISV (independent software vendor) in Turkey. The software developed by this project group is an enterprise resource planning (ERP) software. The snapshot of the software that was retrieved from the version management system dates from March 2011, and it consists of 3,199 java files.

The remaining two datasets come from the largest wireless telecom operator (GSM) company in Turkey. Dataset Telecom1 consists of four versions of a software product that is used to launch new campaigns. On average, 545 java files exist in a single version, and they make modifications to 206 files per version (also on average).

Dataset Telecom2 comes from the billing and charging system and it consists of source code files of the billing and revenue collection systems, as well as the database transaction system. On average, there are 1,092 java, JSP and PL/SQL files in a single version of this software package. However, maintenance, development and software testing activities take place on only 284 of those files.

B. Methodology

1) Metric Collection Process: In order to collect confirmation bias metrics in a controlled manner, we administered the confirmation bias test under a predefined standard procedure. After having conducted the test with developers, we evaluated values for the metrics from the test outcomes.

2) Identification of Developer Groups: For each project, we mined the log file, that was extracted from the version management system. Within each release of a project, for each file, we identified developers, who committed that file before the code freeze date of that release. As a result, we were able to determine groups of developers who committed a common set of files for each release.

3) Estimation of Confirmation Bias Metrics Values of Developer Groups: After having extracted metrics values for each developer individually, we used three different operators (i.e. *min*, *max* and *avg*) in order to estimate confirmation bias metrics values for developer groups.

Assuming that A_{di} represents the i^{th} confirmation bias metric value of d^{th} developer, $d \in G_j$ means that d^{th} developer is among the group of developers who created and/or modified j^{th} source file, and finally, S_{ji}^{op} represents the resulting i^{th} confirmation bias metric value of j^{th} source file when operator op is applied. op can be one of the operators *min*, *max* and *avg*, which are used to find minimum, maximum and average values of the i^{th} confirmation bias metric, respectively. We can formulize the definition for the *min*, *max* and *avg* operators as follows:

$$S_{ji}^{max} = \max\left(A_{di} | \forall d \in G_j\right) \quad (1)$$

$$S_{ji}^{min} = \min\left(A_{di} | \forall d \in G_j\right) \quad (2)$$

$$S_{ji}^{avg} = \text{avg}\left(A_{di} | \forall d \in G_j\right) \quad (3)$$

4) Estimation of Defect Rate for each Developer Group: We defined the defect rate for each developer group as the ratio of the total number of defective files created/updated by that group to the total number of files that group created/updated. Equation (4) gives the formulation of the defect rate dr^i for i^{th} developer group, where $N^i_{defectiveFiles}$ and $N^i_{allFiles}$ stand for the number of defective files and number of all files, respectively.

$$dr^i = N^i_{defectiveFiles} / N^i_{allFiles} \quad (4)$$

5) Construction of the Prediction Models: PLSR and PCR are both methods to model a response variable when there are large number of predictor variables, and those predictors are highly correlated or even collinear. Our metrics set consists of 25 metrics as listed in our previous paper [13]. As a result of applying *min*, *max* and *avg* operators, total number of confirmation bias metrics for each developer group becomes 75. Moreover, these metrics are highly correlated. Therefore, in this study we used Partial Least Squares Regression (PLSR) and Principle Components Regression (PCR) methods to build prediction models. Both methods construct new predictor variables, known as components, as linear combinations of the original predictor variables, but they construct those

components in different ways. PCR creates components to explain the observed variability in the predictor variables, without considering the response variable at all. On the other hand, PLSR does take the response variable into account.

In our research, the predictor variables, which we employed in both PLSR and PCR, are confirmation bias metrics values of developer groups. The response variables employed for the PLSR method are the defect rates of developer groups. In order to calculate the regression coefficients for each release i, where $i = 2, \ldots, N$, such that N is the total number of releases, we used the data , which belong to the first $i\text{-}1$ releases (i.e., for j releases, where $j = 1,\ldots,i\text{-}1$). We later used the estimated regression coefficients to predict the defect rates for the i^{th} release. We performed this procedure for all datasets/software projects (i.e., ERP, Telecom1 and Telecom2).

V. RESULTS AND DISCUSSIONS

In this study, we used PLSR and PCR to build models in order to predict defect rates of developer groups for the next releases of the software. In order to measure the prediction performance of our models, we used Mean Squared Prediction Error (MSEP).

$$MSEP = \frac{1}{N}\sum_{i=1}^{N}(\widehat{dr^i} - dr^i)^2 \qquad (5)$$

In the above equation, N is the total number of developer groups, while $\widehat{dr^i}$ and dr^i are the actual and estimated defect rates of the i^{th} developer group, respectively.

As mentioned previously PLSR and PCR, construct new predictor variables, known as components, as linear combinations of the original predictor variables. As a first step in our analyses, we determined the total number of components to be used in our models in order to minimize MSEP values. For this purpose, we used all data, which we

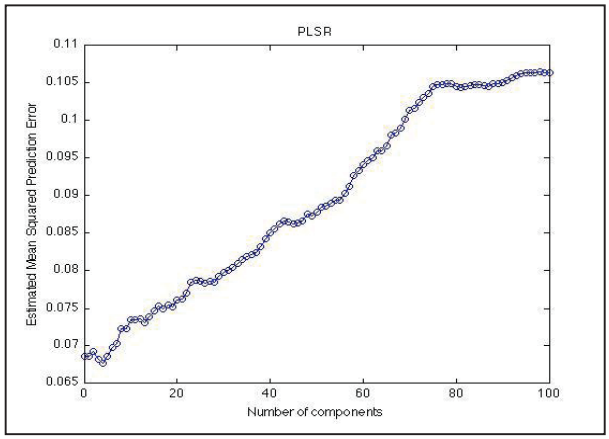


Figure 1. Number of components versus estimated MSEP.

collected from three different software projects. As it can be seen from Figures 1 and 2, using four components minimizes MSEP value. Hence, we decided to use four components, while employing both PLSR and PCR techniques to build our models.

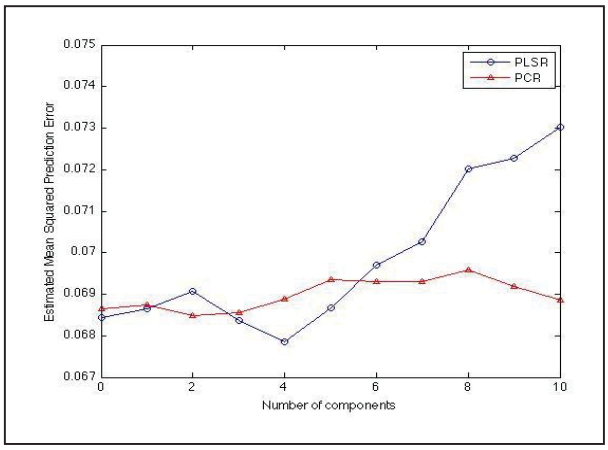


Figure 2. Number of components versus estimated MSEP (closer look).

Performance results of the prediction models are shown in Tables I, II and III for datasets ERP, Telecom1, and Telecom3, respectively. The release labels, which are listed in the first column of each of the Tables I, II and III are the actual release labels used by the software companies to name the releases of their software product.

TABLE II. RESULTS FOR DATASET ERP

		MSEP	
Releases	**Releases used for Model Building**	**PLSR**	**PCR**
2010_R2	2010_R1	0.0421	0.0108
2010_R3	2010_R1, 2010_R2	0.0746	0.0730
2010_R4	2010_R1-2010-R3	0.0504	0.0127
2010_R5	2010_R1-2010_R4	0.0138	0.0061
2010_R6	2010_R1-2010_R5	0.0224	0.0182
2011_R1	2010_R1-2010_R6	0.0176	0.0011

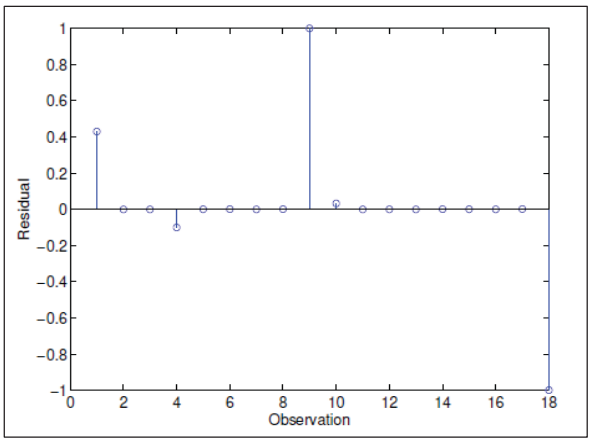


Figure 3. Residual plots for the defect rates of the fourth release of the ERP project.

As it can be seen from Tables I, II and II, both of our results are promising, although PCR slightly outperforms PLSR. This is due to the fact that within each release of each software product (i.e., dataset), on average 65% of the defect rate values (i.e., response variables) are 0.

TABLE III. RESULTS FOR DATASET TELECOM1

		MSEP	
Releases	**Releases used for Model Building**	**PLSR**	**PCR**
2.60	2.59	0.0525	0.0450
2.61	2.59, 2.60	0.0253	0.0078
2.62	2.59-2.61	0.0191	0.0054

Unlike PCR, PLSR considers the observed response values, while finding the linear combinations of predictor variables. Therefore, the slight degradation in the prediction performance of PLSR-based models might be due to the high percentage of response variables being equal to zero.

In addition, for all three datasets the trend in the defect rate prediction performance is improvement, while moving from earlier releases to the more recent ones. This was an expected result, since size of the data used to estimate the regression coefficients increases for the recent releases. For instance, for dataset Telecom1, in order to estimate the regression coefficients of the model to predict defect rates of the release 2.60, only the data belonging to release 2.59 is used. On the other hand, in order to build the prediction model for release 2.62, all data belonging to the previous releases 2.59, 2.60 and 2.61 are used. This also confirms our previous finding in release based defect prediction model that the more information content of the model is enriched the better the prediction performance of the model [1].

TABLE IV. RESULTS FOR DATASET TELECOM2

		MSEP	
Releases	**Releases used for Model Building**	**PLSR**	**PCR**
2.92	2.91	0.0217	0.0272
2.93	2.91, 2.92	0.0552	0.0072
2.94	2.91-2.93	0.0398	0.0060
2.95	2.91-2.94	0.0148	0.0041
2.96	2.91-2.95	0.0081	0.0021

VI. THREATS TO VALIDITY

In order to avoid mono-method bias that is one of the threats to construct validity, we used more than a single version of a confirmation bias measure. In other words, we defined a set of confirmation bias metrics.

Another threat to construct validity is the interaction of different treatments. Before the administration of confirmation bias tests to participant groups, we ensured that none of the participants were involved simultaneously in several other experiments designed to have similar effects.

Evaluation apprehension is a social threat to construct validity. Many people are anxious about being evaluated. Hence, participants may perform poorly due to their apprehension, and they may feel psychologically pressured. In order to avoid such problems, informed the participants that the results would not be used in their performance evaluations and their identity would be kept anonymous. Moreover, participants were told that there was no time constraint for completing the questions.

Another social threat to construct validity is the expectancies of the researcher. Hence, the outcome of the confirmation bias test was independently evaluated by two researchers and one of these two researchers was not actively involved in the study.

In order to avoid internal threats to validity, we set the test dates for all project groups for a time when the work load of the developers was not intense. No event took place in between the confirmation bias tests that could have influenced the performance of the subjects in any of the groups. Another attempt to avoid internal validity was to administer the confirmation bias test in environments that were isolated from distraction factors such as noise.

To avoid external threats to validity, we collected data from two different companies specialized in two different software development domains. We also selected two different projects within one of the companies.

VII. CONCLUSIONS AND FUTURE WORK

In this research by using the confirmation bias metrics, we predicted which developer groups are likely to have defective code in the next release. We built a release-based prediction models by using PLSR and PCR. We compared the prediction results with actual data and obtained promising results.

Our current models are capable of predicting the defect rates of developer groups, which do not exist in previous releases of the software product. In such cases, there are no actual defect rates, which can be used to predict the performance of these developer groups for the next releases. In this study, we used the information about confirmation biases of such developer groups to predict defect rates for the next releases.

As mentioned previously, confirmation bias metrics values of individual developers can be estimated based on the confirmation bias test outcomes. Using *min*, *max* and *avg* operators, it is also possible to estimate confirmation bias metrics values of any possible developer group combination. Therefore, in the long run, enhanced form of our models may guide project managers in task assignment issues (i.e., which developers should touch the same source code files and which developers should not, so that software defects rates are minimized). For this purpose, we aim to replicate our results in further studies. Moreover, our results only indicate the correlation between confirmation biases of developers and defect rates. In this study, we interviewed with the developers and asked them about the unit testing strategies they adopt. In order to strengthen our hypothesis regarding the connection between confirmation bias and defect rates through unit testing, as future work, our field studies will also include observational techniques (e.g., think-aloud protocols, participant observation and observation synchronized shadowing).

Going forward, we would also like to develop a web-based tool to administer confirmation bias tests to developers. This tool will also estimate confirmation bias metrics values from developers' test outcomes, as well as handling noise in the data. Such a tool would make our method easier to use in practice.

REFERENCES

[1] A. Tosun, A. Bener, B. Turhan, and T. Menzies. "Practical considerations in deploying statistical methods for defect prediction: A case study within the turkish telecommunications industry.," Information and Software Technology, 52(11):1242-1257, November 2010.

[2] T. Menzies, C. J. Hihn, and K. Lum, "Data mining static code attributes to learn defect predictors," IEEE Transactions on Software Engineering, vol. 33, issue 1, pp. 2-13, November 2007.

[3] N. Nachi, "Toward a software testing and reliability early warning suite," Proceedings of the 26th International Conference on Software Engineering (ICSE 2004), pp. 60-62, May 2004.

[4] Y. Jiang, B. Cuki, T. Menzies, and N. Bartlow, "Comparing design and code metrics for software quality prediction," in Proceedings of the 4th International Workshop on Predictor Models in Software Engineering, 2008.

[5] N. Nachi and T. Ball, "Using software dependencies and churn metrics to predict field failures: an empirical study," in the Proceedings of the 1st International Symposium on Empirical Software Engineering and Measurement (ESEM 2007), pp. 364-373, September 2007.

[6] A. Tosun-Misirli, B. Caglayan, A. Miransky, A. Bener, and N. Ruffalo, "Different strokes for different folks: a case study on software metrics for different defet categories," in Proceedings of the 2nd Workshop on emerging Trends in Software Metrics, pp. 45-51, May 2011.

[7] B. Turhan, G. Kocak and A. Bener, "Software defect prediction using call graph based ranking (CGBR) framework," in Proceedings of the 34th International EUROMICRO Software Engineering and Advanced Applications Conference, pp. 45-51, July 2008."

[8] B. F. Teasley, L. M. Leventhal, C. R. Mynatt, and D. S. Rohlman. "Why software testing is sometimes ineffective: Two applied studies of positive test strategy," Journal of Applied Psychology, vol. 79, pp. 142-155, November 1994.

[9] W. Stacy and J. MacMillian, "Cognitive bias in software engineering," Communications of the ACM, vol. 38, pp.:57-63, November 1995.

[10] C. Bird, N. Nagappan, H. Gall, B. Murphy and P. Devanbu, "Putting it all together: Using sociotechnical networks to predict failures," in Proceedings of the 17th International Symposium on Software Rreliability Engineering, pp. 109-119, 2009.

[11] E. Weyuker, T. J. Ostrand and R. M. Bell, "Using developer information as a factor for fault prediction," Proceedings of the 1st International Workshop on Predictor Models in Software Engineering, pp. 1–7, 2007.

[12] A. Meneely, L. Williams, W. Snipes and J. Osborne, "Predicting failures with developer networks and social network analysis'" in Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 13–23, 2008.

[13] G. Calikli and A. Bener, "Influence of Confirmation Biases of Developers on Software Quality: An Empirical Study", vol. 21, pp. 377-416, Software Quality Journal, 2013.

[14] T. M. Khoshgoftaar, J. Van Hulse and A. Napolitano, "Supervised neural network modeling: An empirical investigation into learning from imbalanced data with labeling errors," IEEE Transactions on Neural Networks, vol. 21, pp. 813–830, 2010.

[15] T. M. Khoshgoftaar, "Building decision tree software quality classification models using genetic programming," In Proceedings of the genetic and evolutionary computation conference, 2003.

[16] P. C. Wason, "On the failure to eliminate hypotheses in a conceptual task," Quarterly Journal of Experimental Psychology, vol. 12, pp.129–140, 1960

[17] P. C. Wason, "Reasoning about a rule," Quarterly Journal of Experimental Psychology, vol. 20, pp. 273–281, 1968.

[18] B. Teasley, L. M. Leventhal, and S. Rohlman, "Positive test bias in software engineering professionals: What is right and what's wrong'" in Proceedings of the 5th Workshop on Empirical Studies of Programmers,1993.

[19] J. St. B. T. Evans, S. E. Newstead and R. M. Byrne, Human reasoning: The psychology of deduction. East Sussex, UK: Lawrence Erlbaum Associates Ltd., 1993.

[20] 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 32nd International Conference on Software Engineering, 2010.

[21] G. Calikli, B. Arslan and A. Bener, "Confirmation bias in software development and testing: an analysis of the effects of company size, experience and reasoning skills" in Proceedings of the 22nd Annual Psychology of Programming Interest Group Workshop, 2010.

[22] G. Calikli and A. Bener, "Empirical analyses factors affecting confirmation bias and the effects of confirmation bias on software developer/tester performance" in Proceedings of 5th International Workshop on Predictor Models in Software Engineering, 2010.

A Study on First Order Statistics-Based Feature Selection Techniques on Software Metric Data

Huanjing Wang
Western Kentucky University
Email: huanjing.wang@wku.edu

Taghi M. Khoshgoftaar, Randall Wald, and Amri Napolitano
Florida Atlantic University
Email: {khoshgof,rwald1}@fau.edu, amrifau@gmail.com

***Abstract*—Software quality is an important attribute of software products, especially for high-assurance and mission-critical systems. One effective way to produce a high quality software product is to build software quality prediction models which use software metrics collected during the course of software development to identify potentially fault-prone modules. Feature selection can be used to determine which software metrics (which act as features) are most useful for constructing these models. In this paper, we investigate seven first order statistics-based feature selection techniques, evaluating both their stability (ability to produce consistent feature subsets even in the face of changes to the data) and classification performance (ability to select software metrics which are useful for building accurate models) in the context of software quality estimation by testing them on data from a very large telecommunications software system. Our investigation considers four degrees of perturbation, four different sizes of feature subsets, and three different classifiers. The empirical results demonstrate that Signal-to-Noise is the most stable ranker and also performs well in terms of model performance. We also find that while for some rankers, stability and classification performance are correlated, this is not true for other rankers, and therefore performance according to one scheme (stability or model performance) cannot be used to predict performance according to the other.**

I. Introduction

One major challenge in large-scale software development is the inevitable problem of fault-prone software modules. Even in professional software development teams, these cannot be avoided entirely, so quality assurance is necessary to review modules to catch potential faults. However, there are limited resources for quality assurance, so only a fraction of the total project can be reviewed. To help target these quality-assurance efforts, software defect prediction models can use software measurement data (e.g., software metrics) collected during the course of software development to identify software modules which are likely to be particularly fault-prone. However, many data repositories have an overabundance of metrics (features) [1], [2]. In addition, studies also show that not all features make equally important contributions to the problem at hand (in the case of software quality prediction, whether the module is fault-prone (*fp*) or not-fault prone (*nfp*)) [3]. Thus, the selection of software metrics that are important for software defect prediction is critical. This can be accomplished using feature selection, a collection of preprocessing techniques drawn from the domain of data mining and machine learning which choose an optimum subset of features for subsequent analysis. Although feature selection uses a reduced feature subset for building a classification model, it often creates models with performance equal to or even better than models using all features.

During the past decade, numerous studies have examined feature selection with respect to classification performance, but very few studies focus on the stability (robustness) of feature selection techniques. Stability is a measure of how consistent a feature selection technique is when faced with changes in the data. In this study, we evaluate the stability of a feature selection method on a dataset by measuring the changes between the subset chosen using the full dataset and that chosen from modified datasets with instances removed. One could also view this information as before and after instances have been added to the dataset, and thus view this evaluation as how relevant the chosen features will be after more instances are collected. In addition to examining how choice of feature selection technique, size of feature subset, and degree of dataset perturbation can affect the feature subset stability, we also evaluate the effectiveness of defect predictors that estimate the quality of program modules, e.g., fault-prone (*fp*) or not-fault-prone (*nfp*). After all, a feature selection technique must be both stable and accurate (providing features useful for classification) in order for it to be a good technique.

In this paper, we empirically investigate seven feature selection techniques: Signal-to-Noise (S2N), Fold Change Difference (FCD), Fold Change Ratio (FCR), Welch T-Statistic (WTS), Wilcoxon Rank Sum (WRS), Fisher Score (FS), and Significance Analysis of Microarrays (SAM). We refer to these as First Order Statistics (FOS) based feature selection, because they are all based on first order statistics such as mean and standard deviation. We compared stability and classification model performance of the feature subsets selected by the seven feature selection techniques. Three different classifiers (learners) are used to build our prediction models. The empirical validation of the stability measure and model performance was implemented through a case study of data from the development of four consecutive releases of a very large legacy telecommunications software system (denoted as LLTS). To our knowledge this is the first study to group these seven feature selection techniques together and examine both stability and classification model performance of feature rankers in the software engineering domain.

Based on our results, we recommend the use of the S2N feature ranker, as this ranker was found to have both the greatest stability and highest classification performance across all levels of perturbation (for stability) and number of features selected (for both stability and classification performance). The WTS ranker sometimes showed slightly greater stability at the smallest feature subset sizes and greater classification performance at larger feature subset sizes, although this does not change our recommendation. Interestingly, FCR sometimes showed greater stability at larger feature subset sizes, but had the worst classification performance across the board. Conversely, SAM had the worst stability for all but one level of perturbation, but was generally within 0.01 AUC (Area Under the Receiver Operator Characteristic (ROC) Curve) of the top performer in terms of classification. Also, FCD had the top classification performance for one of the three learners, but showed moderate to poor stability. These last three results demonstrate that ranker performance for stability and classification are not necessarily linked, and that these must be examined separately to find a ranker which works well in terms of both.

The rest of the paper is organized as follows. We review relevant

literature in Section II. Section III provides detailed information about the seven FOS feature selection techniques. Section IV describes the datasets used in the study, presents stability results and analysis, and shows model performance results and analysis. Finally, in Section V, the conclusions are summarized and suggestions are made for future work.

II. Related Work

Feature selection is the process of choosing a subset of features (software metrics in our study) in order to reduce dimensionality and potentially improve classification performance. Feature selection can be broadly classified as *feature ranking* and *feature subset selection*. Feature ranking sorts the attributes according to their individual predictive power, while feature subset selection finds subsets of attributes that collectively have good predictive power. Feature selection techniques can also be categorized as *filters* and *wrappers*. Filters are algorithms in which a feature subset is selected without involving any learning algorithm. Wrappers are algorithms that use feedback from a classification learning algorithm to determine which feature(s) to include in building a classification model. In this study, we focused on filter-based feature ranking techniques, because although these can have reduced performance, they also avoid the larger computational cost associated with wrappers and feature subset evaluation [2].

A number of works have evaluated the use of feature selection and its role as part of a larger data mining and machine learning process. Guyon and Elisseeff [4] 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 [5] 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 [1] provided a comprehensive survey of feature selection algorithms and presented an integrated approach to intelligent feature selection. Collectively, these have shown that feature selection is an important preprocessing step prior to building a classification model.

The stability (robustness) of a feature selection method is normally defined as the degree of agreement between its outputs when applied to randomly selected subsets of the same input data [6]. To assess robustness of feature selection techniques, past works have used different similarity measures, such as Hamming distance [7], correlation coefficient [8], consistency index [6], and entropy [9]. Among these four similarity measures, consistency index is the only one which takes into consideration bias due to chance. Because of this, in our work the consistency index was used as stability measure. The term consistency index was defined by Kuncheva et al. [6]. The consistency index is a measure of similarity between two different feature subsets.

III. First Order Statistics (FOS) based Feature Selection

In this paper, we present seven related univariate feature selection metrics: Signal-to-Noise (S2N), Fold Change Difference (FCD), Fold Change Ratio (FCR), Welch T-Statistic (WTS), Wilcoxon Rank Sum (WRS), Fisher Score (FS), and Significance Analysis of Microarrays (SAM). Because all seven are based on first order statistics such as mean and standard deviation, we collectively refer to these as First Order Statistics (FOS) based feature selection. Although some of these metrics have previously been studied, no previous work in the domain of software quality modeling has collected these into a single family and examined their behavior and performance relative to one another. As these seven metrics are less often used in the context of feature selection, we wrote our own implementation in the WEKA data mining toolkit.

Each of these techniques work specifically with binary or two-class datasets. For the datasets in this study, the positive class P (that is, the class of interest) is fault-prone modules, which in this case is also the minority class (class with fewer instances). The negative class N is thus the majority class, containing the not-fault-prone instances. Each of the seven techniques will be described below.

Signal-to-noise is a measure used in electrical engineering to quantify how much a signal has been corrupted by noise. It is defined as the ratio of signal's power to the noise's power corrupting the signal. The signal-to-noise (S2N) ratio can also be used as feature ranking method [10]. For a binary class problem, the equation for signal to noise is:

$$S2N = \frac{\mu_P - \mu_N}{\sigma_P + \sigma_N} \tag{1}$$

where μ_P and μ_N are the mean values of that particular attribute in all of the instances which belong to a specific class, either P or N (the positive and negative classes). σ_P and σ_N are the standard deviations of that particular attribute as it relates to the two classes, respectively. If one attribute's expression in one class is quite different from its expression in the other, and there is little variation within the two classes, then the attribute is predictive. The larger the S2N ratio, the more relevant a feature is to the dataset.

Fold Change Difference (FCD) and Fold Change Ratio (FCR) [11] both use the mean value of the attribute across all instances in the positive class and the mean value of the attribute across all instances in the negative class. The difference between the two techniques is how they use these two measurements. FCD takes the difference between the mean of the attribute for the positive class and the mean of the attribute for the negative class. FCR uses the ratio between the mean of the attribute for the positive class and the mean of the attribute for the negative class.

The Welch T-Statistic [12] (WTS) is a modified version of the t-statistic which does not assume equal variance with each of the classes. The modified equation is shown here:

$$WTS = \frac{\mu_P - \mu_N}{\sqrt{\frac{\sigma_P^2}{n_P} + \frac{\sigma_N^2}{n_N}}} \tag{2}$$

where n_P and n_N are the number of instances in the positive and negative classes respectively.

Wilcoxon Rank Sum [13] (WRS) is different from the standard t-statistic in that it makes no assumptions on whether or not the distribution is normal. The first step is to rank the all of the instances based on the value of the attribute. The next step is to take the sum of all of the rankings in the positive class which we will denote as W_P. Finally, the WRS is found as follows:

$$WRS = \frac{(W_P - \frac{n_P(n_P+1)}{2}) - \frac{n_P n_N}{2}}{\sqrt{\frac{n_P n_N (n_P+n_N+1)}{12}}} \tag{3}$$

Fisher score [14] (FS) is a feature selection technique that selects each attribute independently according to their scores under the Fisher criterion. The FS is calculated as:

$$FS = \frac{n_P[\mu_P - \mu_T]^2 + n_N[\mu_N - \mu_T]^2}{\sigma_T^2} \tag{4}$$

The variance σ_T^2 is the variance of the attribute across instances from both classes collectively. Lastly, μ_T is the average of the attribute over all instances.

TABLE I
SOFTWARE DATASETS CHARACTERISTICS

	Data	#Metrics	#Modules	*%fp*	*%nfp*
LLTS	SP1	42	3649	6.28%	93.72%
	SP2	42	3981	4.75%	95.25%
	SP3	42	3541	1.33%	98.67%
	SP4	42	3978	2.31%	97.69%

Significance Analysis of Microarrays [12] (SAM) is a statistical technique borrowed from the domain of bioinformatics which determines whether changes in attribute value (gene expression in the bioinformatics application domain) are statistically significant. The SAM technique identifies relevant attributes by performing attribute-specific t-tests for each feature that measure the strength of the correlation between each independent feature and the class attribute. The equation of SAM is:

$$SAM = \frac{\mu_P - \mu_N}{\sigma^* + \sigma_0} \tag{5}$$

where σ_0 represents the exchangeability constant and σ^* represents an overall standard deviation. The role of σ_0 is to prevent attributes whose standard deviations are small from having large score. σ_0 is a customizable factor which is generally the top 90th-percentile of standard deviations. For this experiment we use this value. σ is calculated as:

$$\sigma^* = \sqrt{\frac{n_T}{n_P n_N (n_T - 2)} \Big(\sum_{j=1}^{n_P} [x_j - \mu_P]^2 + \sum_{j=1}^{n_N} [x_j - \mu_N]^2 \Big)} \tag{6}$$

where $\sum_{j=1}^{n_P}$ and $\sum_{j=1}^{n_N}$ represent the sum across the instances of the positive class and the instances of the negative class respectively. Additionally n_T is equal to the total number of instances in the dataset.

IV. EXPERIMENTS

A. Datasets

The experiments were conducted to discover model performance and the robustness (stability) of seven rankers and the impact of dataset perturbation. Experiments conducted in this study used software metrics and defect data collected from a real-world software project, a very large legacy telecommunications software system (denoted as LLTS) [15]. LLTS contains data from four consecutive releases, which are labeled as SP1, SP2, SP3, and SP4. The software measurement datasets consist of 42 software metrics, including 24 product metrics, 14 process metrics, and four execution metrics. The dependent variable is the class of the program module, fault-prone (*fp*), or not fault-prone (*nfp*). A program module with one or more faults is defined as *fp*, and *nfp* otherwise. Table I lists the characteristics of the four release datasets utilized in this work. An important characteristic of these datasets is that they all suffer from class imbalance, where the proportion of *fp* modules is much lower than that of *nfp* modules.

B. Experiments - Stability

In order to test the stability of seven feature selection techniques we employed four different software metrics datasets. With each feature ranker and dataset combination, we used four different levels of dataset perturbation along with four different numbers of features chosen.

1) Dataset Perturbation: In this study, we consider stability based on changes to the datasets (perturbations) at the instance level, produced as follows: We chose a fraction c of instances to keep and randomly removed $1 - c$ of the instances from both the majority and minority classes separately, where c is greater than 0 and less than 1. We removed from each class instead of just from the dataset as a whole in order to maintain the original level of class balance/imbalance for each dataset. For each c this process was repeated thirty times giving us thirty new datasets for each original dataset and level of *c*, with each new dataset having $c \times m$ instances (m being the number of instances in the original dataset) and each new dataset being unique, having been generated independently of the others. In this study, c was set to 0.95, 0.9, 0.8, or 0.67. In total, 4 datasets $\times 4$ levels of perturbation $\times 30$ repetitions $= 480$ datasets are generated.

2) Feature Selection: For each dataset and feature ranking technique, the features are ranked first according to their relevance to the class. The rankings are applied to each combination of dataset and level of perturbation. Therefore, (4 datasets $\times$ 7 feature rankers) + (4 datasets $\times$ 4 levels of perturbation $\times$ 30 repetitions $\times$ 7 feature rankers) = 3388 different rankings were computed. Then a subset consisting of the most relevant ones (top k features) is selected. In this study, four subsets are chosen for each dataset. The number of features that is retained in each subset for each dataset are 3, 4, 5, and 6. These numbers were deemed reasonable after some preliminary experimentation conducted on the corresponding datasets [16].

3) Stability Measure: Consistency index [6] is used to measure the degree of stability between the two rankings from the same ranker from different datasets (in our case perturbed vs. original). In this paper, we use the consistency index because it takes into consideration bias due to chance. We compute the consistency index between two feature subsets as follows. Let T_i and T_j be subsets of features, where $|T_i| = |T_j| = k$. The consistency index [6] is obtained as follows:

$$I_C(T_i, T_j) = \frac{dn - k^2}{k(n - k)}, \tag{7}$$

where n is the total number of features in the dataset, d is the cardinality of the intersection between subsets T_i and T_j, and $-1 < I_C(T_i, T_j) \leq +1$. If the two rankings are the same, then the consistency index is 1. The greater the consistency index, the more similar the subsets are.

For all choices of dataset, feature ranker, feature subset size, and level of perturbation, we found the consistency index for comparing the feature subsets from each reduced dataset individually with the feature subset from the complete original dataset. Thus, 30 comparisons were made since there are 30 repetitions for each choice of dataset, feature ranker, feature subset size, and degree of perturbation.

4) Results: The experiment was conducted using seven feature selection techniques on four real-world software datasets. When we apply the consistency index to compare feature subsets chosen from the original dataset and one of the derivative datasets, we can use the resulting consistency measurement to show the stability of the feature ranker. Table II contains the average results of our stability experiments across all four datasets, keeping the ranker, subset size, and perturbation level static. The top value at each column is in **boldface** and the smallest value is in *italicized* print.

In general, it can be observed that S2N shows the most stability, followed by FS, and SAM shows the least stability. Beyond these overall results, there are two main factors which can be examined to observe their effects on stability: degree of perturbation and number

TABLE II
AVERAGE STABILITY ACROSS FOUR DATASETS

Ranker	Perturbation Level: 0.67				Perturbation Level: 0.8				Perturbation Level: 0.9				Perturbation Level: 0.95			
	Number of Features				Number of Features				Number of Features				Number of Features			
	3	4	5	6	3	4	5	6	3	4	5	6	3	4	5	6
S2N	0.7906	**0.7951**	**0.8127**	0.7829	**0.8803**	**0.8503**	**0.8638**	0.8461	0.9162	**0.9171**	**0.9243**	**0.9157**	0.9581	**0.9586**	**0.9470**	**0.9336**
FCD	0.7577	0.6592	0.6235	0.6500	0.8115	0.7260	0.7105	0.7440	0.8774	0.8089	0.7862	0.8201	0.9132	0.8526	0.8411	0.8752
FCR	0.7607	0.7513	0.7976	**0.8056**	0.7966	0.8020	0.8751	**0.8671**	0.8594	0.8618	0.9168	0.8979	0.8923	0.8826	0.9414	0.9222
WTS	**0.8085**	0.7490	0.7332	0.7164	0.8714	0.8135	0.8014	0.8120	**0.9192**	0.8849	0.8732	0.8752	**0.9761**	0.9148	0.9016	0.9190
WRS	0.6620	0.6592	0.6424	0.6938	0.7218	0.7076	0.7162	0.7569	0.7308	0.7513	0.7635	0.8007	*0.7876*	*0.8112*	*0.8051*	*0.8606*
FS	0.7368	0.7651	0.7673	0.7813	0.8295	0.8388	0.8184	0.8574	0.8983	0.8664	0.8581	0.8963	0.9252	0.9125	0.8789	0.9319
SAM	*0.4526*	*0.3783*	*0.3530*	*0.3567*	*0.4915*	*0.4059*	*0.3814*	*0.3924*	*0.5154*	*0.4359*	*0.4286*	*0.4572*	0.9162	0.9010	0.9054	0.9530

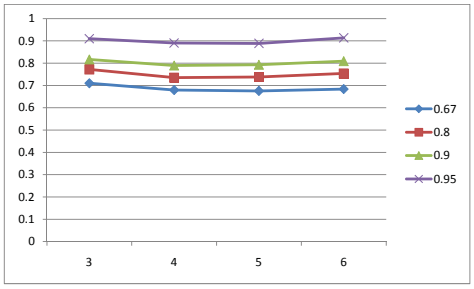


Fig. 1. Degree of Perturbation's Impact on Stability

of features used. Figure 1 shows the effect of the degree of dataset perturbation and feature subset size on the stability of feature ranking techniques across all four datasets and all seven rankers. From this figure, we can observe that the more instances retained in a dataset (e.g., the fewer instances deleted from the original dataset), the more stable the feature ranking on that dataset will be. This leads us to state that after enough change any feature selection method becomes unstable. The second trend visible in this figure is that in general, stability increases as more features are used.

To validate these results, we performed an ANalysis Of VAriance (ANOVA) [17] test to statistically examine the robustness (e.g., stability) of feature selection techniques. 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 then use another test to determine which factors or combinations of factors are associated with the difference. In this study, we performed a one-way ANOVA model with the single factor being the choice among the seven rankers. For this ANOVA test, we only considered the values with a feature subset size of four, and the results from all four datasets were taken into account together. The ANOVA model can be used to test the hypothesis that the stability for all rankers 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, a multiple comparison test using Tukey's Honestly Significant Difference (HSD) criterion [17] can be used to determine which levels (e.g., feature rankers) have statistically-significantly different means. All tests of statistical significance in this study utilize a significance level α of 5%, and all tests were implemented in MATLAB.

The ANOVA results are presented in Table III. The p value for the Ranker factor is 0, which indicates there was a significant difference between the average I_C values of the seven rankers. Thus, at least two rankers are significantly different from one another. To find out which rankers exhibit such a difference, Tukey's HSD criterion was used for multiple comparison analysis.

The multiple comparison results are presented in Figure 2, displaying graphs with each group mean represented by a symbol (○) and the 95% confidence interval as a line around the symbol. Two

TABLE III
ANALYSIS OF VARIANCE FOR STABILITY RESULTS

Source	Sum Sq.	d.f.	Mean Sq.	F	p-value
Ranker	40.657	6	6.77619	156.72	0
Error	144.977	3353	0.04324		
Total	185.634	3359			

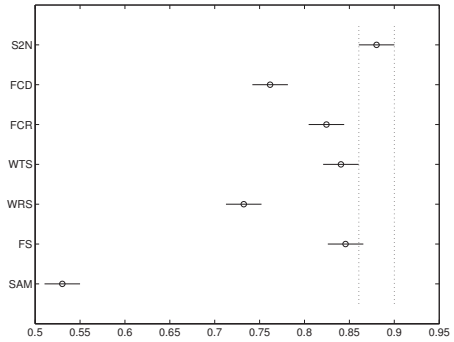


Fig. 2. Tukey's HSD: Ranker – Stability

means are significantly different if their intervals are disjoint, and are not significantly different if their intervals overlap. Results show that we can divide the seven rankers into four groups, {SAM}, {WRS, FCD}, {FCR, WFS, FS}, {S2N}, with the groups ordered by their performances from worst to best. The rankers from different groups produced significantly different results, while the rankers from the same group performed similarly (were not significantly different from one another). These results confirm our earlier observations about the stability of the S2N ranker.

C. Experiments - Defect Prediction Model Performance

To evaluate a feature selection technique, the stability performance of the feature selection may not be enough: we must also consider classification model performance. The goal of software defect prediction models is to improve the fault prediction and risk assessment process. Finding faulty components in a software system can lead to a more reliable final system and reduce development and maintenance costs.

1) Classifiers: In this study, software defect prediction models are built with three well-known classification algorithms: naïve Bayes (NB), support vector machine (SVM), and logistic regression (LR). All three learners themselves do not have a built-in feature selection capability and are commonly used in the software engineering and other data mining applications. All classifiers were implemented in the WEKA tool [18]. In this study, we used the default parameter settings for the different learners as specified in WEKA unless otherwise specified.

1) Naïve Bayes classifier (NB) utilizes Bayes's rule of conditional probability and is termed 'naïve' because it assumes conditional independence of the features.

TABLE IV
AVERAGE CLASSIFICATION PERFORMANCE ACROSS FOUR DATASETS

Ranker	Learner: NB				Learner: SVM				Learner: LR			
	Number of Features				Number of Features				Number of Features			
	3	4	5	6	3	4	5	6	3	4	5	6
S2N	**0.8129**	**0.8119**	0.8085	0.8083	**0.7154**	**0.6936**	**0.6856**	0.6691	0.8217	0.8241	0.8247	0.8264
FCD	0.8081	0.8091	0.8088	0.8090	0.7013	0.6777	0.6803	0.6574	**0.8274**	**0.8295**	**0.8307**	**0.8307**
FCR	*0.6526*	*0.6855*	*0.7128*	*0.7366*	*0.6075*	*0.6171*	*0.6305*	*0.6352*	*0.6382*	*0.6763*	*0.7094*	*0.7440*
WTS	0.8072	0.8088	**0.8105**	**0.8110**	0.7033	0.6749	0.6725	**0.6726**	0.8255	0.8291	0.8289	0.8300
WRS	0.7928	0.7921	0.7896	0.7878	0.7030	0.6907	0.6734	0.6610	0.8087	0.8129	0.8141	0.8133
FS	0.7982	0.8012	0.8033	0.8056	0.6887	0.6739	0.6727	0.6636	0.8185	0.8184	0.8171	0.8193
SAM	0.8056	0.8059	0.8040	0.8060	0.7060	0.6848	0.6753	0.6679	0.8243	0.8276	0.8276	0.8289

2) Support Vector Machine (SVM), also called SMO in WEKA [18], had two changes to the default parameters: the 'complexity constant c' was set to 5.0 and 'build Logistic Models' was set to true. By default, a linear kernel was used.
3) Logistic Regression (LR) is a statistical regression model for categorical prediction by fitting data to a logistic curve.

2) Performance Metric: Because traditional performance measures such as F-measure and overall classification accuracy are inappropriate to use on imbalanced data, we use the Area Under the ROC (Receiver Operating Characteristic) Curve metric. The ROC curve plots how the true positive rate and false positive rate vary as the decision threshold is changed, and the area under this curve shows how the model performs across all thresholds. This value can vary from 0 to 1, with 1 being a perfect model.

When evaluating models, we employed a process known as cross-validation. This involves dividing the data up into N folds, building a model on $N-1$ of these folds, and testing the model on the Nth fold. This process is then repeated until each fold is used as the test fold once. Cross-validation avoids the problem of overfitting by ensuring that the model is not tested on the same data used to build the model, while making sure that each instance gets to be used as test data. Note that in this study, feature selection was performed inside the cross-validation step (that is, on the training folds separately).

3) Results: During the experiments, ten runs of five-fold cross-validation were performed. First, we ranked the attributes using the seven rankers separately. Once the attributes are ranked, the top K attributes are selected to yield the final training data. After feature selection, we applied the classifier to the training datasets with the selected features, and then we used AUC to evaluate the performance of the classification model. In total, 7 rankers $\times$ 4 datasets $\times$ 10 runs $\times$ 5 folds = 1400 combinations of feature ranking techniques were employed, and correspondingly 1400 rankings $\times$ 4 feature subsets $\times$ 3 classifiers = 16800 classification models were built.

All the results are reported in Table IV. Note that each value presented in the table is the average over the ten runs of five-fold cross-validation outcomes across all four datasets. As with the stability table, the highest value in each column is presented in **boldface**, and the lowest value is in *italics*. From the table, we can observe that although a given ranker might perform best in combination with one learner, this may not be true when other learners are used to evaluate models. For example, FCD performed best, on average, in terms of AUC when the LR classifier are used. However, this is not true when the SVM classifier is used; in that case, S2N performed best. Using the NB classifier, S2N was best for smaller feature subset sizes while WTS was best for larger feature subset sizes. Thus, the results demonstrate that although no particular ranker dominates the others, we can conclude that S2N and FCD are most often the best techniques, while FCR is rarely optimal. For the NB and SVM learner, the best model is built with three features selected by the S2N ranker. For the LR learner, the best model is built with five or six features selected by the FCD ranker. In addition, LR performed significantly better than the other two learners and SVM performed worst. Our recent study [3] shows that the reduced feature subsets can have better or similar prediction performance compared to the complete set of attributes (original data set).

TABLE V
ANALYSIS OF VARIANCE FOR CLASSIFICATION RESULTS

Source	Sum Sq.	d.f.	Mean Sq.	F	p-value
Ranker	1.78952	6	0.29825	153.5	0
Learner	1.90347	2	0.95174	489.83	0
Ranker $\times$ Learner	0.07663	12	0.00639	3.29	0.0001
Error	1.59131	819	0.00194		
Total	5.36093	839			

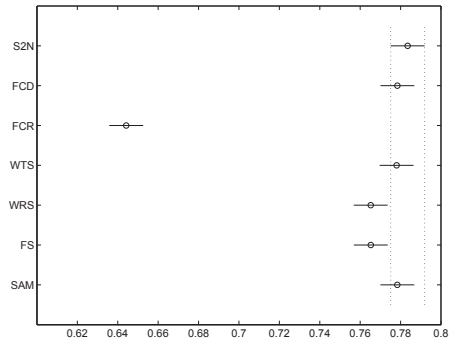


Fig. 3. Tukey's HSD: Ranker – Classification

As with the stability experiments, we used ANOVA and multiple comparison tests to evaluate the statistical power of our results. Here, two-way tests were employed, with the first factor being the choice of ranker and the second factor being the choice of learner; the response variable was the classification performance. Table V presents the ANOVA results, and we can see that for both factors individually and for the two-way interaction term, the p-value is less than 0.05. This indicates that for each factor, there are at least two levels (values of the given factor) which have statistically-significantly different means. In addition, the significance of the interaction term means that these two factors do not operate fully independently, but rather that the change in level for one factor will influence how the change in the other factor's levels affect classification performance.

To investigate which specific levels resulted in these statistically-significant differences, we performed a multiple comparison test on both factors. Due to space limitations, we only present Figure 3, the multiple comparison results for the different rankers. As we can see from this figure, the majority of rankers other than FCR produce similar results, although S2N is the clear winner; FCD, WTS, and SAM form a nearly-indistinguishable group following S2N, and WRS and FS follow behind this group. Only S2N shows a statistically-significant difference, however, and even here it is only different from WRS and FS. Outside of this group, FCR is notable for having results significantly worse than the other rankers; not only are these results

statistically significant, they show a difference of greater than 0.1 AUC. Overall, these results confirm what we observed from Table IV: S2N is the best ranker overall, although FCD and WTS also perform well (SAM is in this group, but never is the top performer), while FCR is much worse than the other rankers.

V. Conclusion

Feature selection is an important preprocessing step when preparing software quality prediction models. In this study, we investigate seven First Order Statistics (FOS) based feature selection techniques (Signal-to-Noise, Fold Change Difference, Fold Change Ratio, Welch T Statistic, Wilcoxon Rank Sum, Fisher Score, and SAM) in terms of stability and model performance in the context of software measurement data. The experiments were conducted on four consecutive releases of a very large telecommunications software system. The key conclusions are summarized as follows:

1) We tested the stability (robustness) of the rankers on four different perturbation levels and four feature subset sizes. We find that S2N is the most stable ranker regardless of level of perturbation and feature subset size, followed by FS, WTS, and FCR. SAM is the least stable ranker. Results also show that, in general, the bigger the feature subset size, the more stable the rankers are. Also, the number of instances deleted from the dataset affects the stability of the feature ranking techniques: the fewer instances removed from (or equivalently, added to) a given dataset, the less the selected features will change when compared to the original dataset, and thus the feature ranking performed on this dataset will be more stable. Our work provides guidance to software practitioners as to which feature selection methods are more stable. This is also useful for other domains in which feature selection is used for preprocessing data.
2) We also built classification models with the selected feature subsets using three different learners. The classification accuracy is evaluated in terms of the AUC performance metric. The experimental results demonstrate that the S2N ranker performed well in terms of robustness and was among the best rankers in terms of model performance. The experiments demonstrate that in our case study, on average, three software metrics are sufficient to build effective software defect prediction models. This is an important fact for software practitioners, since practitioners prefer using fewer software metrics for data collection, management, comprehension, and modeling. We also note that, using a specific classification algorithm can yield different results.
3) Comparing the results from our stability and classification studies, we see that although some rankers perform well in terms of both metrics, others only perform well for one or the other. S2N is a member of this first group, and is the best ranker for both metrics. WTS was similar, showing good performance in terms of both and sometimes beating S2N in certain scenarios. SAM and FCD are more unusual, though, in that while SAM is the worst ranker in terms of stability and FCD is the third-worst ranker, both are statistically indistinguishable from S2N in terms of classification performance. Likewise, FCR is the absolute worst ranker in terms of classification performance, but its stability was only middling compared with the other rankers. These results show that if the goal is to find rankers which perform well in terms of both stability and classification performance, both tests must be run separately, because the results of one do not necessarily predict the results of another.

Future work may include experiments using additional datasets from other software engineering projects, and experiments with other ranking techniques and classifiers for building software quality classification models.

References

[1] 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.

[2] 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, Nevada, August 13-15 2007, pp. 667–672.

[3] K. Gao, T. M. Khoshgoftaar, and A. Napolitano, "Exploring software quality classification with a wrapper-based feature ranking technique," in *Proceedings of 21st IEEE International Conference on Tools with Artificial Intelligence*, Newark, NJ, USA, Nov. 2-5 2009, pp. 67–74.

[4] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of Machine Learning Research*, vol. 3, pp. 1157–1182, March 2003.

[5] 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.

[6] L. I. Kuncheva, "A stability index for feature selection," in *Proceedings of the 25th conference on Proceedings of the 25th IASTED International Multi-Conference: artificial intelligence and applications*. Anaheim, CA, USA: ACTA Press, 2007, pp. 390–395.

[7] K. Dunne, P. Cunningham, and F. Azuaje, "Solutions to Instability Problems with Sequential Wrapper-Based Approaches To Feature Selection," Department of Computer Science, Trinity College, Dublin, Ireland, Tech. Rep. TCD-CD-2002-28, 2002.

[8] A. Kalousis, J. Prados, and M. Hilario, "Stability of feature selection algorithms: a study on high-dimensional spaces," *Knowledge and Information Systems*, vol. 12, no. 1, pp. 95–116, Dec. 2006.

[9] P. Křížek, J. Kittler, and V. Hlaváč, "Improving stability of feature selection methods," in *Proceedings of the 12th international conference on Computer analysis of images and patterns*, ser. CAIP'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 929–936.

[10] C.-H. Yang, C.-C. Huang, K.-C. Wu, and H.-Y. Chang, "A novel gataguchi-based feature selection method," in *IDEAL '08: Proceedings of the 9th International Conference on Intelligent Data Engineering and Automated Learning*, Berlin, Heidelberg, 2008, pp. 112–119.

[11] I. Jeffery, D. Higgins, and A. Culhane, "Comparison and evaluation of methods for generating differentially expressed gene lists from microarray data," *BMC Bioinformatics*, vol. 7, no. 1, pp. 359+, July 2006.

[12] V. G. Tusher, R. Tibshirani, and G. Chu, "Significance analysis of microarrays applied to the ionizing radiation response," *Proceedings of the National Academy of Sciences*, vol. 98, no. 9, pp. 5116–5121, 2001. [Online]. Available: http://www.pnas.org/content/98/9/5116.abstract

[13] R. Breitling and P. Herzyk, "Rank-based methods as a non-parametric alternative of the t-statistic for the analysis of biological microarray data." *Journal of bioinformatics and computational biology*, vol. 3, no. 5, pp. 1171–1189, oct 2005. [Online]. Available: http://view.ncbi.nlm.nih.gov/pubmed/16278953

[14] Q. Gu, Z. Li, and J. Han, "Generalized fisher score for feature selection," in *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*. AUAI Press, July 2011, pp. 266–273.

[15] K. Gao, T. M. Khoshgoftaar, H. Wang, and N. Seliya, "Choosing software metrics for defect prediction: an investigation on feature selection techniques," *Software: Practice and Experience*, vol. 41, no. 5, pp. 579–606, 2011.

[16] H. Wang, T. M. Khoshgoftaar, and N. Seliya, "How many software metrics should be selected for defect prediction?" in *Proceedings of the Twenty-Fourth International Florida Artificial Intelligence Research Society Conference*, May 2011, pp. 69–74.

[17] M. L. Berenson, M. Goldstein, and D. Levine, *Intermediate Statistical Methods and Applications: A Computer Package Approach*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 1983.

[18] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 3rd ed. Morgan Kaufmann, 2011.

Software Effort Estimation using Regularized Radial Basis Function Neural Networks

Khaled Shams
Faculty of Computers and Information
Cairo University

Haitham Hamza
Faculty of Computers and Information
Cairo University

Amr Kamel
Faculty of Computers and Information
Cairo University

Abstract— **The value of Artificial Neural Networks (ANNs) methods in performing complicated pattern recognition and nonlinear estimation tasks has been demonstrated across an impressive spectrum of applications. ANNs methods has been used extensively, in the software cost estimation process, due to the complexity of the relations between the project's attributes. ANNs Radial Basis Function (RBF) networks have advantages of easy design, and strong tolerance to input Noise. This paper, studies the effect of using Regularized Radial Basis Function Networks in improving the accuracy of the software cost estimation, using different training algorithms like K-means method, Genetic algorithm and Particle Swarm Optimizer (PSO). The relative improvements were found to be around 40 %, by using the Regularized Radial basis function with the PSO Algorithm.**

Keywords-*Software effort estimation; Estimation using ANN; Survey on software estimation, Genetic algorithm, Particle Swarm Optimizer*

I. INTRODUCTION

In software engineering, effort is defined as the total time that takes members of a software development team to perform a given task. It is usually expressed in units such as man-day, man-month, and man-year. This value is important as it serves as basis for estimating other values relevant for software projects, like Cost or Total time required to produce a software product. The term "Cost estimation" and "Effort estimation" are sometimes used interchangeably. This paper follows the assumption that the two terms are synonymous. Software effort estimation is different from software sizing. Sizing estimates, the probable size of a piece of software while effort estimation predicts the effort needed to build it. The relationship between the size of software and the effort is that, the size is one of the key attributes to estimate the required effort. For software effort estimation, there are several models developed, which can be grouped into two major categories:

(1) Parametric models, which are derived from the statistical or numerical analysis of historical projects data

(2) Non-parametric models, Non-parametric models differ from parametric models in that the model structure is not specified a priori but is instead determined from data.

In this paper, we will focus on Radial Basis Function, as one of the non-parametric model types, which is have high tolerance to noise, and study the effect of Regularized Radial basis function networks, and its training algorithms in Regularized neural network in improving software estimation accuracy.

II. ANNS IN SOFTWARE ESTIMATION

There are several ANNs methods that has been used in software estimation, bellow are the mostly common ANNs as per our survey:

1. Feed-forward neural network
2. Recurrent neural networks
3. Radial basis function (RBF) network
4. Neuro-fuzzy networks

A. Feed-forward neural network

The Feed-Forward is the first and the simplest type of artificial neural network. In these nets, neurons are arranged in layers, and there are only connections between neurons in one layer to the following. The feed-forward algorithm has several training algorithms, the popular ones are Backpropagation [1][14][23] and Marquardt algorithm [3].

A.1 Feed-forward neural backpropagation network in software estimation

The Feed-Forward multi-layer perceptron with back-propagation learning algorithm are the commonly used in the software cost estimation. In these nets, neurons are arranged in layers. The Feed-Forward backpropagation has been used in several studies [24, 25, 26, and 27]. Venkatachalam [24] constructed back-propagation network using the COCOMO model [19], and used 22 input nodes and 2 output nodes. The input nodes represent the distinguishing features of software projects and the output nodes represent the effort required in terms of person month and the development time required to complete the project. Venkatachalam has suggests that the neural network approach is a promising tool for accurately estimating software costs and development time, but Venkatachalam has not compared the performance of the back-propagation with other models. Samson [26] used also the COCOMO as Venkatachalam, and calculated the MMRE equal 428%. Wittig [25] used the ASMA database of 81 projects. Wittig calculated the MMRE to be equal 17%. Srinivasan [27] compared the performance of the back-propagation, with standard models of software development estimation using COCOMO's database of 63 projects. Srinivasan used 22 input units, 10 hidden units and 1 output unit. Srinivasan reported that the back-propagation has better MMRE (%), the parametric models

A.2 Feed-forward neural network using Levenberg-Marquardt in software estimation

The Levenberg-Marquardt (LM) algorithm [4] is an iterative technique that locates the minimum of a multivariate function that is expressed as the sum of squares of non-linear real-valued functions [5]. The Levenberg-Marquardt method is commonly used to solve nonlinear least squares problems, but it is rarely used in software estimation. As per our survey the, (LM) algorithm has only been used by Aggarwal [28], to estimate the development effort using ISBSG repository [50]. Aggarwal has compared the LM with back-propagation, and showed that the LM has better MMRE. The main problem with feed-forward algorithm, is the local minima, that is due to either data inadequacy or noise.

B. Recurrent Neural Networks

In Recurrent Network [21], additional to the Feed-Forward connections, units have self-connections or connections to units in the previous layers. This recurrency acts as a short-term memory and lets the network remember what happened in the past. This combines the advantage of the nonlinear approximation ability of a multilayer perceptron with the temporal representation ability of the recurrency, and such a network can be used to implement any of the three temporal association tasks, where the output sequence is given as output after a specific input sequence. The input and output sequences may be different. One of the most known recurrent neural networks is Elman neural network [21]. Typical Elman network has one hidden layer with delayed feedback. The Elman neural network is capable of providing the standard state-space representation for dynamic systems. Jagannath Singh [23] has compared the development Time (DT) between Elman Backpropagation Neural Network model and Feed Forward ANN using NASA dataset, Jagannath Singh, shows that layer recurrent neural network has the lowest MMRE.

C. Radial Basis Functions (RBF) neural network

RBF network are embedded in two layer neural network, where each hidden unit implements a radial activated function. The output units implement a weighted sum of hidden unit outputs. The input into an RBF network is nonlinear while the output is linear. Their outstanding approximations capabilities have been studied in [6].Due to their nonlinear approximation properties, RBF network are able to model complex mappings. A large variety of training algorithms has been tested for training RBF networks, like orthogonal least squares using Gram-Schmidt algorithm [7], Backpropagation, and learning vector quantization [8].

The RBF network, uses the bellow hypostasis $h(x)$ for the interpolation to estimate the development effort. The name Radial came from the fact that the following function is used for interpolation

$$h(x) = \sum_{i=1}^{N} w_i \, \varphi(||x - c_i||) \quad (1)$$

Where $\varphi(||x - x_i||)$ { i = 1, 2, ...,N} is a set of N arbitrary (nonlinear) functions known as Radial-Basis Functions.
The most common Radial Basis Function is the Gaussian function which takes the form

$$h(x) = \sum_{i=1}^{N} w_i \exp(-\gamma ||x - c_i||^2) \quad (2)$$

Where c_i is the center vector for neuron i, and γ is the width of the RBF function. The most common techniques to calculate the centers of hidden layer neurons and their width γ are K-means method, Orthogonal least squares, Genetic algorithm [51] and Particle Swarm Optimizer [16]. In this research, we will make comparisons between some of these techniques.

Shin and Goel [17], described a detailed Radial Basis Function modeling study for software cost estimation using NASA dataset. Shin and Goel indicated that, the problem of developing an RBF model can be seen as that of determining its parameters. Shin and Goel studied effect of standard deviation on the MMRE.

D. Neuro-fuzzy neural networks

Neuro-fuzzy refers to combinations of artificial neural networks and fuzzy logic. The Neuro-fuzzy synergizes these two the human-like reasoning style of fuzzy systems with the learning and connectionist structure of neural networks. Neuro-fuzzy combination is widely termed as Fuzzy Neural Network (FNN) or Neuro-Fuzzy System (NFS). Neuro-fuzzy hybridization [11], [12], [13] is done broadly in two ways: a neural network equipped with the capability of handling fuzzy information [termed fuzzy-neural network (FNN)] and a fuzzy system augmented by neural networks to enhance some of its characteristics like flexibility, speed, and adaptability [termed neural-fuzzy system (NFS)]

D-1 Neuro-Fuzzy Systems

Xishi Huang et al [9] developed a Neuro-Fuzzy algorithm (NFA) model for software cost estimation which uses the desirable features of a neuro-fuzzy approach, such as learning ability and good interpretability. In the NFA consists of the following Components:

- Pre-Processing Neuro-Fuzzy Inference System (PNFIS) used to resolve the effect of dependencies among contributing factors of the estimation problem, and to produce adjusted rating values for these factors.
- Neuro-Fuzzy Bank (NFB) used to calibrate the contributing factors by mapping the adjusted rating values for these factors to generate their corresponding numerical parameter values,
- Module that applies an algorithmic model relevant to the nature of the estimation problem to produce one or more output metrics.

In NFA Fuzzy logic is used to calibrate Function Points (FP) complexity degree to fit specific application. Neural network calibrates unadjusted FP weight values to reflect the current software industry trend by learning from ISBSG data. Xishi Huang et al has been checked his model against the COCOMO algorithmic model, and showed an improvement of 15%

D- 2 Fuzzy neural network

Chiu [29], proposes a Fuzzy Neural Network (FNN) approach for embedding artificial neural network into fuzzy inference processes in order to derive the software effort estimates. Chiu [29], used artificial neural network to determine the significant fuzzy rules. Empirical results by Chiu, showed that applying FNN for software effort estimates resulted in slightly smaller mean magnitude of relative error (MMRE) and probability of a project having a relative error of less than or equal to 0.25 as compared with the results obtained by just using artificial neural network and the original model.

Bvanss [31], and Xia [14] have suggested Fuzzy neural Networks, where the fuzzy logic part of the model calibrates the Function point (FP) complexity weights to fit the specific application context [2]. The neural network part of the model

calibrates the unadjusted FP [2] weight values to reflect the current software industry trend. Xia [14] has validated the empirical data repository (ISBSG Release 8). The experimental validation results show a 22% improvement in software cost estimation and demonstrate that FP need calibration and can be calibrated. The problem with Fuzzy neural network, that it does not consider the categorical environmental parameters like development platform (PC, Main Frame, Midrange).

III. Model Overview

This section gives an overview for our recently published model called "Enhanced Fuzzy Multiple Regression Neural Model for Software Effort Estimation" [51]. This section is followed by introducing enhancement techniques, to optimize the previously defined model.

A. Model components overview

As shown in figure 1, the model is composed of four building components, Data Preparation, Categorical Transformation, Fuzzy Regression - Data filter, and Function formulation components.

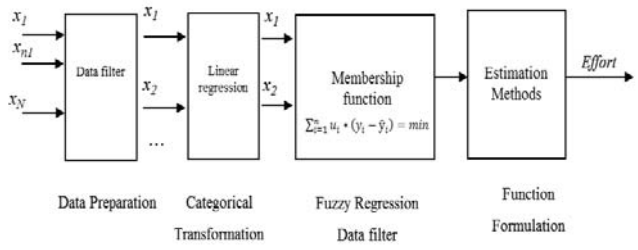


Figure 1: Block Diagram of Fuzzy Regression ANN model

- **Data Preparation – Building Component**

Data imperfections pose a problem for the estimation models, which rely on historical data. These imperfections can have unwanted impact on the effort estimation accuracy. One form of these imperfections are random errors or "noise". Noise refers to incorrect or erroneous values in the data. A common definition of an outlier is given by Barnett and Lewis [13] as, an instance that appears to be inconsistent with the remaining instances in the data. In our experiments we, use the ISBSG dataset, and use the following steps as preliminary noise removal: a) Only ISBSG data with quality = "A", are selected b) Remove records with missing data

- **Categorical Transformation – Building component**

There are a lot of environmental characteristics that cannot be ignored, in the software estimation process, like the Development Platform (Mainframe, Mid-Range, Personal Computers), Development Type (Enhancements, New development, New utility, Re-development) , Language Type(3GL, 4GL, APG), bur it would be meaningless to use these parameters in this form as a regression predictor because the value of the numbers does not reflect its weight during the estimation process. Thus, a transformation of the these categorical data is done based on binary encoding [44].

This process is equivalent to function of the Neuro-Fuzzy Bank in the sub-section D.

- **Fuzzy Regression Data Filter Building Component**

There are several filtering algorithms, to remove noise from the data, other than the preliminary noise detection illustrated in data preparation section like: K-means clustering (K-means) [55], least trimmed squares, Genetic Algorithm [50] and others.
In our model, we used the fuzzy regression as noise filter as a new technique, where the least square method is replaced by below membership function.

$$u_i = \frac{1}{\sigma\sqrt{2\pi}}\ e^{-\frac{1}{2}\left(\frac{y_i-\mu}{\sigma}\right)2} \qquad (5)$$

and the fuzzy least squares method [14] is represented by the following expression:

$$\sum_{i=1}^{n} u_i * (y_i - \hat{y}_i) = min \qquad (6)$$

Where: $u_i = \frac{1}{\sigma\sqrt{2\pi}}\ e^{-\frac{1}{2}\left(\frac{y_i-\mu}{\sigma}\right)2}$ (5)

- **Function Formulation**

We have tested several ANNs models, like Feed-Forward-back propagation and others, but it has been found from the experiments that the RBF is more tolerant to input noise. As a continuation, for our work, we have worked for further improving of estimation accuracy through the Regularization Mechanism [53].

IV. Model Optimization

In RBF network training for exact interpolation passing every data point may leads to over fitting and poor generalization. Regularization may improve the performance of network by imposing smoothness constraints, such as adding a weight penalty term to the sum-squared-error to construct cost function (7) bellow:

$$CF = \frac{1}{2}||y - h(x)||^2 + \frac{1}{2}\lambda\ ||w||^2 \ (7)$$

Where $||y - h(x)||^2$ is the squared residual norm; $||w||^2$ is regularized norm; λ is called the regularization parameter.
Regularization parameter λ suffers from a trade-off between the "size" of the regularized solution and the quantity of the fit that it provides to the given data. If λ is too small this, may cause overfitting. If is too large, the network will not adequately fit the training data. In this paper, we focus improving the estimation accuracy through regularization, to minimize the cost function (7), through the following steps: Differentiating the cost function with respected to weights W, equating the result to zero and rewriting it in compact form, which leads to the following equation (8) :

$$||y - h(x)||^2 = \lambda\ ||w||^2 \ (8)$$

Regularization parameter λ is optimal according to (8),

- **Optimization Model Construction**

Like other nonlinear networks, RBF networks face the same controversy to adjust it's parameter and the number of layers. Incorrect adjustments may lead either to unacceptable approximations or to expensive computation and may cause over fitting problem [9], [12], [38].
To study effect of RBF regularization, the same dataset (ISBSG) and tool - Matlab are used. To calculate the RBF network weight, the following Matlab function is used:

$$W = rbf(x, c, y, \sigma, \lambda) \quad (9)$$

Where:

x is the input data
c is the is the center vector for neuron, determined by the k-means
λ is regularization parameters

Implementation steps:

1. Tune σ with start with 1 then fine tune it
2. λ start 1 then fine tune

V. CASES STUDIES

A. *Effect of Regularization on MMRE*

In this case study, we will measure the effect of the regularization of the Radial basis function on the MMRE

Configuration:

Item	Value
No. of hidden layers	2
Radial basis function type	Generalized Radial Basis Function & Regularized
No. of data Records	640

MMRE at different stages

Model	MMRE
RBF with outliers (RBF)	0.4
RBF after removing the outliers (RBF-WO)	0.3
RBF after using the Fuzzy regression filter	0.25
RRBF (regularization)	0.1

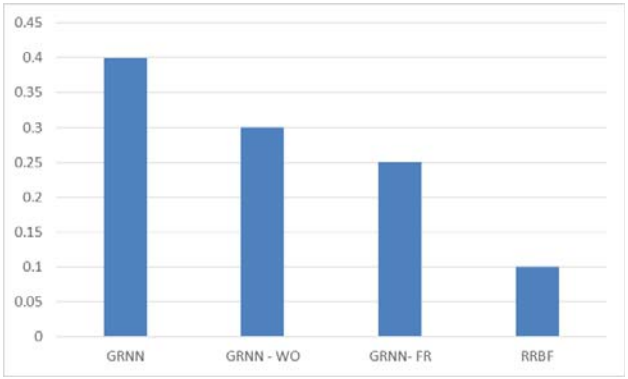


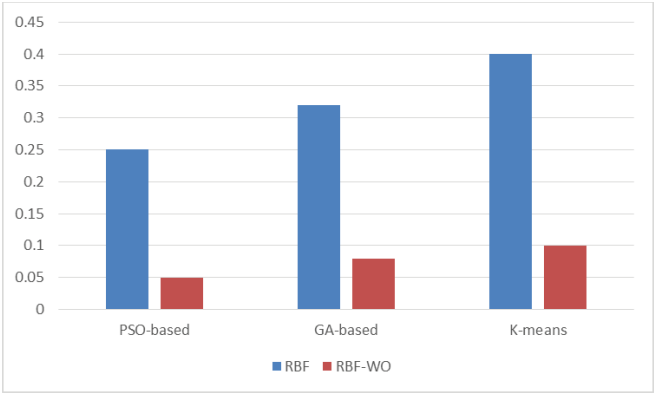
Figure 2: MMRE Comparison between different Regularized and non-regularized

Experiment 2.

B. *Comparision on different techniques to calucate the* neurons *centers and width*

In this case study, we examined the effect of the training algorithm, in the estimation accuracy. We have used the K-means method, Genetic algorithm and Particle Swarm Optimizer algorithms. The K-means was used in the previous case study

	PSO-based	GA-based	K-means
RBF (Regularized RBF with outliers)	0.25	0.32	0.4
RBF-WO (Regularized RBF without outliers)	0.05	0.08	0.1



VI. CONCLUTION

Radial basis function (RBF) networks have advantages of easy design and strong tolerance to input noise. Regularization improves network generalization ability by adding penalty term to original cost function, thus improving software estimation

accuracy MMRE. The RBF has embedded noise filer, which is the K-means. Regularization RBF is not easy and implement, and takes time for adjusting it, but it improves the accuracy in for high scale and noisy data. The paper is aimed to recommend Regularized RBF networks for large scale and noisy data. Choosing the right training algorithm and adjusting the model parameters are also important as choosing the right neural networks. The PSO has shown better results than the Genetic Algorithm and k-means clustering. As a continuation to this work, more training algorithms need to be tested like Orthogonal least squares and others.

REFERENCES

[1] Rumenhart D. E., G. E. Hinton and R. J. Wiliams, "Learning representations by back-propagating errors" Nature, vol. 323, pp. 533-536, 1986

[2] IFPUG, Function Point Counting Practices Manual, International Function Point Users Group, 2004

[3] M. T. Hagan, M. B. Menhaj, "Training Feedforward Networks with the Marquardt Algorithm," *IEEE Trans. on Neural Networks*, vol. 5, no. 6, pp. 989-993, Nov. 1994.

[4] Andersen, Thomas J. and B.M. Wilamowski, "A. Modified Regression Algorithm for Fast One Layer Neural Network

[5] D.W. Marquardt. An Algorithm for the Least-Squares Estimation of Nonlinear parameters. *SIAM Journal of Applied Mathematics*, 11(2):431–441, Jun.1963.

[6] J. Park, I. W. Sandberg, "Universal Approximation Using Radial-Basis-Function Networks" Neural Computation, Summer 1991, Vol. 3, No. 2 , Pages 246-257

[7] Chen, S., Cowan, C. F. N. and Grant, P. M. (1991) Orthogonal least squares learning algorithm for radial basis function networks. IEEE Transactions on Neural Networks, 2, (2), 302-309

[8] kohonen, T. K., (1989) self-orgainsation and associative memory. Berline: Springer-Verlag

[9] Ho, D., Capretz, L.F., Huang, X., Ren, J. **Neuro-Fuzzy Algorithmic (NFA) Models and Tools for Estimation**, *Twentieth International Forum on COCOMO and Software Cost Modeling*, University of Southern California, Los Angeles, October 2005.

[10] Horikawa, T Furuhashi, Y Uchikawa, "On fuzzy modeling using fuzzy neural networks with the back-propagation algorithm" IEEE Transactions on Neural Networks (1992), Volume: 3, Issue: 5, Pages: 801-806

[11] S. K. Pal and S. Mitra, *Neuro-fuzzy Pattern Recognition: Methods in Soft Computing*. New York: Wiley, 1999.

[12] C. T. Lin and C. S. George Lee, *Neural Fuzzy Systems—A Neuro–Fuzzy Synergism to Intelligent Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1996.

[13] L. X. Wang, *Adaptive Fuzzy Systems and Control*. Englewood Cliffs, NJ: Prentice-Hall, 1994.

[14] Wei Xia , Danny Ho2, .A Neuro-Fuzzy Model for Function Point Calibration, WSEAS Transactions on Information Science and Applications, Volume 5 Issue 1, January 2008

[15] Hao Yu; Tiantian Xie; Paszczynski, S.; Wilamowski, B.M."Advantages of Radial Basis Function Networks for Dynamic System Design" - IEEE Transactions on Industrial Electronics, Dec 2011

[16] Eberhart, R. C. & Kennedy, J. (1995). A new optimizer using particle swarm theory, Proceeding of 6th Int. Symp. Micro Machine and Human Science, pp. 39-43

[17] Idri, A. , Mbarki, S. ; Abran, A. "Validating and Understanding Software Cost Estimation Models based on Neural Networks" Information and Communication Technologies: From Theory to Applications, 2004. Proceedings.

[18] B.W. Boehm. Software Engineering Economics. Prentice- Hall, 1981

[19] Y-S Hwang and S-Y Bang. "An Efficient Method to Construct a Radial Basis Function Network Classifier." Neural Networks, vol. 10, no. 8, 1997, pp. 1495-1503

[20] He, Q. "Neural Network and Its Application in IR". University of Illinois, Urbana-Champaign 1999

[21] Elman, J.L., 1990 "Finding structure in time," Cognitive Science, Vol. 14, 179-211

[22] Jagannath Singh, "Software Effort Estimation with Different Artificial Neural Network", IJCA Special Issue on "2nd National Conference-Computing, Communication and Sensor Network" CCSN, 2011

[23] Venkatachalam, A.R "Software Cost Estimation Using Artificial Neural Networks", 1993. Proceedings of 1993 International Joint Conference on neural network, Nogoya

[24] Gerhard Wittig, Gavin Finnie "Estimating software development effort with connectionist models". Information and Software Technology (1997)

[25] Bill Samson, David Ellison "Software cost estimation using an Albus perceptron (CMAC)". Information and Software Technology. Volume 39, Issue 1, 1997, Pages 55‑60

[26] Srinivasan, K. Fisher, D. "Machine learning approaches to estimating software development effort" IEEE Transactions on Software Engineering Feb 1995, 21 , Issue: 2 . Page(s): 126 – 137

[27] Arthur; Abhishek Bhowmick (2009). "A theoretical analysis of Lloyd's algorithm for k-means clustering".

[28] Huang, S., Chiu, N., "Applying fuzzy neural network to estimate software development effort", in Proceedings of Applied Intelligence Journal, Vol. 30, No. 2, pp. 73-83 , Apr. 2009.

[29] Vivian Xia, Danny Ho "Calibrating Function Points Using Neuro-Fuzzy Technique". Information and Software Technology 50 (2008) 670–683

[30] Bvanss prabhakar Rao, P Seethe Ramaiah. IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 1, No 1, January 2012

[31] Idri, A. "Can neural networks be easily interpreted in software cost estimation?" Fuzzy Systems, 2002. FUZZ-IEEE'02. Proceedings of the 2002 IEEE International Conference on Date of Conference: 2002

[32] Saleem Basha "Analysis of Empirical Software Effort Estimation Models". (IJCSIS) International Journal of Computer Science and Information Security, Vol. 7, No. 3, 2010

[33] Chris Schofield, "Non-Algorithmic Effort Estimation Techniques" Department of Computing Bournemouth University Talbot Campus Poole, BH12 5BB England. Technical report 1998

[34] Moloekken-OEstvold, K. "A survey on software estimation in the Norwegian industry". Software Metrics, 2004. Proceedings. 10th International Symposium on

[35] Wolverton, R.W., The Cost of Developing Large-Scale Software, IEEE Transactions on Computers, Volume C-23, No 6, pp 615-636, June 1974

[36] Bente Anda, Hege Dreiem "Estimating Software Development Effort Based on Use Cases-Experiences from Industry". 01 Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools. Springer-Verlag London, UK 2001

[37] Sharareh Afsharian, Marco Giacomobono A framework for software project estimation based on cosmic, dsm and rework characterization Proceeding Proceedings of the 1st international workshop on Business impact of process improvements ACM New York, NY, USA ©2008

[38] M. van Genuchten and H. Koolen, "On the Use of Software Cost Models," Information & Management, vol. 21, pp. 37-44, 1991.

[39] Heemstra, F.J., Software cost estimation. Information and Software Technology, 1992. 34(10): p. 627-639.

[40] Kaczmarek J, Kucharski M, "Size and Effort Estimation for Applications Written in Java," Journal of Information and Software Technology, Volume 46, Issue 9, pp 589-60, 2004

[41] Heiat A, "Comparison of Artificial Neural Network and Regression Models for Estimating Software Development Effort," Journal of Information and Software Technology, Volume 44, Issue 15, pp 911-922, 2002

[42] K. Srinivasan and D. Fisher, "Machine learning approaches to estimating software development effort," IEEE Transactions on Software Engineering, vol. 21, pp. 126-137, 1995.

[43] Musilek, P., Pedrycz, W., Succi, G., Reformat, M., Software Cost Estimation with Fuzzy Models, Applied Computing Review, Vol.8, No.2, 2000, pp.24-29

[44] N. Fenton, S. Pfleeger, "Software Measurement: A rigorous and practical approach", International Thomson Computer Press, 1996.

[45] Manpreet Kaur "Using Bayesian regulation back propagation

algorithm for Software effort estimation and comparing with COCOMO IRACST – Engineering Science, and Technology: An International Journal (ESTIJ), ISSN: 2250-3498, Vol.2, No. 4, August 2012

[46] G. H. Subramanian, P. C. Pendharkar, and M. Wallace, "An Empirical Study of the Effect of Complexity, Platform, and Program Type on Software Development Effort of Business Applications," Empirical Software Engineering, vol. 11, pp. 541-553, 2006.

[47] S. Kumar, B. A. Krishna, and P. S. Satsangi, "Fuzzy systems and neural networks in software engineering project management,"

Journal of Applied Intelligence, vol. 4, pp. 31-52, 1994.

[48] Bhatnagar, R.; Ghose, M.K.; Bhattacharjee, V., "A novel approach to the Early Stage Software Development Effort Estimations using Neural Network Models: a Case Study"; Artificial Intelligence Techniques - Novel Approaches & Practical Applications" of International Journal of Computer Applications (USA), Number 3 - Article 5, 2011 pp 27-30.

[49] About ISBSG, International Software Benchmarking Standards Group, 2004.

[50] Yunfei, B. & Zhang, L. (2002). Genetic algorithm based self-growing training for RBF neural Network, *IEEE Neural Networks*, Vol. 1, pp. 840-845

[51] Khaled Shams, Amr Kamel "Enhanced Fuzzy Multiple Regression Neural Model for Software Effort Estimation" DMS 2012 The 18th International Conference on Distributed Multimedia Systems Knowledge Systems Institute Graduate School, USA

[52] Federico Girosi, michael Jones "Regularization Theory and Neural Network Architectures" . Artfical intellegence Laboratory Massachusetts Institute of Technology, Cambridge AI Memo 1430 1993.

[53] C. R. VOGEL, *Computational Methods for Inverse Problems*, SIAM, Philadelphia, PA, 2002.

[54] Dempster, er, A.P.; Laird, N.M.; Rubin, D.B. (1977). "Maximum Likelihood from Incomplete Data via the EM Algorithm". Journal of the Royal Statistical Society. Series B (Methodological) 39 (1): 1–38.

[55] Moddy, Y. & Darken, C. J. (1989). Fast learning in network of locally tuned processingunites, Neural computation, Vol.1, pp. 281-294

Towards a Unified Framework for Measuring the Properties of Class Diagrams Augmented with OCL

Mohamed Elshaarawy
Computer Engineering dept.
AAST
Cairo, Egypt

Haitham S.Hamza
Information Technology dept.
FCI, Cairo University
Cairo, Egypt

Ismail Taha
Vice President,
AAST
Cairo, Egypt

Abstract*—*There are several studies in the literature to assess the properties of a class diagram without taking into consideration the added complexity due to using object constraint language (OCL). This paper presents integrated frameworks that consider class diagram augmented with rules and constraints embedded in the problem domain and encoded using OCL to investigate the benefit and/or impact of using OCL. The proposed framework is theoretically and empirically validated.

Keywords-component; Software measurement, Class diagram, OCL, Metric

I. INTRODUCTION

A great effort has been made in the field of software measurement for the following; (1) assessing the properties of class diagrams without considering the OCL (Chidamber & Kemerer, Li & Henry, Brito e Abreu & Carapuça, Lorenz & Kidd, Briand, Marchesi , Harrison, Bansiya, Genero, In, Rufai , Zhou, Kang and others [1], (2) measuring the structural properties of OCL expressions in isolation, without considering class diagrams on which these OCL are used, (Reynosa, Genero, Piattini [2], Cabot and Teniente [3]).

In response to the great demand for metrics for measuring the quality characteristics of class diagrams augmented with OCL as a one chunk we have proposed a unified framework contains a set of "Integrated" metrics for that purpose, a set of new class diagram metrics, a set of new OCL metrics in additions to the well-known class diagram and OCL metrics that have been selected for their extensively use, relevant scope and validity.

The rest of this paper is organized as follows. In section 2 a brief overview of the proposed framework has been presented, Section 3 presents a brief overview of the empirical validation of the proposed metrics. In section 4 some concluding remarks and future work are presented.

II. THE PROPOSED FRAMEWORK

A. Philosophy of the Proposed Framework

The proposed framework follows some suggestions provided by Briand about "how to define valid measures" [4]. Also, it follows the following principles: Metrics must be defined pursuing clear goals (using for example the GQM method), Metrics must be theoretically and empirically validated. And we have followed what Fenton [5] suggested that it is not advisable to define a single measure for capturing different structural properties. For that reason we have defined a set of metrics, each of which captures different structural properties of class diagram augmented with OCL.

The goal pursued for the definition of the metrics is measuring the structural properties of class diagrams with its OCL to find early indicators of such qualities' external attributes such as functionality, reliability, usability, efficiency, maintainability and portability.

The structural properties will be measured are: size, length and coupling. And, the scopes of measures are: class diagram, OCL expressions and class diagram augmented with OCL (integrated metrics).

So, the proposed metrics structured as follows: three metrics suites (size measurement metrics suite, length measurement metrics suite and coupling measurement metrics suite). Each suite consists of class diagram scope, OCL expressions scope and integrated class diagram _OCL metrics.

The acronyms for the metrics indicate what will be counted: The first two letters indicates the scope of the metrics (Cd: The class diagram alone and Oc: The OCL expressions).The remainder of the letters indicates the goal of the metrics (NC: Number of classes, NA: Number of attributes…), while the integrated metrics will be expressed by natural language like (Classes density).

B. Size Measurement Metric Suite

1) Class Diagram Size Scope Metrics:

*a) The selected well known (existing literature) metrics are [1]:*CdNC(Class diagram number of classes), *CdNM* (Class diagram number of methods), *CdNAss* (Class diagram number of association relationships) for uni directional and bi directional association only, *CdNAgg* (Class diagram no. of aggregation relationships), *CdNDep* (Class diagram number of dependency relationships), *CdNGen* (Class diagram number of generalization relationships), *CdNGenH* (Class diagram number of generalization hierarchies), *CdNAggH* (Class diagram number of aggregation hierarchies).

b) The proposed metrics are: CdNA (Class diagram number of attributes), *CdNR* (Class diagram no. of relations) it can be calculated by: *CdNR =CdNAss+ CdNAgg+ CdNDep+ CdNGen+ CdNGenH+ CdNAggH*

2) OCL Expressions Size Scope Metrics:

a) *The selected well known (existing literature) metrics are : OcNoBj* **(**the no. of objects in Bexp where Bexp is the bag of objects accessed during the evaluation of an expression) [5], *OcWNM* (OCL expressions Weighted No. of Messages), *OcWNN* (OCL expressions Weighted No. of Navigations) [6]

b) *The proposed metrics are: OcNL* (OCL expressionsNo. of Lines),*OcNC*(OCL expressions No. of constraints),*OcNI* (OCL expressions No. of Invariants),*OcNO* (OCL expressions No. of Operations) Including operations Boolean type, operations on integer and real types, operations on string types, operations on collection types and query operation, *OcNCond* **(**OCL expressions No. of pre and post Conditions), *OcNV* **(**OCL expressions No. of Variables defined through <<Definition>> constraints included number of variables defined by Let expressions in an expression), *OcNaA* (OCL expressions No. of accessed class diagram attributes),*OcNaM* (OCL expressions No. of accessed class diagram Methods), *OcNaR* (OCL expressions No. of accessed class diagram Relations), *OcNaC* (OCL expressions No. of accessed classes)

3) Integrated Size Scope Metrics:

a) *Classes' Density:* The ratio of (total no. of accessed classes in OCL expressions /total no. of classes within class diagram= OcNaC/CdNC) this represents the extra complexity added to CdNC due to using OCL.

b) *Attributes' Density:* The ratio of (total no. of accessed attributes In OCL expressions /total no. of attributes within class diagram= OcNaA/CdNA) this represents the extra complexity added to CdNA due to using OCL.

c) *Methods' Density:* The ratio of (total no. of accessed method. In OCL expressions /total no. of methods within class diagram= OcNaM/CdNM) this represent the extra complexity added to CdNM (no. of methods) due to using OCL.

d) *Relations' Density:* The ratio of (total no. of accessed relations In OCL expressions /total no. of relations within class diagram=OcNaR/CdNR) this represents the extra complexity added to CdNR due to using OCL.

4) Theoritcal validation:

Following the property based framework of Briand [4], it has been assured that all of the size metrics suite fulfill all of the following axioms that characterize size metrics:

PropertySize.1 (Non-negativity): This property is directly proven because it is impossible to obtain a negative value for all of the size metrics suites.

PropertySize.2 (Null value): Any system from the above described three systems have no elements will lead to its related metrics =0.

PropertySize.3 (Module Additivity): This property is directly proven because the size of a system is equal to the sum of the sizes of its modules.

C. Length Measurement Metric Suite

1) Class Diagram Length Scope Metrics:

a) *The selected well known (existing literature): CdMaxDIT* **(**The maximum between the Depth of Inheritance DIT values obtained for each class of the class diagram, The DIT value for a class within a generalization hierarchy is the length of the longest path from the class to the root of the hierarchy) ,*CdMaxHAgg* (The maximum between the Hierarchical Aggregation (HAgg) values obtained for each class of the class diagram ,The HAgg value for a class within an aggregation hierarchy is the length of the longest path from the class to the leaves)Units.[1]

2) OCL Expressions Length Scope Metrics:

a) *The selected well known (existing literature): OcDN* : Depth of Navigations (DN) is the maximum depth of the navigation tree [6].

b) *The proposed metric is:* the added length to the longest path from the class to others related classes in the class diagram due to navigation tree build after using OCL**.**

3) Theoritcal validation:

Following the property based framework of Briand [4], it has been assured that all of the length metrics suite fulfill all of the following axioms that characterize length metrics:

PropertyLength.1 (Nonnegativity) and Null Value are straightforwardly satisfied, the depth of a tree can never be negative, and an expression without navigation has an empty tree.

PropertyLength.2 (Nonincreasing monotonocity)for connected components: If we add relationships between elements of a tree (classes or interfaces) the depth of the tree does not vary.

PropertyLength.3 (Nondecreasing monotonocity) for non-connected components: Adding a relationship to two unconnected components (two trees) makes them connected, and its length is not less than the length of the two unconnected components.

PropertyLength.4 (Disjoint modules): The depth of a tree is given by the component that has more levels from the root to the leaves.

D. Coupling Measurement Metric Suite

1) *Class Diagram Coupling Scope Metrics:*

CdNPCAIC (Number of coupling relationship to Parent classes if there is a Class-attribute interaction between classes of Import coupling type) , *CdNOCAIC* (Number of coupling relationship to other classes if there is a Class-attribute interaction between classes of Import coupling type), *CdNDCAEC* (Number of coupling relationship to descendant classes if there is a Class-attribute interaction between classes of Export coupling type), *CdNOCAEC* (Number of coupling

relationship to other classes if there is a Class-attribute interaction between classes of Export coupling type), *CdNPCMIC* (Number of coupling relationship to parent classes if there is a Class-Method interaction between classes of Import coupling type),*CdNOCMIC* (Number of coupling relationship to other classes if there is a Class-Method interaction between classes of Import coupling type), *CdNDCMEC* (Number of coupling relationship to descendant classes if there is a Class-Method interaction between classes of Export coupling type),*CdNOCMEC* (Number of coupling relationship to other classes if there is a Class-Method interaction between classes of Export coupling type)[4].

2) *OCL Expressions Coupling Scope Metrics:*

OcNPT (The OCL expressions Number of Parameters whose types are classes defined in the class diagram),*OcNAN* (The OCL expressions Number of attributes referred through navigations in an expression, *OcWNM* (OCL expressions Weighted No. of Messages), *OcWNO* (The OCL expressions Weighted Number of referred Operations through navigations)[2].

3) Theoritcal validation:

Following the property based framework of Briand [4], it has been assured that all of the coupling metrics suite fulfill all of the following axioms that characterize coupling metrics:

PropertyCoupling.1 (Non-negativity): Is directly proven, and it is impossible to obtain a negative value for all of stated metrics.

PropertyCoupling.2 (Monotonicity): Is directly verified, adding import interactions - in this case, DU-interactions (interaction from Data declaration to data Used in an OCL expression)

PropertyCoupling.3 **(**Merging Modules): This property can be expressed for our context in the following way: "the sum of the import coupling of two modules is no less than the coupling of the module which is composed of the data used of the two modules". The value of metrics for an expression which consists of the union of two original expressions, is equal to the metrics of each expression merged when the sets of classes, methods, attributes and messages referred to in each original expression are disjointed, otherwise is less than metrics of each merged expression.

III. THE EMPERICAL VALIDATION SUPPORTED CONTROLLED EXPERIMENT

A. Experiments Planning

1) *Definition:*

The purpose of this experiment is to analyze UML class diagrams with its OCL expressions proposed framework metrics and to evaluate the metrics results with the subjects' ratings results indicating the impact of using OCL, combined with UML class diagrams on the understandability and maintainability degree.

2) *Context & Material Selection:*

The context of the experiment is a group related to the area of Software Engineering in different seniority levels. They were grouped into 4 groups (8 subjects per group) and each subject worked individually on each of the two systems, using class diagram+OCL in one case and only class diagram in the other. They performed activities over 2 laboratories, as well as the order of the activities, are summarized in Table 1.

Two comparable, in terms of complexity and size, class diagrams with their OCL expressions representing (Royal& Loyal system and Hotel management system) are used.

3) Hypothesis formulation:

The impact of using OCL will be assessed on two dependent variables: Understandability (U) and Maintainability (M) were measured according to the subjects' ratings (of the experiment's designed questionnaires) targets the system logic(U), and the ability of subjects to correctly specify the affected system elements due to changes in requirements(M). The other dependent variable is Maintenance time which is comprises the time to comprehend the class diagram, analyze the required changes and to implement them.

The null hypothesis (H0): There is no significant correlation between the framework metrics and the above variables.

The alternative hypothesis (Ha): There is a significant correlation between the framework metrics and the above variables.

TABLE I. THE EXPERIMENT LAYOUT

	Group 1	**Group 2**	**Group 3**	**Group 4**
(Lab1)	R&L (no-ocl)	R&L (ocl)	Hmgmt (no-ocl)	Hmgmt (ocl)
(Lab2)	Hmgmt (ocl)	Hmgmt (no-ocl)	R&L (ocl)	R&L (no-ocl)

4) Instrumentation:

Besides the class diagrams with its OCL expressions separate documents, other forms were used like a survey questionnaire was distributed to obtain information about the subjects' background. After the completion of both Understandability and Maintenance tasks, forms were distributed that contains specific questions about the system.

Maintenance questions address the Maintainability variable. Figure 2 shows an example

Figure 1. Example Maintainace question

The hotel has decided to make a group reservation for the attendees of the conference room and this will be an option to either reserve the conference room or not.
a) Which class (es) should be added and/or modified?
b) Which modifications should be applied (additions, change in attributes, operations) must be made in these class (es).

All of the questions were covered different parts of the system, feasible to answer but not trivial and could be answered with or without OCL.

In addition, another after-lab questionnaire was distributed

with questions addresses the selected variables.Those questions were answered on a Likert scale from 1 (Strongly agree) to 5 (Strongly disagree).

5) *Execution and Data Validation:*

The operations were done successfully and all the data have been collected including subjects' answers obtained from the responses of the experiment and the metrics value.

6) Analysis and interpretation

The data collected have been summarized and the metrics values have been calculated to each class diagram with its OCL. A simple statistical test (e.g., t-test) would do for that purpose as the factor which has only two levels. Because most of the subjects performed the same task with and without OCL, the paired t-test can be used and gain additional statistical power by removing individual differences. The ordering effects which are impossible to avoid (e.g., learning effects) should be considered and determine whether it interferes with the presented analysis. The data of each Lab may have to be analyzed independently.
As the ability of subjects has a strong effect on the task results, so in order to refine the data analysis, a two-way ANOVA is used considering Ability (Experience) as a factor as well.

All of the data have been collected were analyzed and tested separately per each lab, per each attempt, per each variable considering the tool (with or without OCL), subjects ability and the ordering. The median, sum of squares, F-Ratio and others were calculated to conduct the required testing's and making the required comparisons.

After applying the Kolmogrov-Smirnov test the data were non-normal a nonparametric test like Spearman's correlation coefficient was used, with a level of significance $\alpha = 0.05$, which means the level of confidence is 95%. Using Spearman's correlation coefficient, each of the metrics was correlated separately to the median of the subject's rating of the three variables.
The computed Spearman's correlation coefficients are above the cutoff, and the p-value < 0.01, the null hypothesis H0, is rejected. Hence, there is a significant correlation between the framework metrics and subject's rating of the dependent variables.

7) Threats to validity:

External validity threats like materials and tasks used. In the experiment we tried to use class diagrams which can be representative of real cases. Related to the tasks, the judgments of the subjects are to some extent subjective.
In terms of internal validity, we have prevented any confounding effect due to the system being used, the ability of subjects and the order of performing the tasks.
To overcome the Fatigue effects. On average the experiment lasted for less than one hour in each attempt (this fact was corroborated summing the total time for each subject), so fatigue was not very relevant. Also, the different order in the tests helped to cancel out these effects.
In order to avoid persistence effects, the experiment was run with subjects who had never done a similar experiment.

IV. CONCLUSION AND FUTURE WORK

This paper shows how we defined a metric suite for measuring the overall Structural Properties of a class diagram augmented with OCL. Also, it shows the best way from our point of view to measure the Structural Properties of a class diagram and the OCL expressions individually. We have added the best metrics in the literature- in every scope- to provide the best framework. We validated our framework theoretically by using the property-based framework of Briand et al. and classified it to size, length and interaction-based metrics for coupling.

A controlled experiment was done to validate the proposed framework empirically and shows that there are correlations between the metrics and their tested variables. And also shows that OCL, as a constraint language complementary to UML class diagrams, is useful in helping understand system models facilitate change and understanding. Obviously that the benefits of using OCL may only be observed if subjects have sufficient training and/or obtained sufficient experience in UML class diagrams and OCL.

The extension of the proposed framework work to cover other UML diagrams is very important to make a unified framework for measuring the overall structural properties of UML diagrams which augmented with OCL.

REFERENCES

[1] M. Genero, M Piattini, C. Calero: "A Survey of Metrics for UML Class Diagrams", in Journal of Object Technology, vol. 4, no. 9, November December 2005, pp. 59-92

[2] Reynoso, L., Genero, M. and Piattini, M.: Towards a metric suite for OCL Expressions expressed within UML/OCL models. Journal of Computer Science and Technology, 4 (1). 38-44, 2004

[3] Cabot, J. and Teniente, E.: A metric for measuring the complexity of OCL, MODELS'06, Genova, Italy, 2006

[4] Briand, L., Morasca, S. and Basili, V. : Defining and validating measures for object-based high level design, In: IEEE Transactions on Software Engineering. 25(5) pp. 722-743, 1999

[5] Fenton, N. and Pfleeger, S.: Software Metrics: A Rigorous Approach. 2nd.edition. Chapman & Hall, London 1997

[6] Reynoso, L., Genero, M. and Piattini, M.: Measuring OCL expressions: a "tracing"-based approach. In: Workshop on Quantitative Approaches in Object-Oriented Software Engineering. QAOOSE'2003, Germany, 2003

Assessing RBFN-Based Software Cost Estimation Models

Ali Idri, Aya Hassani
Department of Software Engineering
ENSIAS, Mohammed V –Souissi University
Rabat, Morocco
idri@ensias.ma, aya.hassani@gmail.com

Alain Abran
Department of Software Engineering
Ecole de Technologie Supérieure
Montréal, Canada, H3C IK3
Alain.abran@etsmtl.ca

***Abstract*— This paper is concerned with the design of the neural networks approach, especially Radial Basis Function Neural (RBFN) networks, for software effort estimation models. The study firstly focuses on the construction of the RBFN middle layer composed of receptive fields, using two clustering techniques: hard C-means and fuzzy C-means. Thereafter, we evaluate and compare the performance of effort estimation models that use an RBFN construction-based either on hard or fuzzy C-means. This study uses the ISBSG dataset and confirms the usefulness of an RBFN-based on fuzzy C-means for software effort estimation.**

Keywords—Software cost estimation; RBF Neural Networks; Hard and Fuzzy C-means; ISBSG dataset.

I. INTRODUCTION

Effort or cost estimation of the software project is one of the most important activities in software project management. To improve estimates and avoid gross mis-estimation, several cost estimation techniques have been developed. Jørgensen and Shepperd carried out a systematic review of software cost estimation studies [1], identifying up to 11 estimation techniques in 304 software cost estimation selected papers. Those techniques may be grouped into two major categories: parametric models, which are derived from the statistical or numerical analysis of historical projects data [2], and non-parametric models, which are based on a set of artificial intelligence techniques such as artificial neural networks [3,4], case-based reasoning [5] and decision trees [6]. Non-parametric models have received increasing attention from researchers; the systemic literature review of Wen et al. [7] has identified eight types of machine learning techniques employed in 84 selected studies. In particular, the CBR and ANN based cost estimation models are the most frequently used, with 37% and 26% respectively. Moreover, their study showed that the CBR and ANN are more accurate, in terms of Pred and MMRE, than the other machine learning techniques (Pred(25)=64% and MMRE=37% for ANN, Pred(25)=46% and MMRE=51% for CBR). This paper is concerned with the neural networks approach, more specifically the Radial Basis Function Neural (RBFN) networks for software effort estimation models.

In our earlier works [4,8], two crisp clustering techniques, APCIII and C-means, were empirically evaluated to design the middle layer of RBFN for software effort estimation. These studies used two datasets: the COCOMO'81 [2] and Tukutuku [9]. It has been illustrated with these two datasets that the RBFN designed with C-means performs better, in term of effort estimates accuracy, than the RBFN designed with APC-III algorithm. In addition, the results showed that the accuracy of RBFN construction-based on C-means depends also on the classification used in the hidden layer. For instance, the classification obtained by minimizing the objective function J leads to better estimates than that one obtained by maximizing the Dunn index D1.

Thereafter, the fuzzy C-means clustering technique has been applied in the design of RBFNs to estimate the development effort of web hypermedia applications [10]. The study used data on web applications from the Tukutuku dataset which contains 53 web projects described using 9 numerical attributes. The aim of the study was to evaluate and to compare the accuracy of an RBFN construction-based on fuzzy C-means with that one of an RBFN construction-based on hard C-means. The results showed that an RBFN using fuzzy C-means performs better, in terms of accuracy and tolerance of imprecision, than an RBFN using hard C-means; however, the importance of these findings is limited by the fact that it has been proven that the accuracy of any estimation technique, for instance RBFN, is highly depended on the characteristics of the used dataset, especially its sample size [1].

This paper revisits these RBFN studies [10] using a larger dataset, that is, the International Software Benchmarking Standards Group repository- ISBSG release 8 [11] which contains more than 2000 software projects described by several attributes such as the number of function points, the development platform and the programming language used in the project. This study uses a set of cases and attributes selection criteria to retain 151 projects described by 6 attributes. More specifically, the following research questions were investigated: 1) Can RBFN networks be useful for software effort estimation? 2) Will an RBFN construction-based on fuzzy C-means tolerates imprecision and uncertainty when dealing with numerical values? 3) Will an RBFN construction-based on fuzzy C-means performs much better, in term of MMRE and PRED, than that of an RBFN construction-based on C-means?

The remainder of the paper is organized as follows. Section 2 presents the dataset description and evaluation criteria. Section 3 describes how hard and fuzzy C-means are incorporated in the construction of the hidden layer of the

RBFN networks. Section 4 discusses the results of this study on the larger dataset. Finally, Section 5 discusses the findings.

II. DATA DESCRIPTION AND EVALUATION CRITERIA

This section describes the data used to perform this study and presents the evaluation criteria used to compare the performance of the designed models.

A. Data Description

The data used in this study come from the International Software Benchmarking Standards Group ISBSG release 8 [11].This ISBSG repository is a multi-organizational dataset containing more than 2000 projects gathered from different organizations in different countries. Major contributors come from Australia (21%), Japan (20%) and United States (18%). To decide on the number of software projects as well as their descriptions, a data pre-processing phase is conducted. The objective of this preprocessing phase is to select data (projects and attributes) in order to retain projects with high quality data for this study. It consists of three steps.

Step 1: Projects selection

This step consists on the selection of the software projects with high quality data according to the following four criteria:

- Data Quality Rating field that contains an ISBSG rating code of A, B, C or D applied to the project data by the ISBSG quality reviewers [11]. This code denotes the soundness and the integrity of the data of each project. The selected projects for this study are those with rating code A or B.
- Ressource levels indicates the type of the data collected about the people whose time is included in the work effort data reported. Four levels are identified in the data collection instrument: "1=development team effort. 2=development team support. 3=computer operations involvement. 4=end users or clients" [11].The selected projects are those with rating code 1 or 2: indeed, in the litterature of software effort estimation, the development effort ony includes the efforts spent for the activities of the development team as well as for its support.
- Unadjusted Function Points Rating indicates an ISBSG rating code of A, B, C or D applied to the unadjusted function point count data by the ISBSG quality reviewers [11]. This code denotes the soundness and the integrity of the UFP count data. The selected projects for this study are those with code rating A or B.
- Development Type describes whether the development was a new development, enhancement or re-development [11]. The selected projects are the new development because this paper deals with software development effort.

TABLE I. DATA QUALITY CRITERIA FOR PROJECTS SELECTION

Criteria	Selected Values	Discarded Values
Data Quality Rating	A or B	C and D
Resource Levels	1 or 2	3 and 4
Unadjusted Function Points Rating	A or B	C or D
Development Type	New Development	Enhancement and Redevelopment

Table I summarizes the quality criteria that had to be met for the projects pre-selection. A total of 151 projects that satisfied all conditions are selected for this study.

Step 2: Attributes Selection

The aim is to identify the attributes to be used as the inputs (effort drivers) of a effort estimation model based on an RBFN network. Selection attributes describing adequately software projects is a complex task. The challenge is to detect a subset of attributes that are relevant (exhibiting a significant relationship with the effort), independent (to avoid multiple uses of an attribute), operational (easy to measure), and comprehensive (well defined). When the number of available attributes is small, a brute force search of all possible subsets may be applied; otherwise, it becomes an NP-hard search and consequently is not a feasible solution. This is the case of the ISBSG dataset in which the projects are described with more than 50 numerical and linguistic attributes. Hence, the solution adopted is to allow the estimators to use the attributes that they believe best characterize their projects and are more appropriate in their environment. Moreover, the use of hard or fuzzy C-means requires numerical attributes. Consequently, the selected attributes must be numerical. Six attributes have been selected since they were usually considered in the literature as relevant cost drivers. Table II contains the descriptions of these six attributes.

TABLE II. SELECTED SOFTWARE ATTRIBUTES OF THE ISBSG REPOSITORY

Software Attribute	Description
Value Adjustment Factor (VAF)	The adjustment of the function points
Unadjusted Function Points (UFP)	The unadjusted function point count
Max Team Size	The maximum number of people that worked at any time on the project
User Base - Business Units	Number of business units that the system services
User Base – Locations	Number of physical locations being serviced by the installed system.
User Base - Concurrent Users	Number of users using the system concurrently.

Step 3: Data normalization

Due to the nature of selected software attributes, some of continuous attributes show a larger range of values than others which may make the effect of these attributes too important. The solution is to scale continuous attributes into the same range. To achieve this, all continuous attributes are normalized applying the min-max normalization formula of the Equation (1) such that all numerical variables are scaled within the range of [0, 1].

$$x_i(k) = \frac{x_i(k) - \min\,(x(k))}{\max\big(x(k)\big) - \min\,(x(k))} \qquad (1)$$

B. Evaluation criteria

The accuracy of the estimates generated by the RBFN is evaluated by the criteria of the Mean Magnitude of Relative Error (MRE) and the Prediction level which are defined by Equation (2) and Equation (3) respectively.

$$MMRE = \frac{1}{N}\sum_{i=1}^{N}\left|\left(\frac{Effort_{i,actual} - Effort_{i,estimated}}{Effort_{i,actual}}\right)\right| \quad (2)$$

$$Pred\ (L) = \frac{k}{N} \quad (3)$$

Where N is the total number of observations, K is the number of observations with MRE less or equal to L. A common value for L is 25%, which is used in the present study. The Pred(25) represents the percentage of projects which MRE is less or equal to 25%. The Pred(25) value identifies the software effort estimates that are generally accurate whereas the MMRE is fairly conservative with a bias against overestimates.

III. RBFN NETWORK CONSTRUCTION

The use of an RBFN to estimate software development effort requires the determination of its architecture parameters according to the characteristics of the historical software projects, especially the number of input neurons, number of hidden neurons, centers cj, widths σj and weights βj. Figure 1 illustrates a possible RBFN architecture configured for software development effort estimation.

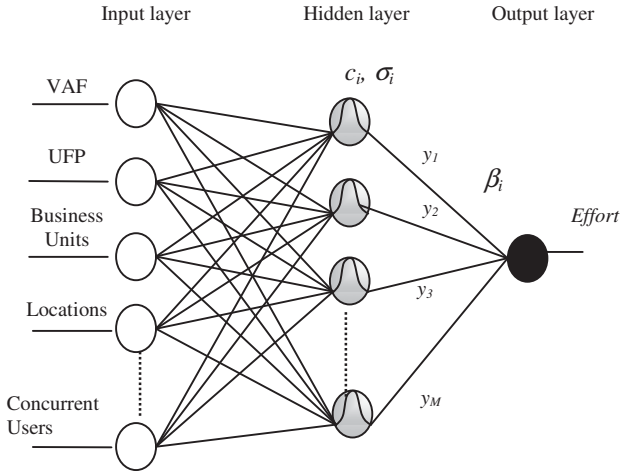


Figure 1. An RBFN network architecture for software development effort estimation.

The number of the input neurons is, usually and simply, the number of the attributes describing the historical software projects in the used dataset. Therefore, when applying RBF networks to ISBSG dataset the number of input neurons is equal to 6.

The construction of the hidden layer of the proposed RBFN networks is achieved using two algorithms of hard and fuzzy clustering techniques. Hard clustering methods are based on classical set theory, and require that an object either does or does not belong to a cluster whereas fuzzy clustering methods allow the objects to belong to several clusters simultaneously, with different degrees of membership. The role of clustering in the design of RBFN is to set up an initial distribution of receptive fields (hidden neurons) across the input space of the input variables (effort drivers). In this work, the two clustering algorithms used in designing RBFN are hard C-means and fuzzy C-means algorithms [12,13].

For the widths σ_j, many heuristics and techniques based on solid mathematical concepts have been proposed in the literature [14,15,16]. The main idea is to determine (σ_j) in order to cover the input space as uniformly as possible [17]. For instance, two alternatives are considered [18,19]. Based on our previous results [10] in which accurate estimates were obtained when using the formula defined by Haykin [19], we adopt this solution which consists in assigning one value to all (σ_j).

Concerning the weights βj, we may set each βj to the associated effort of the center of the j^{th} neuron. However, this technique is not optimal and does not take into account the overlapping that may exist between receptive fields of the hidden layer. Thus, we use the learning Delta rule to derive the values of βj.

IV. OVERVIEW OF EMPIRICAL RESULTS

This section presents and discusses the results obtained when applying an RBFN network using either the hard C-means or the fuzzy C-means to the sample from the ISBSG dataset. The calculations were made using two software prototypes developed with the programming language C under a Microsoft Windows PC environment. The first prototype implements hard and fuzzy C-means clustering algorithms, providing both the clusters and their centers from the ISBSG dataset. The second software implements an effort estimation model based on an RBFN architecture where the middle layer parameters were already determined by the first software prototype.

To decide on the number of hidden units, several tests have been performed with both hard C-means and fuzzy C-means algorithms. Choosing the best classification to determine the number of hidden neurons and their centers is not an obvious task. For software effort estimation, we suggest that the best classification is the one that provides coherent clusters which have satisfactory degrees of similarity, and improve the accuracy of estimates, in terms of MMRE and Pred(25), produced by the RBFN.

A. Evaluation of RBFN-based on hard C-means

To measure the coherence of clusters, the objective function J, rather than the Dunn's validity index, is used. Indeed, in our previous experiments, the use of J had led to accurate estimates when applying an RBFN based on C-means to the COCOMO'81 and the Tukutuku Datasets [2,9]. Several experiments have been conducted with an RBFN using the objective function J as a validity index for each number of clusters c. The classification that minimizes J is taken. These experiments use the ISBSG dataset for training and testing.

Figure 2 illustrates the relationship between the accuracy (MMRE and Pred) of an RBFN architecture and the number of

clusters generated by the hard C-means. We can notice that the accuracy in terms of the Pred(25) criterion is, in general, monotonous increasing according to c whereas it is monotonous decreasing according to c when the MMRE criterion is used. Hence, when c is higher than 134, the estimates accuracy is higher (Pred(25)>70 and MMRE <30). This is normal because when the number of clusters is higher (tends towards 151) the clusters obtained are more coherent; in particular if c is equal to 151, each cluster contains one project. However, the aim is to reduce the number of clusters c and to keep the estimates accuracy acceptable. Consequently, we are concerned with the accuracy when the values of c tend towards zero. It is noticed that when c is lower than 133, the RBFN estimates accuracy in terms of Pred and MMRE is not acceptable (Pred(25)<70% and MMRE>30%). We conclude that the threshold value of c to use an RBFN based on C-means for the ISBSG dataset is around 133. This is very high considering that the maximum possible value is 151 clusters. It may be due to the fact that software projects in the ISBSG dataset are not sufficiently similar, taking into consideration their description with the six attributes of Table II.

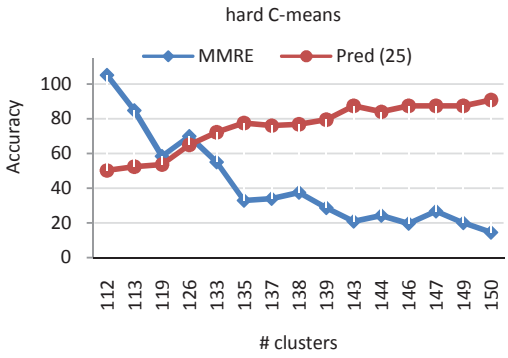


Figure 2. Relationship between the accuracy (MMRE and Pred) of an RBFN construction-based on hard C-means and the number of clusters used in the hard C-means, c.

B. *Evaluation of RBFN-based fuzzy C-Means*

To measure the coherence of clusters, the Xie-Beni validity criterion is used [20]. A small value of XB means a more compact and separate clustering. The goal should therefore be to minimize the value of XB. Several tests have been conducted with an RBFN-fuzzy C-means using the XB as a validity index for each number of clusters C. The classification that minimizes XB is taken. These experiments use the ISBSG dataset for training and testing.

Figure 3 illustrates the relationship between the accuracy (Pred and MMRE) of RBFN architecture and the number of clusters c generated by the fuzzy C-means. We can notice that the accuracy using the Pred(25) criterion is, in general, monotonous increasing according to c whereas it is monotonous decreasing according to c when the MMRE criterion is used. Hence, when c is higher than 127, the estimates accuracy in terms of Pred(25) is in general higher (Pred(25)>70).

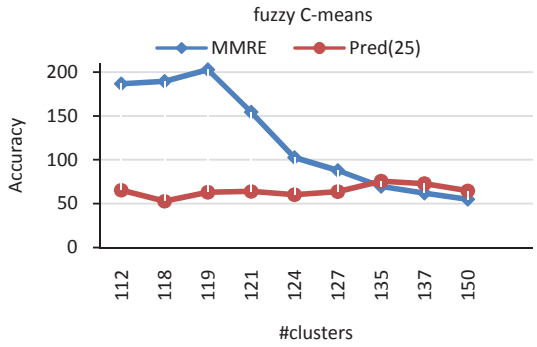


Figure 3. Relationship between the accuracy (MMRE and Pred(25) of an RBFN and the number of clusters used in the fuzzy C-means, c

Regarding the MMRE criterion, it can be noticed that the values obtained are higher than 30 even when the value of c tends towards 151. This is due to the fact that the fuzzy C-means, instead of C-means, allows the projects to belong to several clusters and the MMRE is very sensitive to the outliers. As in the case of RBFN using C-means, we are interested with the accuracy when the values of c tend towards zero. It is noticed that when c is lower than 127, the RBFN using fuzzy C-means estimates accuracy in terms of Pred(25) becomes unacceptable (Pred(25)<70%). According to the MMRE criterion, the RBFN may be considered inaccurate for all values of c (MMRE>30). Taking into consideration only the Pred(25) as an indicator of the accuracy evaluation, we conclude that the threshold value of c to use an RBFN based on fuzzy C-means for the ISBSG dataset is around 127. This is less than of RBFN based on C-means but still very high regarding the maximum possible value of 151 clusters. It may be due to the fact that software projects in the ISBSG dataset are not sufficiently similar.

C. *RBFN using C-means vs RBFN using fuzzy C-means*

In this section, we compare in terms of accuracy and tolerance of imprecision an RBFN based on fuzzy C-means with the one when using hard C-means.

For the accuracy measured with the MMRE criterion, Figure 4 shows that an RBFN using hard C-means is more accurate than an RBFN based on fuzzy C-means for all values of c. However, the MMRE is well known to be very sensitive to the outliers. Hence, we only prefer to compare and evaluate the accuracy by means of the Pred(25) criterion.

Figure 5 compares the Pred(25) of an RBFN using the fuzzy c-means with that of an RBFN when using the hard C-means. It can be noticed that when the number of hidden neurons (number of clusters c) is higher than 127 and tends towards 151, the accuracy of an RBFN based on C-means is better than that one of an RBFN based on fuzzy C-means. However, the aim is to reduce the number of clusters with keeping a better accuracy. This is why it is very important to consider the accuracy when decreasing the number of clusters (c tends towards zero). From this point of view, the accuracy of an RFBN based on fuzzy C-means is better than that of an RBFN based on C-means when the number of clusters is lower than 127 and tends towards 0. Indeed, an acceptable accuracy

of an RBFN-fuzzy C-means is still achieved until the number of clusters is equal to 112; by contrast, it only was acceptable until the number of clusters is equal to 127 in the case of an RBFN based on hard C-means.

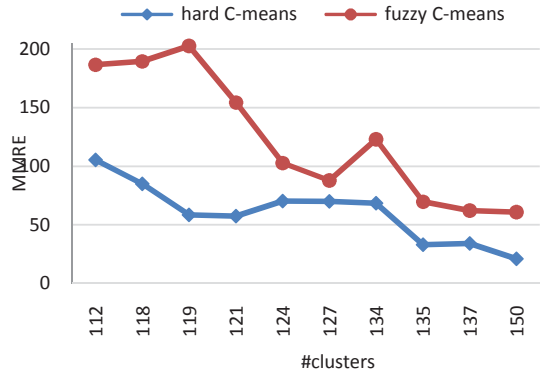


Figure 4. Comparison of MMREs

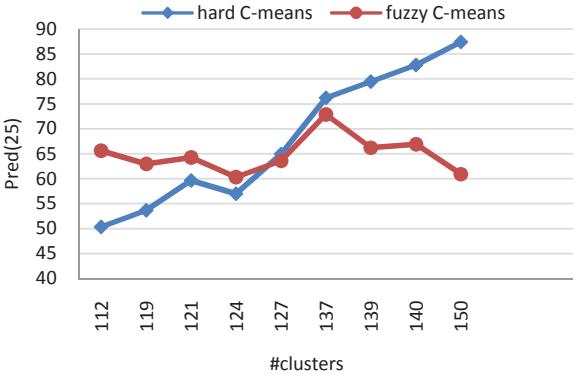


Figure 5. Comparison of Preds(25)

Regarding the tolerance of imprecision, an RBFN based on fuzzy C-means is better than an RBFN based on C-means. Indeed, the fuzzy concept allows the middle layer of an RBFN to perform a fuzzy classification of each project and hence the project may belong to several clusters. This is why even the number of clusters is set to 151, an RBFN based on fuzzy C-means avoid the well-known over fitting problem. By contrast, when the number of clusters is set to 151 for an RBFN based on C-means, its middle layer generates 151 clusters (e.g. each cluster contains one project) and hence each project only belongs to his cluster; so, there is a high risk of an overfitting.

V. CONCLUSION

In this paper, we have evaluated and compared RBFN based software effort estimation models using the ISBSG dataset. The RBFN construction is based on two clustering techniques: hard and fuzzy C-means. The results obtained confirms the findings with the COCOMO'81 and Tukutuku datasets except for the MMRE criterion: (1) RBFN networks based either on hard or fuzzy C-means are a promising technique for software effort estimation; (2) RBFN using fuzzy C-means generates more accurate estimates than an RBFN using hard C-means especially when decreasing the number of clusters; (3) An RBFN using fuzzy C-means better tolerates imprecision than an RBFN using C-means and hence may avoid the overfitting problem.

REFERENCES

[1] M. Jorgensen and M. Shepperd, "A Systematic Review of Software Development Cost Estimation Studies", IEEE Transactions on Software Engineering, vol. 33, no. 1, 2007, pp. 33-53.

[2] B.W. Boehm, "Software Engineering Economics", Prentice-Hall, 1981.

[3] G. R. Finnie, G. Witting, and J.M. Desharnais, "A Comparison of Software Effort Estimation Techniques: Using Function Points with Neural Networks, Case-Based Reasoning and Regression Models, Systems and Software", vol. 39, no. 3, 1997, pp. 281-289.

[4] A. Idri, A. Abran, and S. Mbarki, "An Experiment on the Design of Radial Basis Function Neural Networks for Software Cost Estimation", 2nd IEEE International Conference on Information and Communication Technologies: from Theory to Applications, vol. 1, 2006, pp. 230-235.

[5] M. Shepperd and C. Schofield, "Estimating Software Project Effort Using Analogies", Transactions on Software Engineering, vol. 23, no. 12, 1997, pp. 736-747.

[6] R. W. Selby and A.A. Porter, "Learning from examples: generation and evaluation of decision trees for software resource analysis", IEEE Transactions on Software Engineering, vol. 14, no. 12, 1988, pp. 1743-1757.

[7] J. Wen, S. Li, Z. Lin, Y. Huc, and C. Huang, "Systematic literature review of machine learning based software development effort estimation models", Information and Software Technology, vol. 54, 2012, pp. 41-59.

[8] A. Idri, A.Zakrani and A. Zahi, "Design of Radial Basis Function Neural Networks for Software Effort Estimation", International Journal of Computer Science Issues (IJCSI),vol. 7,Issue 4, no 3, 2010,pp. 21-31.

[9] B. A. Kitchenham and E. Mendes, "A Comparison of Crosscompany and Within-company Effort Estimation Models for Web Applications", 8th International Conference on Empirical Assessment in Software Engineering, 2004, pp. 47-56.

[10] A. Zakrani and A. Idri, "Applying Radial Basis Function Neural Networks Based on fuzzy Clustering to estimate web Applications Effort", International Review on Computers and Software (I.RE.CO.S), vol. 5, n.5, 2010.

[11] ISBSG, International Software Benchmarking Standards Group, Data Release 8 Repository, 2003, http://www.isbsg.org.

[12] J. B. MacQueen, "Some Methods for Classification and Analysis of Multivariate Observations", Fifth Berkeley Symposium on Mathematical Statistics and Probability, 1967, pp. 281-297.

[13] J. C. Bezdek, "Pattern Recognition with Fuzzy Objective Function Algoritms", Plenum Press, New York, 1981.

[14] F. Ros, M. Pintore, and J. R. Chretien, "Automatic design of growing radial basis function neural networks based on neighboorhood concepts", Chemometrics and Intelligent Laboratory Systems, vol. 81, Issue 2, 2007, pp. 231-240.

[15] N. Benoudjit and M. Verleysen, "On the Kernel Widths in Radial-Basis Function Networks", Neural Processing Letters, vol. 18, no. 2, 2003, pp. 139-154.

[16] F. Schwenker and C. Dietrich, "Initialisation of Radial Basis Function Networks Using Classification Trees", Neural Networks World, vol. 10, 2000, pp. 476-482.

[17] Y. S. Hwang and S. Y. Bang, "An Efficient Method toConstruct a Radial Basis Function Network Classifier", Neural Networks, vol. 10, no. 8, 1997, pp. 1495-1503.

[18] A.Saha and J.D. Keeler, "Algorithms for better Representation and Faster Learning in Radial Basis Function Networks", Advances in Neural Information Processing Systems 2, NIPS, 1989, pp. 482-389.

[19] S.Haykin, "Neural Networks: A comprehensive Foundation", Prentice Hall, 1998.

[20] X. L. Xie and G. Beni, "A validity Measure for Fuzzy Clustering", IEE Trans. Pattern Anal. Mach. Intell, vol. 13, 1991, pp. 841-847.

Proposal of an Automated Approach to Support the Systematic Review of Literature Process

Jefferson Seide Molléri
UNIVALI – Universidade do Vale do Itajaí
Rua Uruguai, 458, Centro
Itajaí, Brazil
+55 47 30452242
jefferson.molleri@univali.br

Luiz Eduardo da Silva
UNIVALI – Universidade do Vale do Itajaí
Rua Uruguai, 458, Centro
Itajaí, Brazil
+55 47 30452242
linkz.ns@univali.br

Fabiane Barreto Vavassori Benitti
UNIVALI – Universidade do Vale do Itajaí
Rua Uruguai, 458, Centro
Itajaí, Brazil
+55 47 33417544 (#8057)
fabiane.benitti@univali.br

ABSTRACT

Context: *Systematic Literature Reviews (SLR) is a scientific method to identify and assess all available studies related to a specific research topic. Due its characteristics, SLRs are a time consuming, hard process that requires a properly documented protocol for scientific acknowledgment.*

Objectives: *In this context, this paper aims to propose a business model to support automation of the systematic review method, contributing to the productivity and quality of the process.*

Method: *Through the results of a previous review, we identified several contributions to the systematic review process. We define the process using the Business Process Model Notation (BPMN) and relate possible tool contributions to the proposed model.*

Results: *The model and contributions proposed in this paper can be used to guide the development of computational tools to support SLR process and the proper execution of its methodology.*

Conclusions: *The implementation of tools supporting SLR processes seems relevant to reduce effort and to ensure the quality of the application of the methodology.*

Keywords

Systematic Literature Review, Computational Tool, Business Process Model.

1. INTRODUCTION

Systematic Literature Review (SLR) is an empirical methodology of research which aims to gather and evaluate all the available evidence about a specific research topic [1]. SLRs are a key tool for evidence-based research and practice as they combine the results of multiple studies. These literature reviews are important, as the volume of studies to be considered by the researchers is constantly expanding [2].

Despite its importance, the SLR process is not a easy task, as it uses specific concepts usually unknown to researchers familiar with traditional literature reviews. Even when conducted according to their 'good practice' rules, often SLRs suffers from lack of scientific rigor in its several steps. An automated systematic review process aims a stricter and better managed method of conducting this methodology, avoiding the the biases of the unsystematic review [1].

Further, the systematic reviews require considerably more effort than traditional reviews, as they provide additional data on variations in the primary studies [3]. Therefore, some systematic reviews strongly depend on a computerized infrastructure to support their processes [4]. In view of difficulties in applying this empirical research methods, there is a need to invest efforts in support tools for planning and conducting systematic literature reviews [1]. An automated or semi-automated tool is essential to provide quality and productivity to the process [5].

Based on the systematic review process proposed by Kitchenham [6] and later improved by Kitchenham and Charters [2] we develop a business model to support systematic reviews process through automated tools. Through a prior systematic review of literature [7], we also identified several proposed contributions to this automated process.

This paper is organized as follows: section 2 introduces an overview of the model to an automated systematic review process; sections 3 to 5 present the proposed contributions to each stage of the process; and section 6 discusses the results obtained with the proposed model facing the identified contributions, and further works on implementation of the automated process.

2. PROPOSED MODEL

As described by Kitchenham [6] and later by Biolchini [1], the systematic review process consists of three main phases, each one containing a number of discrete activities and specific stages to its conclusion. The process is sequential, and later stages depend on results of their predecessors.

Thus, we present an overview of the process with a focus on automation of its activities, described by the Business Process Modeling Notation (BPMN), as detailed by the Object Management Group[1]. BPMN provides a graphical notation for specifying processes based on a flow-charting technique very similar to the activity diagram of Unified Modeling Language (UML) [8]. The model can be used for modeling workflow processes and to represent the semantics of complex processes.
actors for success or factors are not relevant for the research [12].

[1] Available at http://www.bpmn.org/

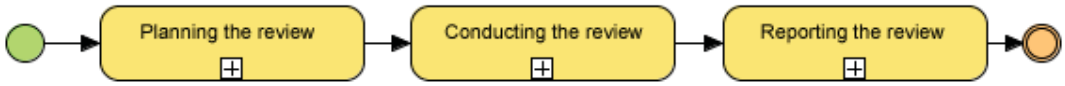


Figure 1. Systematic Review Process

The business model proposed in this paper involves three phases of systematic review process presented by Kitchenham [6], as illustrated in Figure 1. Each phase and its stages are described as follows, and the contributions identified are related to each one. We detail further discrete activities, artifacts produced, participants involved, and the communication mechanisms. The completeness of the process takes place when finalizing systematically all the steps in this modeling.

Following elements are used to represent the SLR process in a BPMN context: (i) events, representing initial and end states; (ii) activities, as phases and stages; (iii) gateways, representing conditions and convergence actions; (iv) sequence flows, that integrate events and activities; (v) communication flows, which represent the interaction among participants; (vi) data objects representing the artifacts of the process; (vii) messages, which details the contents of a communication; and (viii) associations, to connect actions and artifacts.

3. CONTRIBUITIONS TO PLANNING THE REVIEW PHASE

SLR process starts with a planning phase, consisting in five distinct stages, as illustrated in Figure 2. First stage, **identification of the need for a review** aims the originality of the work by consulting knowledge repositories in the study field, as conceived by Lopes and Travassos [9] or to identify a SLR to be repeated in a new context or iteration that contributes to previously obtained knowledge. Repeatability is one of the basic premises for assuring the quality of a SLR [10].

Commissioning the review stage is suggested when an organization requires information about a specific topic but does not have the time or expertise to perform a systematic review itself. Proposed model allows the management of stakeholders in this stage[2]. Stakeholders can assume different roles in the process: (i) the research coordinator, (ii) the researchers; (iii) independent reviewers and moderators, and (iv) potential observers with no assigned tasks in the process. Even when conducted individually, stakeholder management enables to invite field experts as observers or moderators in the process.

The **specifying the research question(s)** stage is based on the PICOC (Population, Intervention, Comparison, Outcome, Context) criteria to frame research questions, as described by Pai *et al.* [11] and expanded by Petticrew and Roberts [12]. Aiming the quality of the review, the researcher is encouraged to fill out all the criteria, but it is possible to perform SLR with less formalism blanking the comparison criterion in characterization reviews [13] or the context criterion when the factors for success or factors are not relevant for the research [12].

Proposed model should also suggests generate a search string by concatenating each of the terms proposed for PICOC acronym, using the logical operator 'AND' to associate criteria. The logical operator 'OR' can be used for concatenate various terms within the same criterion on the research question.

Developing a review protocol consists of documentation of protocol elements and some additional planning information of a systematic literature review as proposed by Kitchenham and Charters [2]: (i) background; (ii) the research questions; (iii) research strategy; (iv) study selection criteria; (v) study selection procedures; (vi) study quality assessment procedures; (vii) data extraction strategy; (viii) synthesis of the extracted data; (ix) dissemination strategy; and (x) project timetable.

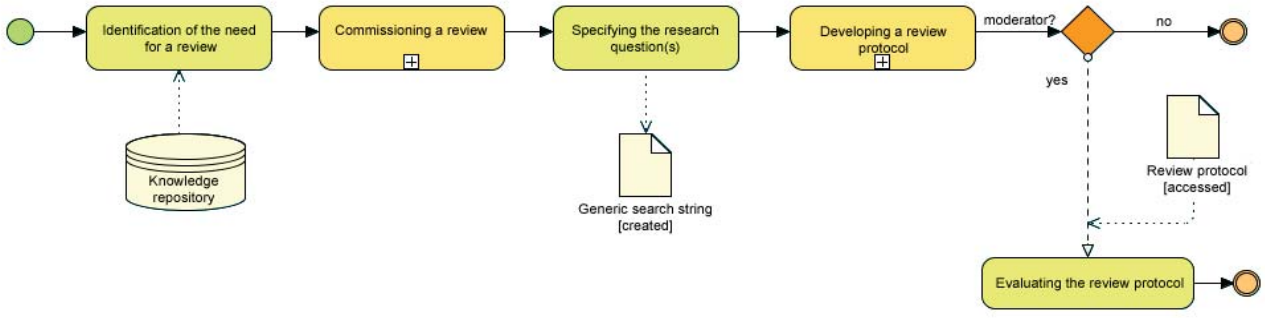


Figure 2. Planning review phase

[2] Available at http://goo.gl/PHXZu

Elements of the protocol should be stored in a computational artifact that can be accessed later. Selection criteria, electronic databases references, extraction data and the timetable must be also stored in a suitable data structure, as presented in the process model[3]. After the protocol documentation, stakeholders should be informed of their assigned tasks and dates. The stage further comprises exporting a review protocol document supported by guidelines such as Kitchenham and Charters [2] or templates as detailed by Biolchini *et al.* [1].

Evaluating the review protocol stage is optionally conducting to ensure the quality of the process. The protocol documentation is reviewed by moderators, who submit comments to the research coordinator that performs the necessary changes.

This proposed model suggests several contributions in automated support to the planning review phase, described in Table 1. Identified contributions aim to address all stages in the planning phase, automating a number of activities through specific methods and techniques.

Table 1. Contributions to Planning Review Phase

Stage 1.1 Identification of the need for a review
Consult a knowledge repository, as Lopes and Travassos [9].
Stage 1.2 Formatting the main report
Manage stakeholders, according to communication strategies [14].
Stage 1.3 Specify the research question(s)
Address research question by filling the five criteria of the PICOC acronym [11][12].
Concatenate the terms proposed for each criteria on the PICOC acronym to automatically generate a search string.
Input the RSL protocol elements and additional information for planning the review in a computational artifact.
Stage 1.14 Developing a review protocol
Documentation of the RSL protocol elements and additional information for planning the review in a computational artifact.
Export a review protocol document in rich text format, containing all its elements and additional information for planning the review [2].
Manage inclusion and exclusion criteria in a computational structure that can be used during selection of primary studies stage.
Manage process timetable, assigning tasks to the stakeholders [2][20].
Inform stakeholders tasks and dates, according to communication strategies [14].
Use guidelines such as Kitchenham and Charters [2], or protocol templates, as by Biolchini *et al.* [1] to support the SLR process.
Stage 1.5 Evaluating the review protocol
Manage comments from moderators in the review protocol artifact.

4. CONTRIBUITIONS TO CONDUCTING THE REVIEW PHASE

Conducting the review phase also involves five stages, as detailed in Figure 3. We notice that not all stages in this phase are performed sequentially, but there is also a strong dependence of the early stages.

During the **identification of research** stage[4], search strings are automatically generated for each electronic database, as suggested by Brereton *et al.* [15]. Search strings are created by applying over the generic search string (detailed at the research question(s) stage) specific requirements of the electronic databases. Automatically generated strings can also be reformulated by researchers to suit specific formats or rules.

Based on that, proposed model suggests preliminary searches on electronic databases to check volume and accuracy of the identified studies, as well as possible biases. Researchers can also include comparison studies that, if identified in preliminary searches could be analyzed by query expansion techniques to suggest new terms for the search string [16].

Results of preliminary searches may cause changes in the search string and methods documented in the review protocol. Once finalized the changes, preliminary searches could be repeated, refining the search string until it is suitable to the researchers. To solve divergences about changes on search strings, proposed business model also suggests communication mechanisms between stakeholders. Following the activity of preliminary searches, the references of the studies should be collected and managed.

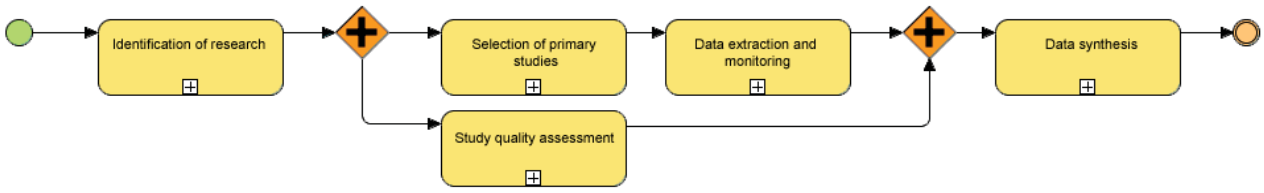


Figure 3. Conducting the review phase

[3] Available at http://goo.gl/GKh9b

[4] Available at http://goo.gl/1C40T

Proposed model also includes reference management of collected studies by manual input of specific references or importing references through common formats such as EndNote and BibTex. Automation can be done by integration with Application Programming Interfaces (API) of the electronic databases. Details of the studies should also be stored in the search process documentation, an artifact containing a list of the primary studies collected, that serve as input for the subsequent stages in the conducting the review phase [6].

The **selection of primary studies** stage assess the collected studies list according to selection criteria defined in the review protocol. This assessment can be done in three different ways, as illustrated in the process model[5]. The selection through multiple researchers involves every research team member assessing all the studies individually. Divergences found should be solved by consensus or by the intervention of a mediator. For such, communication mechanisms upon the divergences are part of the proposed model. Statistical coefficients, as the Cohen Kappa statistic [16] should also be available as supporting tools, providing helpful information in measure the agreement between researchers.

Another selection method involves inter-rater reliability tests, as cited by Khan, Niazi and Ahmadd [18], on which a researcher assess all collected studies, and later a secondary reviewer assess only a sample of these. The secondary reviewer can be another member of the research team or the research coordinator. If the assessment of the sample matches the primary researcher's, the selection is considered valid. Otherwise, the selection of the studies should be redone to ensure the reliability of the study selection criteria applied.

The last selection method involves the automatic assessment of collected studies from a minimum rating of acceptable quality. Details regarding the quality assessment are provided in the stage below.

Study quality assessment stage consists in creating a list of assessment criteria to be answered, ensuring the quality of the selected studies. Studies with a higher ranking are more significant for the systematic review than the others. This questionnaire can also be applied as a method for selection of primary studies, or to score the primary studies suitable for the data synthesis stage. The research coordinator can define a specific questionnaire, or choose study quality procedures such as the CRD Guidelines [18] and the Cochrane Reviewers' Handbook [19] as assessment criteria. At the end of the stage, the search process documentation is updated with the scores of selected studies.

Data extraction and monitoring stage involves collect specific information needed to address the review questions. Extraction forms are designed based on the review protocol and filled in several ways: some data could be automatically extracted; other should be collected from the study references; finally the context-specific data require that the researcher read the selected studies and interpret their results.

[5] Available at http://goo.gl/EP2ts

The **data synthesis** summarize the extracted data as tables and charts that demonstrate in a more natural way the results of the selected studies. The proposed model includes application of statistical models as the random effects model or the fixed effects model [12], forest plots and/or other appropriate formats for graphical representation of data [2], and the selection of prominent sentences to supply an informative summary [20] for an automated systematic review process.

Finally, contributions to the model in conducting the review phase are presented in Table 2.

Table 2. Contributions to Conducting the Review Phase

Stage 2.1 Identification of research
Formulate and maintain specific search strings for electronic databases.
Generate search strings to electronic databases, as Brereton *et al.* [15].
Include known primary studies for comparison.
Suggest new terms for the search string composition from application of text mining techniques over comparative studies [16].
Conduct preliminary searches on electronic databases to check the volume and accuracy of the identified studies.
Identify and highlight comparative studies during preliminary searches.
Facilitate communication among stakeholders to resolve divergences concerning search string and/or search methods changes.
Manage references of identified studies in preliminary searches.
Import references of the primary studies identified through common formats such as EndNote and BibTex.
Import references of the primary studies identified through integration with APIs in electronic databases.
Document the search process and electronic databases information.
Stage 2.2 Selection of primary studies
Select primary studies according to criteria for inclusion and exclusion of studies defined in the review protocol.
Manage divergences of selection of primary studies between researchers.
Inform research coordinator/moderators about divergences of selection of primary studies.
Apply statistical coefficients, as the Cohen Kappa statistic [17] to assist in solving divergences.
Conduct inter-rater reliability tests between researchers for quality assurance in the selection of studies [18].
Stage 2.3 Study quality assessment
Develop and apply study quality assessment lists.
Assess primary studies through the quality criteria list.
Consult quality assessment support procedures, as the CRD Guidelines [18] and the Cochrane Reviewers' Handbook [20].
Use templates of quality assessment lists [19].

Stage 2.4 Data extraction and monitoring
Maintain data extraction forms.
Automatically generate data extraction forms from data defined in the review protocol.
Record specific data to selected studies into data extraction forms.
Automatically extract data from the primary study references, such as title, authors, journal, publication details [2].
Stage 2.5 Data synthesis
Simple tabulating numerical data extracted from the primary study.
Calculating the weighted average by the application of statistical models: the random effects model or the fixed effects model [12].
Summarizing the extracted data in graph form.
Automatically generating graphical representation of data in forest plots and/or other appropriate formats [2].
Obtaining proeminent sentences through text mining techniques to provide an informative summary of the review [14].
Selecting prominent sentences to increase the data synthesis and the main report

5. CONTRIBUITIONS TO REPORTING THE REVIEW PHASE

The last phase of SLR process involves a report of the systematic review, and later its dissemination and evaluation, as illustrated in Figure 4. Its first stage, **specifying dissemination mechanisms**, consists in define the format of main report, as a technical report or a journals or conference paper. The proposed model provides templates for automatic generation of reports, as Kitchenham and Charters [2] or Biolchini *et al.* [1]. Knowledge repositories in software engineering could also be used to record and disseminate the research [9].

Formatting the main report stage is done by combining the appropriate report template with the data synthesis results. Researchers must be able to write specific sections of the report in a similar way to the review protocol, and later export this document.

After finish the report formatting, mediators can **evaluating the report**, adding comments as in the evaluating the review protocol stage. Quality assessment for secondary studies [19] and an efficient communication mechanism among stakeholders are proposed support tools for this stage.

Table 3 shows the contributions intended to reporting the review phase in proposed model.

Table 3. Contributions to Reporting the Review Phase

Stage 3.1 Specifying dissemination mechanisms
Define the publication format.
Automatically generate the report through report templates.
Record main report in a knowledge repository, as proposed by Lopes and Travassos [9].
Stage 3.2 Formatting the main report
Document main report as a computational artifact.
Export the main report in rich text format containing all the sections documented combined with the template specified.
Include comments from moderators in the main report artifact.
Stage 3.3 Evaluating the report
Use lists of quality templates for assessing the quality of secondary studies [19].
Submit main report to a knowledge repository, as conceived by Lopes and Travassos [9].

After the report is approved by the moderators, the systematic review is given as finished. Despite this, their data and documents created in the process should be available for further evaluation, acting as a validation mechanism of the methodology and sustaining the quality of SLR produced. This data could also serve as basis for repeating the systematic review in a new context or iteration.

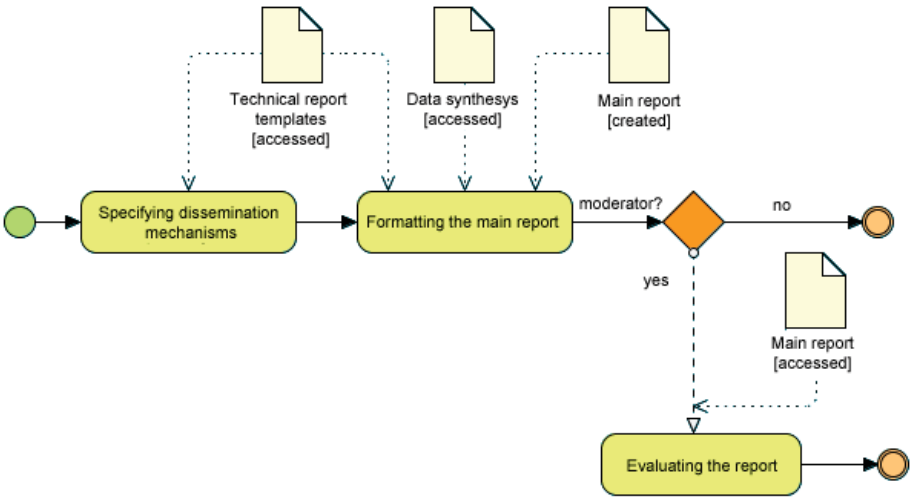


Figure 4. Conducting the review phase

6. CONCLUSIONS

The process of conducting systematic reviews involves a series of discrete activities, arranged in specific stages and phases identified by Kitchenham [6], later by Biolchini [1] and finally Kitchenham and Charters [2]. In addition to these activities, several authors provide further guidelines, that contribute to the quality of the process or reducing the effort by automating more exhaustive tasks.

This paper enhances these guidelines with the proposal of a business model that addresses and support the SLR process. The proposed model is detailed in a BPMN notation and includes all phases and stages of the process, providing support on the proper execution of its methodology. Contributions identified in a previous paper, its supporting methods and techniques were related to the specific stages of the model, forming a basis for development of computational tools to support the process.

As further works, we plan to implement features that comprise the model proposed, also plan and conduct the software evaluation on productivity and quality criteria. Given this, the model proposed seems to be relevant in the context of empirical studies, providing effort reduction through automated tasks, and ensuring the quality through the systematic conduction of the methodology.

7. REFERENCES

[1] Biolchini, J., Gomes, P., Cruz, A., & Travassos, G. 2005. *Systematic review in software engineering.* Technical Report. Universidade Federal do Rio de Janeiro.

[2] Kitchenham, B., Charters. S. 2007. *Guidelines for performing systematic literature reviews in software engineering* (version 2.3). Technical Report. University of Durham.

[3] Shull, F., Carver, J., and Travassos, G. H. 2001. An empirical methodology for introducing software processes. In *Proceedings of 8th European Software Engineering Conference* (Vienna, Austria, September, 2001). ACM. 288–296.

[4] Zamboni, A., Thommazo, A., Hernandes, E. C. M., Fabbri, S. C. P. F. 2010. StArt – Uma Ferramenta Computacional de Apoio à Revisão Sistemática. In *Proceedings of CBSOFT – Congresso Brasileiro de Software: Teoria e Prática* (Salvador, Brazil, September, 2010) SBC. 1–6.

[5] Ali, S., Briand, L. C., Hemmati, H., Panesar-Walawege. R. K., 2010. A Systematic Review of the Application and Empirical Investigation of Search-Based Test Case Generation, *IEEE Transactions on Software Engineering*, 36, 6 (Nov.-Dec. 2010), 742–762.

[6] Kitchenham, B. 2004. *Procedures for Performing Systematic Reviews*. Technical Report. Keele University Joint Technical Report.

[7] Molléri, J. S., Benitti, F. B. V., 2012. Automated Approaches to Support Secondary Study Processes: a Systematic Review. In *Proceedings of 24th SEKE – International Conference on Software Engineering & Knowledge Engineering* (San Francisco, USA, July, 2012). Knowledge Systems Institute Graduate School. 143–147.

[8] OBJECT MANAGEMENT GROUP. Business Process Model and Notation. Available at: <http://www.bpmn.org/>. Acessed: jun 2012.

[9] Lopes, V. P., Travassos, G. H. 2008. Infra-estrutura Conceitual para Ambientes de Experimentação em Engenharia de Software. In *Proceedings of ESELAW'08 - Experimental Software Engineering Latin American Workshop* (Salvador, Brazil, November, 2008) SBC. 34–44.

[10] Biolchini, J., Gomes, P., Cruz, A., Uchôa, T., & Travassos, G. 2007. Scientific research ontology to support systematic review in software engineering. *Advanced Engineering Informatics*, 21, 2 (Apr. 2007), 133–151.

[11] Pai, M., Mcculloch, M., Gorman, J. D., Pai, N., Enanoria, W., Kennedy, G., Tharyan, P., Colford, J. M. 2004. Systematic reviews and meta-analyses: an illustrated, step-by-step guide. *National Medical Journal of India*, 17, 2 (Mar-Apr. 2004), 89-95.

[12] Petticrew, M., Roberts, H. 2006. *Systematic Reviews in the Social Sciences: A Practical Guide*. Blackwell Publishing, Malden, MA.

[13] Travassos, G. H., dos Santos, P. S. M., Neto, P. G. M., & Biolchini, J. 2008. An environment to support large scale experimentation in software engineering. In *IEEE International Conference on Engineering of Complex Computer Systems ICECCS* (Belfast, Northern Ireland, March, 2008). IEEE. 193–202.

[14] Felizardo, K. R., Andery, G. F., Maldonado, J. C., Minghim, R. 2009. Uma Abordagem Visual para Auxiliar a Revisão da Seleção de Estudos Primários na revisão sistemática, In: *Proceedings of VI Experimental Software Engineering Latin American Workshop – Eselaw*, 6. (São Carlos, Brazil, November 11-13, 2009). SBC. 83–133.

[15] Brereton, P., Kitchenham, B. A., Budgen, D., Turner, M., Khalil, M. 2007. Lessons from Applying the Systematic Literature Review Process within the Software Engineering Domain. *Journal of Systems and Software*, 80, 4 (Apr. 2007), 571–583.

[16] Ananiadou, S., Okazaki, N., Procter, R., Rea, B., Thomas, J. 2009. Supporting Systematic Reviews using Text Mining. *Social Science Computer Review*, 27, 4 (Nov 2009), 509–523.

[17] Cohen, J. 1968. *Weighted Kappa: nominal scale agreement with provision for scaled disagreement or partial credit*. Psychological Bulletin, 70, 4 (Oct. 1968), 213–220.

[18] Khan, K. S., Ter Riet, G., Glanville, J., Sowden, A. J., Kleijnen, J. 2000. *Undertaking Systematic Review of Research on Effectiveness*. CRD's Guidance for those Carrying Out or Commissioning Reviews, 2.ed. CRD Report n.4. York: NHS Centre for Reviews and Dissemination, University of York.

[19] Dybå, T., Dingsøyr, T. 2008. Strength of evidence in systematic reviews in software engineering, In: *Proceedings of ESEM'08 - ACM-IEEE International Symposium On Empirical Software Engineering And Measurement* (Kaiserslautern, Germany, October 9-10, 2008). ACM New York. 178–187.

[20] Alderson, P., Green, S., & Higgins, J. P. T. 2004. Cochrane reviewers' handbook 4.2. 2. In: *The Cochrane Library*, 1 (2004). Chichester: John Wiley & Sons Ltd.

Automated Computation of Use Cases Similarity can Aid the Assessment of Cohesion and Complexity of Classes

Renato C. Juliano*, Bruno A. N. Travençolo†, Michel S. Soares ‡, Marcelo de A. Maia§
*†‡§Faculty of Computing - Federal University of Uberlândia - Uberlândia, Minas Gerais, Brasil

†travencolo@gmail.com, ‡mics.soares@gmail.com, §marcmaia@facom.ufu.br
*Institute of Exact and Human Sciences - Centro Universitário do Planalto de Araxá - Araxá, Minas Gerais, Brasil
*renatocorrea@uniaraxa.edu.br

***Abstract*—Use cases are widely used in early software development phases such as requirements analysis. In this paper, we investigate how use case similarity can impact the classes that implement them. We studied whether the similarity of use cases can have impact on the lines of code shared between them and on metrics of classes that implement them, such as coupling, cohesion and method complexity. We have also successfully applied an automated approach to assess the similarity of use case names. We found that there is a statistically significant correlation, although not strong, between use case similarity and sharing of lines of code. Interestingly, we have also found that classes that are shared between different use cases tend to have lower cohesion. Moreover, classes that are shared between similar use cases tend to have higher method complexity in classes. We found no relation between use case similarity and coupling.**

Keywords*—*Use Case Similarity, Software Metrics, Software Maintenance

I. Introduction

Software maintenance has been object of study for many years [2], [10], [16], as it is well-established that software needs to evolve in order to keep being useful. Software maintenance accounts for up to 80% of software costs [8], [18]. In addition, according to some studies, the faster an error is discovered, the easier and less expensive and time-consuming is to correct it [6]. Therefore, any activity that facilitates the discovering of software errors or at least helps in predicting future problems is useful for maintaining the software. It is even more appropriate when this activity belongs to initial phases of software development process, such as during requirements analysis and design activities.

Use cases, which are commonly described using a tabular approach or simple natural language and using the UML use case diagram, are responsible for describing the external activities of a software [11]. Use cases are useful to describe the functions to be provided by the software, usually from an actor point of view.

The proposal in this article is to describe a relationship between the software metrics CBO, LCOM and WMC, proposed in 1994 by Chidamber and Kemerer [3], with the level of similarity between the items of a Use Case diagram. The process to perform this task consists of two steps.

In the first step, we will study if it is feasible to use an automated algorithm to assess use case similarity based on algorithms for computation of string similarity. This step will be evaluated by human experts. In the second step, we will study how the similarity between two use cases can be related to CBO, LCOM and WMC of the concrete software.

The reminder of the article is organized as follows. The next section is about describing the literature basics to perform this work. Among the topics, we described the analysis of similarity between strings of use cases, software metrics and the correlation between them. In section III, we describe the approach to collect data, to extract software metrics, and to relate software metrics with the level of similarity between use cases. In Section V the results of the article are described, and Sections VI and VII brings the threats to validity and conclusion.

II. Background

A. Similarity between texts in use case

In order to perform the analysis proposed in this paper, it is necessary to find similarities between items in a use case diagram. There are two ways in doing this analysis, one by using a human specialist and other by automated techniques.

Human analysis has a series of advantages, specially regarding the context awareness and interpretation of texts, combined with the experience of the developer. However, these advantages are only valid when the experience and the knowledge about the problem are clearly defined, otherwise the recognition will be compromised.

Automated computation of similarity can significantly reduce time of analysis. However, this is not a trivial task, because defining the context of the used words is difficult [5]. Several proposals were found in the literature in order to evaluate similarity between texts [4]. In this work we use the *Jaro-Winkler* [22] and *Levenshtein* [12] proposals.

In this work, we found a similarity threshold value that balanced the precision/recall relation according to the analysis of human specialists. Therefore, it is possible to automate the process of similarity computation.

B. Software Metrics

Many studies have shown that the process of software development can be significantly improved if the quantification of involved process is adopted along all stages [3]. Several metrics were proposed in order to quantify different features of the software [1], [3], [13]. Among then, in this work we used the set of metrics proposed by Chidamber and Kemerer [3]. This set of metrics, known as CK suite, is composed of six metrics related to the object-oriented paradigm [9].

The use of CK metrics can indicate several characteristics of the process of software development, as error-proneness [15], [19], [21], bad smells identification [17], analysis of the effectiveness of refactoring [20] and assessment of the maintainability of the process of software development [6]. In this work, three CK metrics were used, namely WMC, CBO and LCOM:

- WMC - *Weighted Methods Per Class* : The WMC is a measure defined by the sum of the complexity of the methods of the class. The definition of the complexity was not specified by Chidamber and Kemerer [9], which suggested the use of a measure expressed by a natural number. The WMC is related to the number of methods of the class, and their complexity may be proportional to the amount of time necessary to develop and maintain the class (the greater the number of methods of a class, the greater the impact on their subclasses). Classes with a large number of methods normally refer to specific applications, reducing the possibility of reuse [3]. In this paper, for WMC, is used the McCabe Cyclomatic Complexity [14].
- CBO - *Coupling Between Object Classes*: The CBO metric represents the number of classes in which it is coupled. One class is coupled to another when at least one method of one of the class uses the method or attributes of the other class. By using this measurement it is possible to identify the level of reuse of a class and its degree of modularity. The lower the value of CBO is, more modularized is the class and, as consequence, the possibility of reuse increases. In addition, it is possible to analyze the rigor of software testing, by means of the degree of importance, the amount of time used and number of tests. The higher the CBO is, more complex will be the testing [9].
- LCOM - *Lack of Cohesion in Methods*: The LCOM metric quantifies the similarity between the methods inside a class. Two or more methods are cohesive when they share the attributes of the class. When the value of LCOM is high, it means that the class does not have well defined functionality, as several methods modify the same attributes. On the other hand, if LCOM is low, the methods of the class are cohesive and, hence, less similar [9].

III. Study Setting

The conduction of this study was organized in several steps that will be described below.

Definition of the subject system

The system used in this study is a system in the educational domain. The system has 17 use cases, which are implemented with 3451 LOC, 34 classes and 181 methods. The system is written in C#.

There are several reasons that justified the choice of the system: it is a real working system with over 20 end-users, it has well-documented use cases, the company that developed the system has provided both the source code and developers, which were allocated to recover the traceability of use cases and classes that implement them and to assess the results of the similarity algorithms.

Use cases

The use cases of the subject system used in this study are shown in Table I. We decided to maintain the original names in Portuguese (the language used in the specification), because this was the actual input to similarity computation. We also provided a translation of the terms to English.

TABLE I. Use cases.

ID	Description
uc1	Lançar Dados da Banca (*Insert examining board information*)
uc2	Visualizar TCC Postados (*Visualize Final Project (FP) posted*)
uc3	Disponibilizar Visualização do TCC no Site do Uniaraxá (*Provide visualization of FP on Uniaraxá's Site*)
uc4	Manter título do TCC (*Keep title of FP*)
uc5	Impressão da ata de defesa (*Printing protocol of examining board*)
uc6	Impressão da lista de presença (*Printing presence list*)
uc7	Impressão da lista de alterações/correções (*Printing fix list*)
uc8	Lançar nota(status) do TCC (*Insert grades(status) of FP*)
uc9	Escolha dos alunos que farão TCC (*Choose students that will do FP*)
uc10	Postar TCC (*Post SCC*)
uc11	Incluir título do TCC (*Insert title of SCC*)
uc12	Visualizar nota(status) do TCC (*Visualize grades(status) of FP*)
uc13	Lançamento do Status TCC para a disciplina (*Insert status FP in discipline*)
uc14	Relatório Gráfico por curso: alunos aprovados vs todos (*Report Chart by Course: approve vs reproved*)
uc15	Relatório de alunos aprovados/reprovados por curso (*Report of students approved/reproved by course*)
uc16	Lançamento da disciplina de TCC (*Insert FP's discipline*)
uc17	Visualização do título do TCC (*Visualize title of FP*)
	FP* : Final Project

Traceability recovery of use cases and classes

This step was conducted with the team that participated actively in the development and documentation of the system. The team consisted of four developers with at least three years of experience. The result of this step was the definition of a set of classes that implemented each use case. It is possible that a class participates in the implementation of one or more use cases.

Evaluation of similarity algorithms

This step was conducted to evaluate if algorithms to evaluate string similarity would be a substitute for human analysis of use case similarity. The approach consisted in applying an algorithm to compute the string similarity, which was tested in a pairwise comparison of all pairs of use cases. The chosen algorithms was Jaro-Winkler [22] and Levenshtein [12]. The result of the algorithms is a number between zero (completely dissimilar) and one (completely similar). The result of the algorithm was assessed against a gold answer set, which

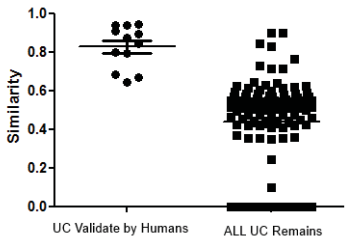


Fig. 1. Similarity values between pairs of use cases: (left) considered similar by developers (right) considered dissimilar by developers.

was produced by the development team consisted of four developers with three, seven, eight and ten years of experience, respectively. The developers received the use case diagram of the system and the following instructions to perform the comparison:

- Consider *similarity* use case names (titles) that have some semantic similarity between them that could indicate the sharing of implementation classes.
- Select the pairs of use cases that you consider that have some degree of similarity.

After the instruction phase, the developers produced their set with pairs of similar uses cases. After that, they shared and discussed their results to reach a consensus.

In order to define a threshold for the similarity algorithm we used a ROC curve to assess precision and recall values of the similarity algorithms. The area under the ROC curve of Jaro-Winkler was higher than the area of Levenshstein, so we decided to consider the algorithm Jaro-Winkler. Figure 1 depicts the similarity values for use case pairs that were considered similar by the developers (right) and the similarity values of the pairs that were not considered similar by developers (left). This figure presents the feasibility of automatic similarity computation.

The next step was the definition of the threshold for balanced precision/recall of the similarity algorithm. Table II presents a inflexion point in the ROC curve that provide a balanced threshold. For that threshold (similarity > 0.6409), we get recall equals 60% and precision equals 87.93%.

TABLE II. PRECISION AND RECALL OF JARO-WINKLER BASED ON EVALUATION BY DEVELOPERS.

Feature	Value
Similarity	> 0.6409
Recall	60,00
95% CI	36.05% to 80.88%
Precision	87,93
95% CI	80.58% to 93.24%
Likelihood	**4.97**

Classification of classes shared between use cases

In our study, our goal is to analyze if classes shared between similar and non-similar use cases have special characteristic concerning the metric values for LCOM, CBO and WMC. Our criteria to classify similar use cases and to classify shared classes between use cases is shown in Table III, where *X* is the value of Jaro-Winkler metric and *Y* is the number of lines of code (LOC) of classes that are shared between use cases.

In order to assess the percentage of LOC sharing we will use the following equation:

$$\frac{\sum number\ of\ Nonshared\ LOC's - number\ of\ Shared\ LOC's}{\sum (number\ of\ Shared\ LOC's)}$$

TABLE III. CRITERIA FOR SIMILARITY OF USE CASES AND SHARING OF CLASSES.

Classification	Value
Dissimilar	X < 0.6409
Similar	X >= 0.6409
There is class sharing	Y > 0
No class sharing	Y = 0

Metrics extraction

In order to extract our chosen metrics we used the open source library *Mono.Cecil*[1]. Using this library, we calculated the values for WMC, CBO and LCOM.

A. Research questions

The main focus of our research is the analysis of the impact of use case names similarity in the sharing of classes between use cases and what kind of impact this sharing can have in the metrics WMC, CBO and LCOM.

The justification for this study is that according to the analysis of use cases, we would predict what impact the similarity of use cases can have in software metrics that assess the design quality. We suggest that this analysis can help developers to prevent anomalies in software design, either in early stages or during maintenance.

Research Question 1 (RQ1). *In similar uses cases, do the classes that implement them have higher percentage of shared lines of code (LOC) compared to dissimilar use cases?* Our hypothesis is that if use cases u_1 and u_2 are considered similar, then they will have shared lines of code because we expect some reuse in the implementation of those use cases and that sharing is supposed to be higher in classes that implement similar use cases.

Research Question 2 (RQ2). *In similar uses cases, do shared classes that implement them have lower cohesion compared to the other classes of the system?* Our hypothesis is that if use cases u_1 and u_2 are considered similar, then classes that implement them have lower cohesion than the other classes of the system because although they work in similar problems they need to manage the variability of the use cases.

Research Question 3 (RQ3). *In dissimilar uses cases, do shared classes that implement them have lower cohesion (higher LCOM) compared to the other classes of the system?* Our hypothesis is that if use cases u_1 and u_2 are considered dissimilar and still have shared classes that implement them, then those classes would have lower cohesion because they have responsibility to provide specialized services to dissimilar use cases.

[1]http://www.mono-project.com/Cecil

Research Question 4 (RQ4). *In dissimilar uses cases, do shared classes that implement them have higher complexity (WMC) compared to the other classes of the system?* Our hypothesis is that if use cases u_1 and u_2 are considered dissimilar and still have shared classes that implement them, then those classes would have higher complexity because they have to cope with the variability of different use cases.

Research Question 5 (RQ5). *In similar uses cases, do shared classes that implement them have lower coupling compared to the other classes of the system?* Our hypothesis is that shared classes of similar use cases perform similar activity and so use the same classes for that use cases, and this would require lower coupling.

Research Question 6 (RQ6). *In dissimilar use cases, do shared classes that implement them have higher coupling (CBO) compared to the other classes of the system?* Our hypothesis is that shared classes of dissimilar uses cases perform different activities, and so, need to use different sources of classes and thus would have a high coupling with those classes.

Research Question 7 (RQ7). *Is the cohesion of classes shared by dissimilar use cases lower than the cohesion of classes shared by similar use cases?* Our hypothesis is that classes that implement dissimilar use cases would perform distinct tasks, so they would have lower cohesion than classes that, in principle, perform similar tasks.

IV. Results

In this section we present the results that will support the answers for the defined research questions.

The metrics collected from the subject system are shown in Table IV.

TABLE IV. Metrics of the subject system.

Variable	Mean	SD	Min	1Q	2Q	3Q	Max
LOC	101.5	112.7	14	30	58	127.3	596
WMC	23.5	15.4	4	12.3	19	28.5	72
CBO	5.4	5.5	0	0	4	11.3	20
LCOM	16.1	13.6	0	7	11	25.5	64

In order to answer the research questions the *Mann-Whitney U* test was applied to analyze whether the value of a metric in one data group is lower or higher than the other data group.

RQ1. According to the data shown in Figure 2, we applied the Spearman correlation test to analyze if the similarity of classes are correlated with the LOC sharing. The result is shown in Table V. Although we have a weak correlation ($\rho = 0.1891$), that correlation is statistically significant at p-value 0.0086.

TABLE V. Spearman correlation between use case similarity and shared LOC.

Number of XY Pairs	192
Spearman ρ	**0.1891**
95% confidence interval	0.04460 to 0.3259
P value (two-tailed)	**0.0086**
Exact or approximate P value?	Gaussian Approximation
Is the correlation significant? (alpha=0.05)	**Yes**

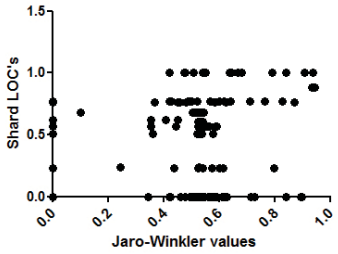


Fig. 2. Use case similarity vs. shared LOC.

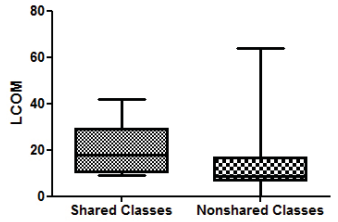


Fig. 3. Boxplot for RQ2 results.

RQ2. Figure 3 depicts the boxplot drawn for shared classes of **similar** use cases compared to other classes. Table VI shows that the cohesion of shared classes are **lower** than the cohesion of the other classes.

TABLE VI. Mann-Whitney Test for RQ2 - LCOM - Similar.

Feature	Value
P value	0.0167
One or two-tailed P value?	one-tailed
Are medians signif. different? (P < 0.05)	**Yes**
Mann-Whitney U	72.50

RQ3. Figure 4 depicts the boxplot drawn for shared classes of **dissimilar** use cases compared to other classes. Table VII shows that the cohesion of shared classes is **lower** than the cohesion of the other classes.

RQ4. Figure 5 depicts the boxplot drawn for shared classes of **dissimilar** use cases compared to other classes. Table VIII shows that the complexity measured by WMC of shared classes is **higher** than the complexity of the other classes.

RQ5. Figure 6 depicts the boxplot drawn for shared classes of **similar** use cases compared to other classes. Table IX shows that the there is **no significant difference** between the coupling measured by CBO of shared classes and the coupling of the other classes.

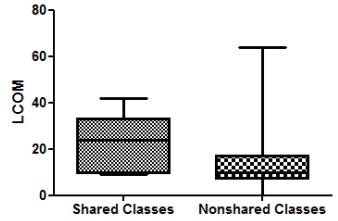


Fig. 4. Boxplot for RQ3 results.

TABLE VII. MANN-WHITNEY TEST FOR RQ3 - LCOM - DISSIMILAR.

Feature	Value
P value	0.0317
One or tow-tailed P value?	one-tailed
Are medians signif. different? (P < 0.05)	**Yes**
Mann-Whitney U	64.50

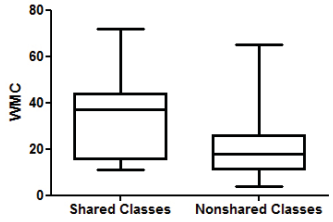


Fig. 5. Boxplot for RQ4 results.

TABLE IX. MANN-WHITNEY TEST FOR RQ5.

Feature	Value
P value	0.3090
One or tow-tailed P value?	One-tailed
Are medians signif. different? (P < 0.05)	**No**
Mann-Whitney U	118.0

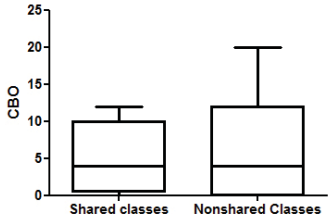


Fig. 7. Boxplot for RQ6 results.

RQ6. Figure 7 depicts the boxplot drawn for shared classes of **dissimilar** use cases compared to other classes. Table X shows that the there is **no significant difference** between the coupling measured by CBO of shared classes and the coupling of the other classes.

Therefore, we can see that the coupling of shared classes between use cases do not depend on the use case similarity.

RQ7. Figure 8 depicts the boxplot drawn for shared classes of **similar** use cases compared to shared classes of **dissimilar** use cases. Table XI depicts that there is **no significant difference** between the cohesion in those classes.

The final result was that we could answer positively RQ1, RQ2, RQ3 and RQ4. However, contrary to our hypotheses, RQ5, RQ6 and RQ7 were answered negatively.

V. DISCUSSION

Our proposed automated analysis for use case similarity had a precision of 90% at 60% of recall. This result support our hypothesis that the identification of similar use case can be automated. The major benefit of this automated approach is higher productivity during this process. This is essential to engage developers in the adoption of the approach.

Another interesting finding was that the higher the similarity of use cases is, the higher the LOC sharing between the classes that implement those use cases. This LOC sharing is not necessarily related to code clones, but related to the number of classes that are shared among the use cases.

Concerning the relation of shared classes and design metrics, some interesting findings are presented.

We have observed that independently of the use case similarity, classes that are shared between different use cases tend to have lower cohesion. In this case, we need to have special attention to these classes and analyze the possibility of refactoring those classes. In fact, we can even suggest that during the development phase, developers pay more attention on classes being shared between different use cases.

We also have observed that the complexity measured with WMC of classes shared between dissimilar use cases tend to have higher complexity than classes shared between similar use cases. Just as in the case of cohesion, developers should pay attention on those classes and eventually try to find some pattern of refactoring that could be applied to those classes to enhance their overall WMC.

Concerning coupling of classes, we could not observe an

TABLE VIII. MANN-WHITNEY TEST FOR RQ4 - WMC.

Feature	Value
P value	0.0192
One or tow-tailed P value?	one-tailed
Are medians signif. different? (P < 0.05)	**Yes**
Mann-Whitney U	59.00

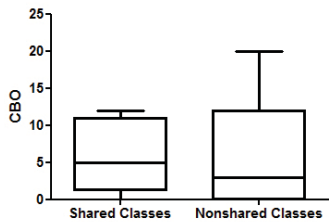


Fig. 6. Boxplot for RQ5 results.

TABLE X. MANN-WHITNEY TEST FOR RQ6.

Feature	Value
P value	0.4681
One or tow-tailed P value?	One-tailed
Are medians signif. different? (P < 0.05)	**No**
Mann-Whitney U	110.0

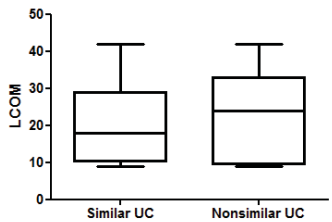


Fig. 8. Boxplot for RQ7 results.

TABLE XI. MANN-WHITNEY TEST FOR RQ7.

Feature	Value
P value	0.5000
One or tow-tailed P value?	One-tailed
Are medians signif. different? (P < 0.05)	**No**
Mann-Whitney U	53.50

interesting pattern that could call the attention of developers.

VI. THREATS TO VALIDITY

The subject system used in this study have some features that could have some influence on the results. The use of *POJO*[2] classes, i.e., classes that have only attributes and *getters and setters*, and the use of domain-driven development [7] may have influence on the coupling metric (CBO). Other studies with different domains and architecture would strength the value of our results.

Another threat was the use of developers to map use cases to classes and to evaluate the similarity of use cases. Because this is a manual process, mistakes could have occurred. We tried to mitigate this threat replicating the work of the developers and getting the consensus from them. Another mitigation criterion was the selection of experts that had participated in the implementation and documentation of the system, thus their knowledge on the system would produce less error-prone results.

A threat to the external validity is the representativeness of the chosen subject system. This is a system that do not represent the large universe of software. As a consequence, our results are more likely to be valid with typical information management systems.

VII. CONCLUSION AND FUTURE WORK

In this paper, we have studied the impact that the similarity of use cases can have in LOC sharing of classes that implement them, and in metrics for coupling, cohesion and complexity. We found that there is a significant relation, although not strong, between use case similarity and LOC sharing. Interestingly, we have also found that classes that are shared between different use cases tend to have lower cohesion. Moreover, classes that are shared between similar use cases tend to have higher method complexity in classes. We found no relation between use case similarity and coupling. These findings can guide the developer, either in early development phases or in maintenance activities, to produce designs with higher cohesion and lower method complexity in classes.

We could extend this work in several ways, such as, reproducing the study with other subject systems and investigating the relation between use case similarity and code clones.

ACKNOWLEDGMENT

This work was partially supported by FAPEMIG grant CEX-APQ-0286-11 and CNPQ grant 475519/2012-4.

[2]http://docs.jboss.org/hibernate/orm/3.5/reference/pt-BR/html/persistent-classes.html

REFERENCES

[1] F. B. Abreu and R. Carapuça, "Object–Oriented Software Engineering: Measuring and Controlling the Development Process," in *Object–Oriented Software Engineering: Measuring and Controlling the Development Process*, McLean, VA, USA, oct 1994, pp. 1–8.

[2] K. H. Bennett and V. T. Rajlich, "Software Maintenance and Evolution: A Roadmap," in *Proceedings of the Conference on The Future of Software Engineering*, ser. ICSE '00, 2000, pp. 73–87.

[3] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *Software Engineering, IEEE Transactions on*, vol. 20, no. 6, pp. 476–493, jun 1994.

[4] W. W. Cohen, P. Ravikumar, and S. E. Fienberg, "A comparison of string distance metrics for name-matching tasks," 2003, pp. 73–78.

[5] I. Dagan, L. Lee, and F. C. Pereira, "Similarity-Based Models of Word Cooccurrence Probabilities," *Machine Learning*, vol. 34, pp. 43–69, 1999.

[6] S. K. Dubey and A. Rana, "Assessment of maintainability metrics for object-oriented software system," *SIGSOFT Software Engineering Notes*, vol. 36, no. 5, pp. 1–7, 2011.

[7] E. Evans, *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Pearson Education, 2004.

[8] P. Grubb and A. A. Takang, *Software Maintenance: Concepts and Practice*, 2nd ed. World Scientific, 2003.

[9] R. Harrison, S. Counsell, and R. V. Nithi, "An Investigation into the Applicability and Validity of Object-Oriented Design Metrics," *Empirical Software Engineering*, vol. 3, no. 3, pp. 255–273, 1998.

[10] J. Hayes, S. Patel, and L. Zhao, "A Metrics-Based Software Maintenance Effort Model," in *Eighth European Conference on Software Maintenance and Reengineering, 2004.*, March, pp. 254–258.

[11] I. Jacobson, M. Christerson, P. Jonsson, and G. Övergaard, *Object-Oriented Software Engineering - A Use Case Driven Approach*. Addison-Wesley, 1992.

[12] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions," and reversals. Technical Report 8, Tech. Rep., 1966.

[13] M. Lorenz and J. Kidd, *Object-Oriented Software Metrics: A Practical Guide*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1994.

[14] T. McCabe, "A complexity measure," *Software Engineering, IEEE Transactions on*, vol. SE-2, no. 4, pp. 308–320, 1976.

[15] T. G. Nair and R. Selvarani, "Defect proneness estimation and feedback approach for software design quality improvement," *Information and Software Technology*, vol. 54, no. 3, pp. 274–285, 2011.

[16] J. T. Nosek and P. Palvia, "Software Maintenance Management: Changes in the Last Decade," *Journal of Software Maintenance*, vol. 2, no. 3, pp. 157–174, 1990.

[17] S. Olbrich, D. S. Cruzes, V. Basili, and N. Zazworka, "The evolution and impact of code smells: A case study of two open source systems," in *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 390–400.

[18] S. R. Pressman, *Software Engineering - A Practitioner's Approach*, 7th ed. McGraw Hill, 2005.

[19] R. Shatnawi and W. Li, "The effectiveness of software metrics in identifying error-prone classes in post-release software evolution process," *Journal of Systems and Software*, vol. 81, no. 11, pp. 1868–1882, 2008.

[20] S. Singh and K. S. Kahlon, "Effectiveness of refactoring metrics model to identify smelly and error prone classes in open source software," *SIGSOFT Software Engineering Notes*, vol. 37, no. 2, pp. 1–11, 2012.

[21] R. Subramanyam and M. S. Krishnan, "Empirical Analysis of CK Metrics for Object-Oriented Design Complexity: Implications for Software Defects," *IEEE Transactions on Software Engineering*, vol. 29, no. 4, pp. 297–310, 2003.

[22] W. E. Winkler, "The State of Record Linkage and Current Research Problems," Statistical Research Division, U.S. Census Bureau, Tech. Rep., 1999.

Generation of Thematic Maps using WPS-Cartographer: An experimental study

Francisco Carlos M. Souza, Alinne C. Corrêa dos Santos, Vinicius Pereira, Ellen Francine
Institute of Mathematics and Computer Science – ICMC
University of São Paulo – USP
São Carlos, Brazil
{fcarlos, alinne, vpereira, francine}@icmc.usp.br

Vinícius Ramos Toledo Ferraz
Computing Department
University Federal of São Carlos – UFSCar
São Carlos, Brazil
vinicius_ferraz@dc.ufscar.br

***Abstract*— Presently, most geographic information systems provide neither automation nor guidance on the selection of an adequate cartographic language to produce a thematic map. This paper reports an experiment to evaluate a Web service, called WPS-Cartographer that is used for the semiautomatic selection of cartographic symbols, to analyze and evaluate time and quality of the generated thematic maps. The experiment was carried out with either with MSc. Students and PhD. students of computer science without experience in thematic maps generation. The results show that the use of WPS-Cartographer did not influence the generation of thematic maps regarding to time. However, WPS-Cartographer was efficient with respect to the quality of the generated maps, because the possibility of selecting of the most appropriate mapping technique for a given domain and dataset.**

Keywords- experimentation; thematic maps; WPS-Cartographer

I. INTRODUCTION

Thematic maps can visually summarize and present large volumes of analytical data, making easier and more agile complex analyzes of geospatial phenomena [1]. Geographic Information Systems (GIS) help produce these maps, activity known as Thematic Cartography Project (TCP).

The simple appearance of a thematic map hides some difficulties of the users related to its composition as the correct selection of the cartographic parameters, the graphic design, clear visualization of the forms and patterns, as well as the constant search for updated information.

Currently, most of the GIS available provide a range of tools and options to generate Thematic Maps, including automation functionalities in the acquisition, standardization and availability of thematic geospatial data. However, these systems do not provide automation features and guidelines to help the choice of a cartographic language that composes a Thematic Map.

However, despite several researches [2], [3], [4] e [5] addressing possible solutions to restrict or direct appropriate decisions for TCP, they do not have automation functionality for TCPs. Though, Ferraz [6] proposed a Cartographer Methodology based on Cartographer Ontology that is a process to increase the level of automation of TCPs assisting a user in choosing the appropriate symbols.

To corroborate the Cartographer Methodology Ferraz [6] developed a Web service, called WPS-Cartographer[1] based in this methodology for semiautomatic selection of cartographic symbology. Thus, this paper describes an experiment to analyze and evaluate the impact of the use of WPS-Cartographer for thematic maps generation in relationship at the time and quality.

This paper is organized as follows: Section II shows the design of the experiment in detail. Section III provides the results and discussions concerning the experiment. Finally, Section IV concludes the paper and describes future work.

II. EXPERIMENT

The experiment aimed at analyzing and evaluating whether the use of service WPS-Cartographer improves the time and quality of the generated maps in comparison with the classical method. The experiment was conducted with 18 master's and PhD students at ICMC – USP without experience in thematic maps generation. The students were randomly divided into two groups (G1 and G2).

The experimental format this experiment was defined with one factor, two treatments with comparison in pairs. The factor is thematic maps generation and the treatments are: *(i)* T1 (using the WPS-Cartographer) and *(ii)* T2 (using classical method). Furthermore, both groups have performed both treatments. Process experimental is composite for a sequence of activities, which are detailed below.

A. Definition

To achieve the goal proposed, two questions were developed to guide the definition (Table I) and implementation of the experiment:

Q1: Are thematic maps generated faster using WPS-Cartographer?

Q2: Are thematic maps generated with better quality using WPS-Cartographer?

TABLE I. DEFINITION OF THE EXPERIMENT

CHARACTERISTICS	DESCRIPTIONS
Objective of Study	Analyze the WPS-Cartographer
With the purpose of	Perform an evaluation
With respect to the	Time and quality
From the perspective of	Postgraduate students
In the context of	Generate thematic maps

Furthermore, dependent and independent variables were defined based in objectives and in research questions. Dependent variables are composite for using

1 *http://www.opengeospatial.org/standards/wps*

WPS-Cartographer and using classical method. Independent variables are composite for time and quality. The variables were measured using the following metrics:

- **Time Spent:** represents the amount of time spent using or without WPS -Cartographer;
- **Quality:** represents the quality of the maps generated. The quality will be calculated based on the scores attributed to predefined criteria. The weight of each criterion changes according to the topic, which can be seen in detail in the work of Ferraz [6]. The quality is calculated by:

$$\mathbf{Q = (Type * 5) + Classification* 2 + Layout + Names = Values\ (Topics\ 1\ and\ 2)} \quad (1)$$

where the sum of all criteria in the formula results in scores, which define quality index, ranging from 0 to 100 points.

B. Planning

This step defines instrumentation and experiment validation process. Firstly six hypotheses were defined for the experiment regarding to time and quality with using WPS-Cartographer and using classical method. Concerning the time spent of the thematic maps generated, the following hypotheses were formulated:

- **Null hypothesis (H0):** in general, there is no impact on the use of WPS-Cartographer with respect to the average time (µT) required for the thematic maps generation → (***H0:*** *µT-Cartographer = µT-Classical);*
- **Alternative hypothesis (AH1):** the average time (µT) using the WPS-Cartographer is shorter than the average time (µT) spent using the classical method of thematic maps generation. → (***AH1:*** *µT-Cartographer < µT-Classical);*
- **Alternative hypothesis (AH2):** the average time (µT) using the WPS-Cartographer is longer than the average time (µT) spent using the classical method of thematic maps generation. → ***AH2:*** *µT-Cartographer > µT-Classical);*

Regarding quality (Q) of the thematic maps generated the following hypotheses were formulated:

- **Null hypothesis (H0):** in general, there is no impact on the use of WPS-Cartographer, with respect to the average quality (µQ) of thematic maps generated → (***H0:*** *µQ-Cartographer = µQ-Classical);*
- **Alternative hypothesis (AH1):** the average quality (µQ) using the WPS-Cartographer is higher than the average quality (µQ) achieved using the classical method of thematic maps generation. → (***AH1:*** *µQ-Cartographer > µQ-Classical);*
- **Alternative hypothesis (AH2):** the average quality (µQ) using the WPS-Cartographer is higher than the average quality (µQ) achieved using the classical method of thematic maps generation. → (***AH2:*** *µQ-Cartographer > µQ-Classical).*

To validate the experiment measure descriptive statistics such as mean, median and standard deviation were calculated. Furthermore, the hypothesis tests were conducted using Shapiro-Wilk [7], Levene [8], T [9] and Mann-Whitney test [10] to analyze, interpret and validate results.

The participants were randomly divided into two groups, to participate in two executions: *(i)* pack I (G1 generated the thematic maps using the WPS-Cartographer and G2 generated the thematic maps using classical method) and *(ii)* pack II (G1 generated the thematic using classical method and G2 generated the thematic maps using the WPS-Cartographer).

The IBM SPSS[2] tool helped in analysis and interpretation the collected data and in hypotheses verification. With the help this tool were calculated the samples normality, homoscedasticity, comparison of averages and unpaired groups in order to check whether they belong or not to the same population. The discussion of the analyses and interpretations will be detailed in the next section.

III. Results and Analysis

During the experiment 36 thematic maps were generated in parallel by both groups (G1 and G2). The first analysis was performed by using descriptive statistics, followed by hypothesis tests to validate the analysis and interpretation of data collected regarding time and quality of the generated maps.

A. In relation time

The results regarding the time in generation of thematic maps of the Packs I and II are illustrated in Figure 1 and 2 respectively. Time was calculated in seconds. Figure 2 shows the group G1 (participant 1 to 9) and the group G2 (participant 10 to 18). As can be seen in relation to the average time (µT), in most cases, the group G1 spent lesser time than the group G2 in the generation of thematic maps.

In most situations in Pack I, we observed little difference in time between the groups G1 and G2. Furthermore, it can be seen that the average time spent in thematic maps generation by G1 is lesser than time spent by group G2. Figure 3 shows the group G2 (participants 10 to 18) and the group G1 (participant 1 to 9) in Pack II.

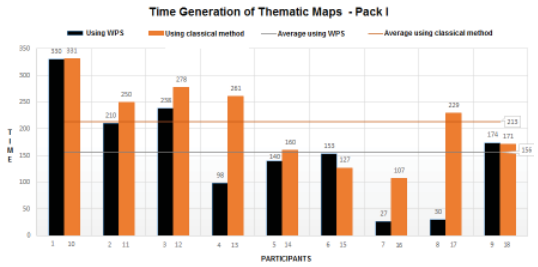


Figure 1. Average time (in seconds) in the generation of thematic maps using of WPS-Cartographer (G1) and using classical method (G2) of pack I

In Pack II, as can be seen in relation to the average time (µT), in most cases, the group G2 spent lesser time than the group G1 in generation of thematic maps. However, in most cases we observed more expressive difference in time between some participants (10 and 1), (12 and 3) and (16 and 7). Furthermore, it can be seen that the average time spent in thematic maps generation by group G1 is lesser than time spent by group G2.

To validate the analyze of data and justify the results were carried out statistical tests. First, the Shapiro-Wilk test was applied in the packs I and II .

[2] *http://www-01.ibm.com/software/analytics/spss/*

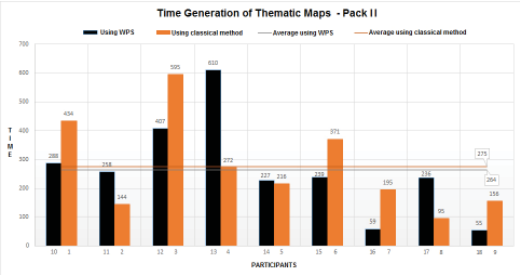


Figure 2. Average time (in seconds) in the generation of thematic maps using of WPS-Cartographer (G2) and using classical method (G1) of pack II

In pack I, we checked that the p-value (Sig 0,831) of the G1 and p-value (Sig. 0,785) of the G2 are higher than 0,05 (significance level), so the null hypothesis (H0: the data has a normal distribution) was not rejected. Thus, one can conclude that with 95% confidence level there is evidence that the data follow a normal distribution. In pack II, the data follow a normal distribution, since the p-value (Sig. 0,334) of the G1 and p-value (Sig. 0,241) of the G2 are higher than 0,05 (significance level).

After the analysis of normality, we applied the Levene's test to check the homoscedasticity of the samples. In pack I, we checked that the p-value (Sig. 0,671)> 0.05, so the null hypothesis (H0: variables have similar variances) was not rejected. Thus, one can conclude that with 95% confidence level there isn't evidence of the equality of variances.

In pack II, there is also evidence of equal variances, since the p-value (Sig. 0,785) > 0.05. The T test was applied to in normal distributions (G1 and G2) in relation to the time of the packs I and II.

In packs I and II, the T test results used are in the first column (Equal variances assumed). Considering p-value (0,184) > 0.05 (Sig 2-tailed) in pack I, so the null hypothesis (***H0:*** *μT-Cartographer = μT-Cl ssico)* for equality of means of the two groups (G1 and G2) is rejected.

Thus, one can conclude that there is no statistically significant difference between the mean times of the two groups. In pack II, there is no also statistically significant difference between the mean times of the two groups, considering that p-value (0,890)> 0.05 (Sig 2-tailed).

B. In relation to quality

The results related to quality in generation of the thematic maps of Packs I and II are illustrated in Figure 3 and 4 respectively. In Figure 3, in relation to the average quality (μQ), in most cases, the thematic maps generated by group G1 have achieved better quality than the group G2.

In most situations in Pack I, we observed little difference quality in the thematic maps generated between the groups G1 and G2, with the exception of some participants (4 and 13) and (9 and 18). Figure 4 shows results of pack II. As can be seen in relation to the average quality (μQ), in most cases, generated thematic maps by G2 have better quality than generated by G1.

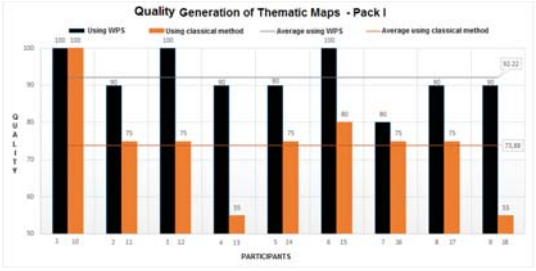


Figure 3. Average quality (in points) in the generation of thematic maps using of WPS-Cartographer (G1) and using classical method (G2) of pack I

In most situations in Pack II, we observed that the difference in quality between the groups G1 and G2 was not very expressive, with the exception of some participants (13 and 4), (16 and 7) and (17 and 8). Furthermore, such as in the pack I, in pack II there were no cases in which the group G2 have achieved lesser quality than group G1 in the generation of thematic maps.

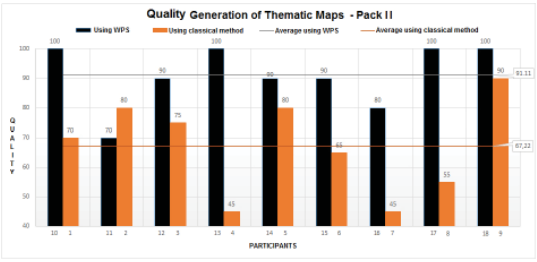


Figure 4. Average quality (in points) in the generation of thematic maps using of WPS-Cartographer (G2) and using classical method (G1) of pack II

The Shapiro-Wilk test was also applied in the packs I and II. In pack I, we checked that the p-values (Sig 0.028) and (Sig. 0.047) > 0.05, so the null hypothesis (H0: the data has a normal distribution) was rejected. Thus, one can conclude that with 95% confidence level there isn't evidence that the data follow a normal distribution.

In pack II, the data did not follow a normal distribution, because the p-values (Sig. 0.039) and (Sig. 0.049) <0.05. After the analysis of normality, Levene's test was applied.

In the pack I, we checked that the p-value (SIG. 0.382) > 0.05, so the null hypothesis (H0: the variables have similar variances) was not rejected. Thus, one can conclude that with 95% confidence level there is evidence of equality of variances. In pack II, there is also evidence equality of variances because the p-value (Sig. 0.149) > 0.05.

The Mann-Whitney test was applied because the distributions of both packages in relation to quality were not normal. In pack I, considering the p-value (Asymp. Sig. (2-tailed)) 0.03 < 0.05 the null hypothesis of equality of means of two groups (H0: Cartographer-μQ = μQ-Classical) was rejected. Thus, one can conclude that there is a difference statistically significant between the groups G1 and G2.

In pack II, there is also a difference statistically significant between the two groups (G1 and G2), considering that p-value (Asymp. Sig. (2-tailed)) 0.02 < 0.05.

C. Relationship between time and quality

The generation of thematic maps also was analyzed graphically by relationship between time and quality. The results of packs I and II are illustrated in Figures 5 and 6, respectively.

In Figure 5 is evident that the group G1 generates maps with higher quality and in lesser time. Most participants of the group G1 generated thematic maps with quality equal to or above 90 points of the total of 100 points in the mean time (μT) of 156 seconds. Already the quality of the thematic maps generated by group G2 was not high, because 7 of 9 participants generate maps with quality lesser to 80 points in mean time (μT) of 213 seconds.

Furthermore, should note that 3 participants of the G1 generate maps with quality equal to 100 points, whereas only one participant of the G2 generated a map with quality equal to 100 points.

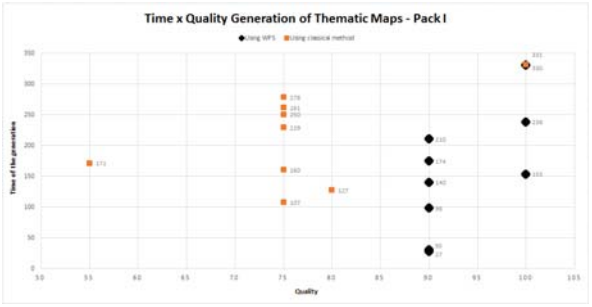


Figure 5. Time x Quality in generation of thematic maps using WPS-Cartographer (G1) and using classical method (G2) of the pack I

Such as in Figure 5, in Figure 6 is also evident that the maps are generated with high quality in an acceptable time using WPS-Cartographer. In pack II, 5 participants of the G2 thematic maps generated with quality equal to 75 points ranging between 220 and 410 seconds. The quality of the maps generated by G1 was not high, because 2 participants generate maps with quality equal to 75 points and with time above 400 seconds.

It is important highlight that 7 of the 9 participants of the G2 generate maps whose quality was equal to or above 75 points. However, three participants of the G1 reached 80 points in maximum 216 seconds. It is evident that in both packages the use of WPS-Cartographer influences more the quality than time in the generated maps.

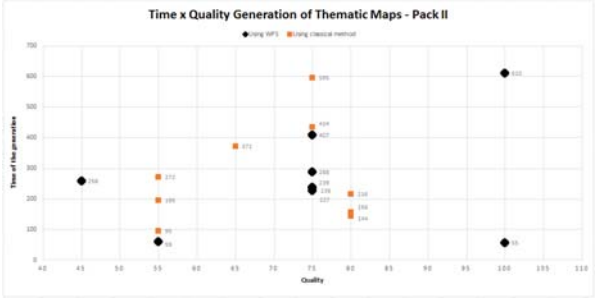


Figure 6. Time x Quality in generation of thematic maps using WPS-Cartographer (G2) and using classical method (G1) of the pack II

IV. Results and Analysis

This paper describes an experiment to analyze and evaluate the impact of the use of WPS-Cartographer for thematic maps generation in relationship at the time and quality. In relation to threats of validity, some can endanger the validity of this experiment such as:

- The selection of the treatments and sample size can influence the interpretation of the analysis and statistical tests;
- The use WPS-Cartographer did not influence significantly the generation of thematic maps, so there is no time savings using WPS-Cartographer;
- Achievement of a pilot study to selection of the most appropriate metrics and minimization of problems in relation to operating systems (OS/X and Linux), browsers (Internet Explore and Google Chrome);
- The results cannot be generalized because this experiment was conducted with a relatively small group of 18 lay students in the thematic maps generation.

At the end of the experiment it was observed that the use of WPS-Cartographer positively influenced the final quality of the thematic maps generated, being significantly better in relation to time. As future work we hope to perform new experiments with industry experts to verify the applicability of the WPS-Cartographer to understand this new domain and compare with the results obtained in this experiment.

Acknowledgments

The authors would like to thank the Brazilian funding agencies (FAPESP, CAPES, CNPq) and INCT-SEC (CNPq 573963/2008-8, FAPESP 08/57870-926) for their support.

References

[1] Monmonier, M.S. How to Lie with Maps. Chicago: The University of Chicago Press, 1996.

[2] ZHAN, F. B. and BUTTENFIELD, B. P. Object-oriented knowledge-based symbol selection for visualizing statistical information. International Journal of Geographical Information Science, v.9 (3), p.293–315, 1995.

[3] LEE, D. and HARDY, P. Automating generalization-toolsandmodels. In: Proceedings of International CartographicCongress. ACoruna, Spain, 2005.

[4] Shepherd, I. D.H. Travails in the third dimension: A critical evaluation of three dimensional geographical visualization. In: 111 River Street, USA: Wiley, 2008.

[5] SLOCUM, T. A. et al. Thematic Cartography and Geographic Visualization. 2. ed. Upper Saddle River, NJ: Pearson Prentice Hall, 518 p., 2005.

[6] Ferraz, V. R. T. and Santos, M. T. P. Globeolap: Improving the geospatial realism in multidimensional analysis environment. In: Proceedings of the 12th International Conference on Enterprise Information Systems. Funchal, Madeira – Portugal, 2010.

[7] Shapiro, S. S. and Wilk, M. B. An analysis of variance test for normality (complete samples). Biometrika 52 (3-4), p. 591–611, 1965.

[8] Levene, H. Robust tests for equality of variances. In Ingram Olkin, Harold Hotelling, et alia. Stanford University Press. pp. 278–292, 1960.

[9] Teste T. Disponível em: http://sistemas.eeferp.usp.br/myron/arquivos/2540410/e8fc3b72347400901a2750cb214bf4e0.pdf. Acessado em: 20/11/2012.

[10] Mann, H.B. and Whitney, D. R. Em um teste para saber se uma de duas variáveis aleatórias é estocasticamente maior que o outro. Annals of Mathematical Statistics 18 (1) pp. 50-60, 1947.

Automated Support for Controlled Experiments in Software Engineering: A Systematic Review

Marília Freire,[a,b] Daniel Costa,[a] Edmilson Campos,[a, b] Tainá Medeiros,[a]
Uirá Kulesza,[a] Eduardo Aranha,[a] Sérgio Soares,[c]
[a] Federal University of Rio Grande do Norte, Campus Universitário, 59.078-970, Natal-RN, Brasil
[b] Federal Institute of Rio Grande do Norte, Av. Salgado Filho, 1559, Tirol, 59015-000, Natal-RN, Brasil
[c] Federal University of Pernambuco, Av. Prof. Moraes Rego, 1235, C. Universitária, 50670-901, Recife-PE, Brasil
{marilia.freire, daniel.calencar , edmilsoncampos, tainamedeiros}@ppgsc.ufrn.br, {eduardo, uira}@dimap.ufrn.br,
scbs@cin.ufpe.br

***Abstract*— *Context*: There is an increasing need to perform controlled experiments in software engineering. *Objective*: This systematic review (SR) shows the current state of the art on the tools and infrastructures that provide automated support for controlled experiments in software engineering. *Method*: We performed direct searches (without search string) in journals and conferences proceedings for papers describing supporting tools and environments to conduct controlled experiments. *Results*: We found and reviewed 25 primary studies according to inclusion and exclusion criteria, resulting in 15 relevant studies. *Conclusion*: There are few supporting environments for conducting controlled experiments, despite of the increasing demand for this kind of study in software engineering. We also highlight many limitations of these tools, which configures great potential for future research.**

Experimental software engineering; automated support; controlled experiment.

I. INTRODUCTION

In recent years, the software engineering research community has given more attention and importance to the development and reporting of experimental studies, considering that simple proof of concept is no longer acceptable in the assessment of new proposed methods. One important type of empirical study for the research in software engineering is the controlled experiment [1].

A controlled experiment allows testing research hypotheses and cause and effect relationship between variables involved in the study. The process of a controlled experiment is typically composed of the following phases: definition, planning, execution, analysis, and packaging [2]. Each one of these phases is associated with the execution of different activities that consume and/or produce artifacts related to the experiment. Controlled experiments require great care in planning so they can provide useful and significant results [3]. However, the process of planning, conducting, and reporting the various activities involved in a controlled experiment is very complex [4] .

Despite the growing need to run controlled experiments in software engineering, their development is still very complex. Furthermore, controlled studies need to be replicated because a single controlled experiment may be insufficient and their results are limited in terms of conclusions' generalization [5]. Conduction and replication of large-scale experimental SE studies is even more complex [6]. One factor that contributes to that is the lack of automated and integrated tools supporting the experiment process phases.

This paper focus on presenting the results of a systematic review [7] [8] that analyzes the current state of the art on the tools and infrastructures developed to provide automated support to conduct controlled experiments in software engineering. Our study has found seven tools specifically developed or adapted for conducting experiments in software engineering. Our systematic review aims to provide findings to appropriately points out new research efforts and opportunities related to development of automated support for the software engineering experimental process.

The remainder of this paper is organized as follows. Section II explains our review method. Section III presents our results. The limitations of our study are stated in Section IV. Section VI presents some discussions based on the data we got in our systematic review, and, finally, Section VII presents the conclusions and possible future works.

II. REVIEW METHOD

This section describes the protocol used to conduct this systematic review, which was defined based on specific guidelines [8]. The process was performed in the early 2012. Due to space restrictions some contents related to the protocol is not presented in this paper, for additional details please refer to: http://bit.ly/10zA6FY.

A. Research Questions

Our research was guided by questions about the empirical support tools and infrastructures that provide some automated support to the phases of controlled experiments in software engineering. The four main research questions (RQ) investigated in the systematic review were:

- **RQ.1** What tools have been proposed to support controlled experiments in software engineering?
- **RQ.2** Which stages of controlled experiments the proposed tools are supporting?
- **RQ.3** What are the main features supported by the proposed tools?

- **RQ.4** Who has been developing tools to support controlled experiments? When?

B. Inclusion/exclusion criteria

The inclusion criteria defined in our study were: (i) only studies written in English were considered, and (ii) only studies that were not related to specifics domains of SR were included, because we are interested in infrastructures for controlled experiments in general and not, for example, an environment to compare two specific algorithms. The exclusion criterion defined in our study was: (i) studies that did not present supporting environments for conducting controlled experiments in software engineering were excluded.

C. Decision Procedure

In a systematic review, it is important to define how to solve possible conflicting situations. These conflicts may happen during the study selection, quality evaluation, or data extraction. Therefore, we defined our process to support decision-making and consensus as follows: two members from our team read selected studies assigned in a random way. Any disagreement among researchers was resolved by a third reviewer (one professor).

D. Data Sources and Search Strategy

Our search strategy was established in two steps: (i) a manual search across the main publication vehicles in the Experimental Software Engineering area. By manual search we mean searching directly on publication vehicles without a search string; and (ii) a reference search where the reference section of all selected primary studies were analyzed searching for new research work related to our systematic review questions. For the first step the following conferences and journals were considered: Journal of Empirical Software Engineering, Journal of Systems and Software, ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM) (and the previous METRICS and ISESE), Experimental Software Engineering Latin American Workshop (ESELAW), and International Conference on Software Engineering (ICSE). It is important to say that for publication sources as ICSE, Journal of Empirical Software Engineering, and Journal of Systems and Software, we limited the search period from January/2002 to December/2011, because these events are already made long ago and we would not have time to check them out. However, for the analysis of referred papers we do not consider any restriction regarding publication venues and dates.

E. Study Selection

The study selection process was realized in two steps as follows: (i) titles and abstracts of papers found during the manual search were read and irrelevant papers were removed; (ii) the complete reading of the selected primary studies was performed to assess whether they address the inclusion/exclusion criteria.

F. Data Extraction Process and Synthesis

As usual, each research question motivated some data extraction (see Table I). In addition, general information was extracted from the studies, such as title, authors, publisher, and publication year. The data extraction was also performed with the aid of spreadsheets containing forms to extract portions of the selected studies.

TABLE I. EXTRACTED DATA

Research Question	Attribute	Data
RQ.1	Tool	study title, tool name, origin (academic or industrial), tools compared
RQ.2	Process	study stages supported
RQ.3	Features	functionalities supported
RQ.4	Mapping	first author's affiliation and country

III. RESULST AND FINDINGS

In this section we show the results of our study addressing the research questions presented in Section II. Our systematic review was interested in primary studies that present some automated solutions to support software engineering experiments.

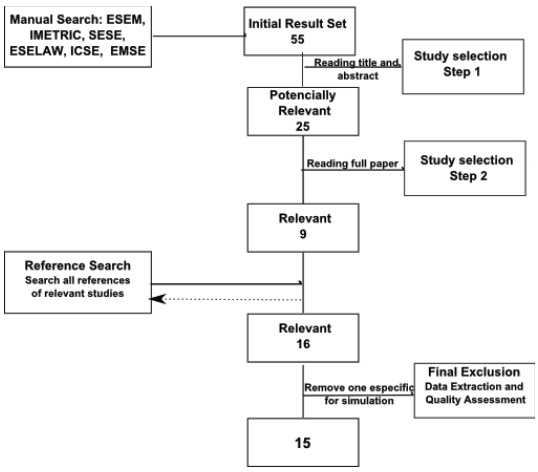


Figure 1. Studies Search and Selection Process

Figure 1 illustrates the systematic review process showing the primary studies that were found and selected. During the manual search 55 papers were selected from the venues previously specified. For this selection, the titles and abstracts of the paper studies were read resulting in 25 papers. These 25 papers were completely read in order to discard irrelevant studies. After this selection, we had 9 studies. After that, a search on the reference sections of the papers was accomplished aiming the selection of other relevant studies. During this process we found other 7 primary studies. In the final step we removed studies that were not specific to a single domain of SE. The final number of relevant studies was 15.

During the search on the reference sections, we found papers from other conferences: IEEE TSE (IEEE Transactions on Software Engineering), Advances in Computing Journal, NJC (Nordic Journal of Computing), NWPE (Nordic Workshop on Programming and Software Development Tools and Techniques), JIISIC (Jornadas IberoAmericanas en Ingenieria del Software e Ingenieria del Conocimiento), and ICECCS (International Conference on Electronics, Communication and Computer Science). The results found for each research question are discussed in the following sections.

A. Tools to support controlled experiments (RQ.1)

The following seven environments – tools, infrastructure – were found: Simula Experiment Support Environment (SESE), experimental Software Engineering Environment (eSEE), value-based empirical research (VBER), Ginger2, Experiment Manager framework (EMF), Framework for Improving the Replication of Experiments (FIR), and Mechanical Turk (MT). Following, we have a short summary of the seven tools and the respective ID according to the Table 2:

SESE [9] [10] [11] [12]: It is a web-based tool that supports participants management, capturing the time spent during the experiment, enable the work product collection, and participants activities monitoring. Its weaknesses are the data collection and analysis (PS1, PS2, PS3, PS4).

eSEE [13] [14] [15] [16] [6] [17]: An environment to manage several kinds of empirical software engineering studies. It works in three levels of knowledge organization about the experimentation process: knowledge for any kind of experimental study (meta level), knowledge for each experimental study (configuration level), and knowledge for a specific experimental study (instance level). It has a prototype and an initial set of tools to populate the eSEE infrastructure has been built (PS5, PS6, PS7, PS8, PS9, PS10).

VBER [18]: A value-based framework to the planning phase. It assists the stakeholders to compare the benefits and risks of potential empirical study variants (PS3).

Mechanical Turk (MT) [19]: A crowdsourcing tool adapted to support empirical studies in the experimental software engineering context. It offers facilities to access and manage a large pool of study participants and enables recruiting the right type and number of subjects to assess a software engineering technique or tool (PS11).

Ginger2 [20]*:* An experimental environment constructed based on the CAESE framework. Although the CAESE covers the complete processes, Ginger2 is restricted to the execution and analysis phase. Its strength is the variety of low-level detail collected (PS15).

FIRE [21]: A Framework for Improving Replication of Experiments that focuses on knowledge sharing issues to allow cooperation between research groups. Fire is a framework with seven steps that assumes researchers are collaborating closely using a variety of communication mechanisms. Its weakness is that it gives only a conceptual support (PS13).

Experiment Manager Framework (EMF) [22]: This framework is an integrated set of tools to support software engineering experiments. It was used only in high performance computing (HPC) experiments. It helps the subjects by applying heuristics to infer programmer activities. Its analysis tools are simple (PS14). Table 2 presents the 15 studies included in this review, including the study ID, title, publisher year, name of related supporting environment (tool, framework or infrastructure), university or industry that developed the environment, and the publisher source.

For the studies that are part of a collaborative work we fill in the table with the first author information. Another point to clarify is that the Feedback-collecting tool described in PS2 is implemented as part of the web-based Simula Experiment Support Environment (SESE).

Although PS15 was published in 1999, we included it in our revision after the reference search because it presents the Ginger2 that is a tool based on the CAESE (Computer-Aided Empirical Software Engineering) framework. This framework defines a complete solution for conducting experiments but, according to the study, Ginger2 have been implemented to only support the data collection and analysis stages. A major feature of these tools is allowing the collection of several empirical data as mouse clicks and keystrokes, eye traces, three-dimensional movement, skin resistance level, and video-taped data. We were not able to find more details of the current development stage of these solutions. The eSEE tool is the only one that reports a web site, although we were not able to find out updated information from it.

B. Supported Stages of Controlled Experiments (RQ.2)

This question aims to point the main experimental process stages that are supported by the investigated tools. A typical controlled experiment process has the following stages [2]:

1) Definition: In this phase the study has to be characterized in terms of problem, objective and goals. It determines the foundation for the experiment.

2) Planning: It prepares for how the experiment is conducted. It comprises the hypothesis formulation, variables selection (dependent and independent), selection of subjects and, design of experiment determination. It also considers the threats to experiment evaluation.

3) Operation (execution): It follows from the design. It comprises: (i) the study configuration (preparation), where participants are chosen and the materials are prepared; and (ii) the execution that collects the data that should be analyzed.

4) Analysis and Interpretation: Responsible by the compilation of collected study data. It comprises descriptive statistics, data set reduction and hypothesis testing.

5) Presentation and Package: In this stage the information about the study is presented and the package is generated. It is essential for the study replication.

Each stage defines activities to be accomplished and specific work products. Table 3 presents the stages covered for each experimental environment found in our systematic review. Almost all empirical environments selected in our study give some kind of support to the controlled experiment definition, planning, and execution phases. The analysis stage is only supported by two of them. Moreover, four of the studied environments give support to the packaging step, which is important for replication, but none of them defines an explicit format or pattern to package experiments. CAESE framework is the only that mentioned supporting the complete process but the Ginger2 does not implement all these phases

C. SE Empirical Tools Functionalities Supported (RQ.3)

It is fundamental to understand the level of support provided by each different tool for each different stage of the process. Table 4 shows some features described by the primary studies.

TABLE II. SELECTED DATA

ID	Title	Year	Reported Tool	University/Industry	Publish Source
PS1	Conducting realistic experiments in software engineering	2002	Web-based Simula Experiment Support Environment (SESE)	Simula Research Laboratory	ISESE
PS2	Collecting Feedback During Software Engineering Experiments	2005	Feedback-collecting tool	Simula Research	ESEM
PS3	A Web-based Support Environment for Software Engineering Experiments	2002	Simula Experiment Support Environment (SESE)	Simula Research Laboratory + KompetanseWeb AS	NJC
PS4	SESE – an Experiment Support Environment for Evaluating Software Engineering Technologies	2002	Simula Experiment Support Environment (SESE)	Simula Research Laboratory + KompetanseWeb AS	NWPER
PS5	Infrastructure for SE Experiments Definition and Planning	2004	experimental Software Engineering Environment (eSEE)	COPPE/UFRJ	ESELAW
PS6	eSEE: a Computerized Infrastructure for Experimental Software Engineering	2004	experimental Software Engineering Environment (eSEE)	COPPE/UFRJ	ESELAW
PS7	A computerized infrastructure for supporting experimentation in software engineering	2005	experimental Software Engineering Environment (eSEE)	COPPE / UFRJ	ESELAW
PS8	Supporting Meta-Description Activities in Experimental Software Engineering Environments	2005	Meta-configurator from experimental Software Engineering Environment (eSEE)	COPPE/UFRJ	ESELAW
PS9	An environment to support large scale experimentation in software engineering	2008	experimental Software Engineering Environment (eSEE)	COPPE/UFRJ	ICECCS
PS10	Towards a Computerized Infrastructure for Managing Experimental Software Engineering Knowledge	2004	experimental Software Engineering Environment (eSEE)	COPPE/UFRJ	JIISIC
PS11	Exploring the use of crowdsourcing to support empirical studies in software engineering	2010	Mechanical Turk	University of Nebraska	ESEM
PS12	Value-Based Empirical Research Plan Evaluation	2007	value-based empirical research (VBER) planning framework	Vienna Univ. of Technol.	ESEM
PS13	A Framework for Software Engineering Experimental Replications	2008	FIRE - Framework for Improving the Replication of Experiments	Salvador University	ICECCS
PS14	An Environment for Conducting Families of Software Engineering Experiments	2008	Experiment Manager framework	University of Nebraska	Advances in Computers
PS15	Ginger2: An Environment for Computer-Aided Empirical Software Engineering	1999	CAESE Framework and Ginger2	Nara Institute of Science and Technology	IEEE TSE

TABLE III. EXPERIMENTAL PROCESS STAGES SUPPORTED PER TOOL

Process Stage	SESE	eSEE	VBER	MT	FIRE	Ginger2	EMF
Definition	■	■		■	■	■	
Planning	■	■	■	■	■	■	
Operation	■	■		■	■	■	■
Analysis and Interpretation						■	■
Presentation and Package	■	■			■	■	

TABLE IV. TOOLS FEATURES

Functionalities	Tools
Define the experiment (questionnaires, tasks descriptions, artifacts and roles)*	SESE, eSEE, Ginger2
Allow the researcher defining the kind and number of subjects that should take part in the experiment	SESE, eSEE
Allow the user (subject) fills in questionnaires and downloads task descriptions, and other required documents. User carries out the tasks, answers questions along the way and uploads the finished documents.	SESE, eSEE, MT
Do continuous timestamp for activities	SESE, MT., EMF, Ginger2
Monitor experiment (the progress of each subject)	SESE, eSEE
Collect and storage data	SESE, Ginger2, EMF
Store and publish the experimentation process in a process model repository	eSEE
Specify and visualize the Experimental Plan trough well-defined process	eSEE
Elaborate the documents produced/consumed throughout the Experimentation Process	eSEE
Make the experimental tasks available	eSEE, Ginger2, MT

Functionalities	Tools
Control the experiment	eSEE
Register lessons learned	eSEE
Gather feedback from subjects	SESE
Characterize the study (using GQM)	Ginger2, VBER
Support Experiment Design	Ginger2
Support Data Analysis	Ginger2, EMF.
Integrate data among different tools, integrate different control tools, integrate analysis tools	Ginger2
Support Packaging	eSEE, SESE, Ginger2, FIRE
Manage payment (to subjects)	MT
Address knowledge sharing issues both at the intra-group (internal replications) and inter-group (external replications)	FIRE
Frequent interaction among groups through e-mail and phone calls	FIRE
Execution of pilot studies	FIRE
Help identifying potential conflicts that indicate project risks	VBER
Elicit empirical study principal stakeholders (industry and academia) and their key value propositions expected	VBER

In an experimental process, one needs to choose the experimental study design. This choice determines how to organize (participants, experimental material, and treatments), to run the experiment, and to analyze the experiment collected data using a specific statistical analysis method. However, no tool details what support is given to set up the experiment design during the planning.

We also realized that analysis tools that are part of investigated infrastructures are relatively simple, except the Ginger2 that explicitly mentions the existence of internal tools to support observational and computational analysis. Another issue that is not addressed by the environments is how to define metrics to be collected during the experiment execution. Ginger2 mentions a Statistical Metrics Tool for data analysis that computes and returns various statistical values and metrics that have been defined by experimenters but it does not detail how it works.

Finally, although Ginger2 and SESE enable the experiment definition, they have a predetermined process that cannot be adjusted according to the needs of each new experiment.

D. Developing Tools to Conduct SE Experiments (RQ.4)

The purpose of RQ.4 is to investigate who are the researchers involved in the area of supporting infrastructures for controlled experiments and to investigate the evolution of the area. For example, we were interested in knowing if this kind of infrastructure has been recently developed.

In our systematic review, we have initially restricted to select only papers published between 2002 and 2010, when searching for the papers from the investigated conferences and journals. In the second step of our study, we included the studies found while searching the reference section of the primary studies even if they are outside of the established initial period, as established by our protocol. Our main aim was to try to capture a wide set of related research work. Our study results demonstrated that research on automated execution of SE controlled experiments was performed mainly over the last decade.

Table 5 shows the *country*, *affiliation* and the *distribution* of the selected studies per *affiliation*. One can observe that the selected studies were originated from five different countries. It is important to emphasize that some studies have researchers coming from different countries but we have only considered the first author affiliation and country. Among the selected studies, 47% comes from Brazil (mainly from COPPE/UFRJ) and other 27% comes from the Simula Research Laboratory/ Norway.

TABLE V. DISTRIBUTION OF STUDIES OVER AFFILIATION/COUNTRIES

Country	Affiliation	Dist.	Approach
Japan	Nara Institute of Science and Technology	1	Ginger 2
USA	University of Nebraska	2	EMF
			MT
Norway	Simula Research Laboratory	4	SESE
Austria	Vienna University of Technology	1	VBER
Brazil	COPPE / UFRJ	6	eSEE
Brazil	Salvador University	1	FIRE

IV. STUDY LIMITATIONS

Limitations are most related to our search strategy. The first plan was to perform both manual and automatic searches. When we started to define our search string we realized that we are in a very large scope due to the diversity of our research. Many research works in software engineering present frameworks, tools, environments and infrastructures for other different contexts than empirical software engineering. In addition, there are also many research works that describe experimental studies and controlled experiments. Because of those reasons, it was extremely difficult to perform an automated search without resulting in a large number (thousands) of studies not related to the purpose of our systematic review. As a result we decided to execute only manual searches. We agree that there is a significant effort to examine many irrelevant studies when submitting general automatic searches, but on the other hand we can ensure the gathering of relevant studies when choosing specific conferences and journals. Similar strategies have been adopted by other existing systematic reviews [23]

V. DISCUSSION

After performing the systematic review we have identified some weaknesses and opportunities for future improvements. In this section, we present these new perspectives based on the results of the systematic mapping.

A. Environment customizations based on the experiment design

The proposal of the investigated tools is to facilitate planning and conducting an experiment, minimizing threats to validity, and reducing the time spent in preparation, execution, and analysis of a controlled experiment. However, they do not mention how to set up the experimental design or how to organize the execution according to a statistic experimental design, even the most known, such as completely randomized design (CRD), randomized complete block design (RCBD), or Latin square (LS). We believe that providing assistance in how controlled experiments will actually be arranged according to statistical design can not only reduce the effort of skilled researchers on experimental software engineering, but also encouraging researchers that are no experts to perform such kinds of experiments.

B. Improved analysis capabilities

Although the analysis phase has been supported by two of the found approaches, all the environments exhibit weaknesses that should be addressed, such as: (i) help to set up the experimental design; (ii) automatic workflow generation of the execution procedure for each experiment participant in order to facilitate the automatic collection of their specific data; and (iii) finally, analysis capabilities that facilitates the production of graphics and data that help the analysis of the study according to the chosen experiment design.

C. Guidance and automatic data collection

The effort to run and manage the great volume of information collected in the experiment is substantial. In this context, actions are necessary to minimize the manual data collection effort, and the time consumed to run the experiment, in other words, such environments should enable subjects to

keep updated about their current activities through guidance and automatic data collection. It can simplify the process, since the participant would not have to collect data such as time for each activity performed, and to follow their activities through of a systematic and customized workflow of your duties. We believe these represent existing deficiencies of existing environment.

D. Future improvements for experimental software engineering environments

We have observed in our systematic review that current some environments for empirical software engineering are adaptable and extensible for specific needs of certain experiments. We identified that those environments should address the following requirements: (i) flexibility for integration with external tools, as the execution of controlled experiment involves a wide range of external tools that have to be integrated and monitored to support the complete experiment, such as process management tools, integrated development environments (IDE), testing tools, and statistic tools; and (ii) flexibility to extend the environment – to address the variety of requirements of experiments from different domains, it is also fundamental to promote the extensibility of the SE experimental environment to support new experimental study design, collected metrics, strategies to collect information from subject, and so on.

VI. Conclusions and Future work

This paper reported a systematic review study of automated support to conduct experiments in software engineering. The results indicate a restricted number of existing environments, infrastructures, tools or frameworks (total of seven). Moreover, there are few empirical studies reporting the usage of these tools. Potential future improvements for the development of experimental software engineering environments are the support to their customization to address specific needs of experiments to give more flexibility to extend their basic functionality, and to allow the integration with external tools. Inspired on the results and illustrated challenges of this systematic review, we are developing a customizable model-driven environment for supporting and conducting controlled experiments in software engineering.

Acknowledgment

This work was partially supported by the National Institute of Science and Technology for Software Engineering (INES, www.ines.org.br), funded by CNPq under grants 573964/2008-4, 560256/2010-8, and 552645/2011-7, and by FAPERN.

References

[1] D.I.K. Sjoeberg et al., "A survey of controlled experiments in software engineering," *IEEE Transactions on Software Engineering*, vol. 31 , Issue: 9, pp. 733 - 753, Sept. 2005.

[2] Claes Wohlin et al., *Experimentation in Software Engineering: An Introduction*. Boston/Dordrecht/London: Kluwer Academic Publishers, 2000.

[3] Shari Lawrence Pfleeger, "Experimental design and analysis in software engineering: Part 2: how to set up and experiment," *ACM SIGSOFT Software Engineering Notes*, vol. 20, pp. 22-26, jan 1995.

[4] Arilo C. Dias et al., "Infrastructure for SE Experiments Definition and Planning," *1st Experimental Software Engineering Latin American Workshop*, 2004.

[5] V.R. Basili, F. Shull, and F. Lanubile, "Building Knowledge through Families of Experiments," *IEEE Trans. Software Engineering*, vol. 25 (4), pp. 456-473, July/August 1999.

[6] Guilherme H. Travassos, Paulo Sérgio Medeiros Santos, Paula Gomes Mian, Arilo Cláudio Dias Neto, and Jorge Biolchini, "An environment to support large scale experimentation in software engineering," *13th IEEE International Conference on Engineering of Complex Computer Systems*, pp. 193-202, 2008.

[7] B. A. Kitchenham, T. Dyba, and M. Jorgensen, "Evidence-based software engineering, In ICSE '04," no. ICSE, pp. 273–281, 2004.

[8] B. Kitchenham and S. Charters, "Guidelines for performing Systematic Literature Reviews in Software Engineering," Keele University and Durham University, Tech. Rep, Tech. Rep EBSE 2007-001, 2007.

[9] E. Arisholm, D. I. K. Sjøberg, G. J. Carelius, and Y. Lindsjørn, "A Web-based Support Environment for Software Engineering Experiments," *Nordic Journal of Computing*, no. 9(4), pp. 231-247, 2002.

[10] Erik Arisholm, Dag I.K. Sjøberg, Gunnar J. Carelius, and Yngve Lindsjørn, "SESE – an Experiment Support Environment for Evaluating Software Engineering Technologies," *Nordic Workshop on Programming and Software Development Tools and Techniques*, pp. 81-98, 2002.

[11] AMELA KARAHASANOVIC et al., "Collecting Feedback During Software Engineering Experiments," *Empirical Software Engineering*, no. Springer Science + Business Media, pp. 113–147, 2005.

[12] Dag I.K. Sjøberg et al., "Conducting Realistic Experiments in Software Engineering," *International Symposium on Empirical Software Engineering* , 2002.

[13] Arilo Cláudio Dias Neto, Rafael Ferreira Barcelos, Paulo Sérgio Medeiros Santos, Sômulo Nogueira Mafra, and Guilherme H. Travassos, "Infrastructure for SE Experiments Definition and Planning," *ESELAW'04, Brasília, Brazil*, 2004.

[14] P. G MIAN, G. H. TRAVASSOS, A. R. C. ROCHA, and A. C. C. NATALI, "Towards a Computerized Infrastructure for Managing," *Jornadas Iberoamericanas em Ingeniería del Software e Ingeniería del Conocimiento*, 2004.

[15] P. Mian, G. Travassos, and A.R.C Rocha, "eSEE: a Computerized Infrastructure for Experimental Software Engineering," *Experimental Software Engineering Latin American Workshop (ESELAW'04)*, 2004.

[16] Paula G. Mian, Guilherme H. Travassos, and Ana Regina C. Rocha, "A computerized infrastructure for supporting experimentation in software engineering," *Experimental Software Engineering Latin American Workshop*, 2005.

[17] W. A. Chapetta, P.S.M. Santos, and G. H. Travassos, "Supporting Meta-Description Activities in Experimental Software Engineering Environments," *ESELAW'05, Brazil*, 2005.

[18] Stefan Biffl and Dietmar Winkler, "Value-Based Empirical Research Plan Evaluation," *International Symposium onEmpirical Software Engineering and Measurement*, September 2007.

[19] Kathryn T. Stolee and Sebastian Elbaum, "Exploring the use of crowdsourcing to support empirical studies in software engineering," *2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, 2010.

[20] Koji Torii, Kumiyo Nakakoji, Yoshihiro Takada, Shingo Takada, and Kazuyuki Shima, "Ginger2: An Environment for Computer-Aided Empirical Software Engineering," *IEEE Transactions on Software Engineering*, vol. (25)4, pp. 474–492., July 1999.

[21] Manoel G. Mendonça et al., "A Framework for Software Engineering Experimental Replications," *IEEE International Conference on Engineering of Complex Computer Systems*, 2008.

[22] Lorin Hochstein et al., "An Environment for Conducting Families of Software Engineering Experiments," *Advances in Computers*, vol. 74, pp. 175–200, 2008.

[23] V. Alves, N. Niu, C. Alves, and G. Valença, "Requirements engineering for software product lines: A systematic literature review," *Information & Software Technology*, vol. 52, no. 8, pp. 806-820, 2010.

SIGAA Mobile – A sucessful experience of constructing a mobile application from a existing web system

Gibeon Soares de Aquino Júnior
Department of Informatics and Applied Mathematics
Federal University of Rio Grande do Norte
Natal, Brazil
gibeon@dimap.ufrn.br

Itamir de Morais Barroca Filho
Informatics Management Office (SINFO)
Federal University of Rio Grande do Norte
Natal, Brazil
itamir@info.ufrn.br

***Abstract*—With the advent of mobile devices a new usage scenario of computing is arising, changing old habits and creating new ways of our society access information and interact with computer systems. This scenario requires new kinds of system and applications with specific requirements, capabilities and constraints. For this reason the development of systems in this context requires reviewing how the current knowledge we possess about software development fit in it. In this paper are presented some strategies successful used to implement a mobile application version from an existing web-based system.**

Keywords-component; mobile computing; system evolution; android

I. INTRODUCTION

Computers are becoming ever more present in people's lives. Nowadays, computing is so much more intense and accelerated with the rise of mobile technologies in the world, such as smartphones and tablets, connected to mobile networks increasingly available everywhere. The ITU[1] estimates that there are more than 6 billion mobile subscribers worldwide [4]. According to Gartner, 1.75 billion people have mobile phones with advanced capabilities [1] and points to further growth of such technology adoption in the coming years. There is a global trend towards increasing the number of users connected to the network via mobile devices, which in turn will produce an increasing demand for information systems, applications and content for such equipment.

We are facing a new usage scenario of computing that is changing old habits and creating new ways for our society to access information and interact with computer systems. In this new scenario arises the needs for new kind of applications and systems with heterogeneous characteristics that are composed of many computational elements, which simultaneously possess characteristics of web systems and embedded systems in addition to containing mobile devices connected to the network, which interact with the latter consuming and producing information.

The development of these new types of systems involves several activities, such as: the construction of applications that run on mobile devices; integration with exclusive services on these devices, such as GPS, SMS, NFC and other telephony services; development or evolution of existing web and embedded systems; and integration between the mobile and latter. Finally, these types of system have new requirements, capabilities and constraints. For this reason, the development of systems in this context requires reviewing how the current knowledge we possess about software development - particularly on the methods, techniques, patterns, best practices and architectures for building systems - fit into this new technological and behavioral scenario.

In this paper, we present the experience of adopting mobile technologies in the context of an existing information system at UFRN[2]. It intents to provide new ways how the users interact with the SIGAA[3], particularly the interaction from modern mobile devices. Currently, this purpose is being conducted by two actions:

- Adaptation of existing features, created specifically for web systems, to be also accessible from mobile devices. Such adaptation involves the development of applications in native platforms, like Android and iOS, the development of Restful Web Services [10] and the integration with existing enterprise software components which implement the business rules.
- Development of new specific features for mobile devices - These devices offer new possibilities and new technologies embedded in them, therefore an important and promising initiative in this context would be the development of features not existent in the scope of old systems, which is only possible with mobile devices. These features gain relevance with the use of these possibilities and technologies.

[1] International Telecommunication Union

[2] http://www.ufrn.br

[3] Integrated System for Management of Academic Activities (http://www.sigaa.ufrn.br)

II. SIGAA: A MANAGEMENT ACADEMIC SYSTEM

SIGAA is a corporative web system developed by UFRN that computerizes the business process of the academic area through the modules that composes it: graduation; masters degree (strictu and latu sensu); technical study; elementary and high school; research projects submission and control; research grants; academic extension actions control; teaching projects submission and control (monitoring and innovation); register and reports of professors' academic production; and a virtual teaching place called "virtual class". These modules are presented in Figure 1 and are enabled according to user profiles. Nowadays, SIGAA has 41,397 users, divided into the following profiles: students, professors and technicians. This web system was developed using open technologies such as Java, Hibernate, JavaServer Faces, Richfaces, Struts, EJB and Spring. It uses PostgreSQL as DBMS and is deployed into JBoss Application Server.

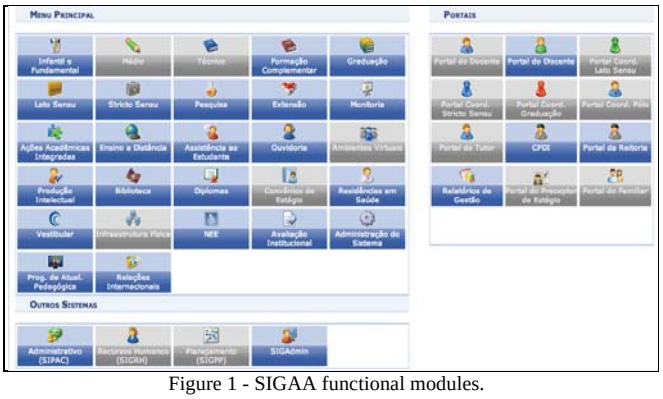


Figure 1 - SIGAA functional modules.

In terms of physical metrics, the SIGAA has 646,382 lines of code, 4,750 classes and 1,135 tables divided into 40 schemes. Already in terms of functional metrics it contains 1,858 functionalities, accounting a total of 22,369 function points[4].

It depends on a software architecture also developed by UFRN, serving as an infrastructure to this corporative system. This infrastructure is composed by four components presented in Figure 2:

- Architecture composed by abstract classes, helpers and framework services.
- LIBS composed by libraries such as JSF, Struts, Hibernate, Spring and Jasper Reports.
- Common Entities composed by common domain classes such as User, Person and Permission.
- Integrated Services composed by data transfer objects and interfaces for features available by web services.

Using this software infrastructure, the SIGAA is organized into three layers:

- Presentation: responsible for controlling interaction between users and system, containing JSF and Struts.
- Business: responsible for centralizing the domain business system logic, composed by EJB Commands and Spring.
- Data access: responsible for persistence and data queries into database, containing Hibernate and DAOs.

With the SIGAA's success in managing UFRN's academic activities, this system has been released for others Brazilian federal universities after 2009. Nowadays, about nineteen Brazilian universities are using SIGAA.

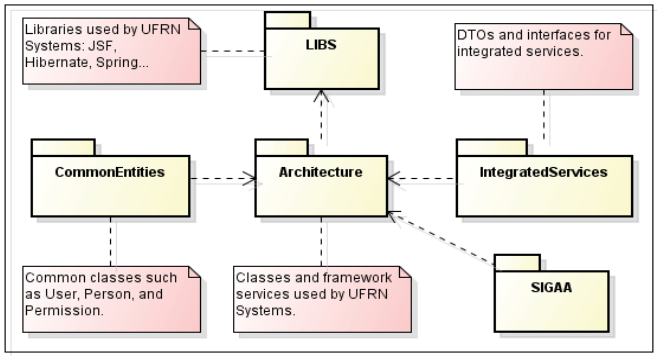


Figure 2 - UFRN Software Architecture and SIGAA.

III. STRATEGIES FOR THE DEVELOPMENT OF MOBILE APPLICATION FROM AN EXISTING WEB-BASED SYSTEM

As stated in the introduction section, mobile application development involves a specific context and therefore requires the use of new approaches related to different facets of software development. Here, we categorize these approaches in three groups: (a) Business, involves strategies related to the scope, stakeholders involvement and publicity strategies; (b) Technical, which involves aspects related to the product's source code, such as architectural approaches [7], technologies utilization, frameworks, software patterns and best practices; (c) User Interface, that involves approaches related to the visual appearance and the way the user interacts with it.

A. Business-related Approaches

At the beginning of the project, it is very important to plan which functionalities are relevant in the context of mobile environments. The process of creating a mobile application from existing web enterprise system is not a direct mapping of functionality-to-functionality. This kind of simplification is a common mistake and must be treated carefully. Mobile devices have some intrinsic restrictions such as screen size, difficulties to type long texts and no guarantee of network access availability. Moreover, it has a different mode of user interaction with touch support, gesture events and rapid actions. For this reason, it is important to follow some specific strategies on projects involving mobile application development.

One of these strategies is related to scope or more specifically to the choice of functionalities and the way they are adapted for the mobile context. Moreover, this strategy may be

[4] Based on the NESMA account method.

subdivided in four practices: (i) Choose popular functionalities; (ii) Avoid long-steps functionalities or long-fill forms; (iii) Adapt existing functionalities; (iv) Create specific functionalities for mobile application. As stated in Section II, the SIGAA is a big system and it makes no sense to try to implement each of these functionalities in the mobile application. For this reason, we started with a minimized scope, based on functional requirements of existing enterprise system, but with only the most popular and convenient functionalities (practices i and ii). Moreover, some of the chosen functionalities had to be reviewed and adapted considering the new context, as suggested by practice iii. The first version of SIGAA Mobile Application, published in April 2012, was released with only thirteen functionalities and supported the two main roles: professors and students. Finally, the third version, published in March 2013, released some new functionalities that are missing in the web-based system but are important for the mobile application.

Another important strategy is the early user involvement. For a successful software development initiative, it is well-known the importance of user involvement [5],[8], but to construct mobile application based on existing web system this practice gains even more value. The potential users of the mobile version already use the mature web-based system and, for this reason, expect a system as good as the one that already exists. Moreover, they expect that a native mobile application will make the work of using the system easier when they are on a mobile device. Based on this strategy, we tried to involve several professors and students in the project before starting development, in order to receive their expectations and feedbacks about our plans.

Finally, one of the most important strategies is the publicity around the mobile application. The potential user must know this new style of enterprise system exists and they may be motivated to try this new way of accessing the system. Only publishing the application on a platform store, e.g. Android Play, Apple App Store, is not enough to make it known among users. For this reason, its existence should be well communicated for the target audience. In our case, the first version of SIGAA Mobile had no action to communicate it was available for use. However, at the second version release, the AGECOM (Communication Agency of UFRN) did an intensive publicity about it and the numbers of installations grew significantly as can be viewed in Figure 5.

B. Technical-related Approches

The first challenge that needs to be addressed is how to integrate with the existing system. Moreover, we need to be concerned about how to reuse its already implemented business components. Nowadays, the use of the layer pattern [3] is common in particular for the web-based systems. For this reason, a generic approach that can be used to integrate with the existing web system is the definition of a new separated layer providing a set of services that must be used by the mobile application. This new layer integrates with the existing business rules layer using the already implemented and stable code, as showed by Figure 3. In the SIGAA Mobile project, this layer is composed by several restful web services that uses the business components (session beans, persistence entities and utilities classes) provided by the existing business layer.

Although it seems so simple the integration with the business layer, it may require some refactoring in existing code. It is caused by a habitual phenomenon called “software architecture erosion”, where the violation of architecture principles during the system maintenance occurs without a malicious intent. In our case, the current business layer was prepared to provide services independently of its client technology, but as it was used and matured integrated only with a unique type of client, the web view layer, some of these violations were found. A typical example found in SIGAA was the movement of specific objects from web frameworks beyond business layer boundary. For this reason, an important practice is to plan some rework in order to reform the business layer and prepare it to the expected integration.

Finally, one of the most important technical regard is that you are developing a solution in a new context with dissimilar non-functional requirements and, for this reason, others concerns must be taken. A typical example is related to the network connectivity reliability. Mobile devices are connected with a network often using wireless connections and the availability of latter may be low. Moreover, the connectivity may be unstable while the user is interacting with the system. For this reason, one of the most interesting functions implemented in SIGAA Mobile was the offline work capability. Even when there is no network connectivity, the user can enter locally and do all the operations he can do when he is online. It is possible because the application maintains a local data cache filled in background while the user interacts with the application when it is online. We also implemented a mechanism that detects the unavailability of network connection anytime while the user is interacting with the application and asks the user if he wants to continue his work in offline mode, storing the user input in a local database and synchronizing later when the connectivity is resumed. With this approach a new problem arises: the data synchronization and its consistency evaluation. Our solution was based on the same mechanism which detects the unavailability and availability of connectivity. When the application changes from offline to online state, a background routine is executed in order to perform the data synchronization. The data changed in offline state were specially tagged and this routine search for them, trying to post each modification on the server. Actually, we are performing the simplest approach, avoiding to override the data on server when it has conflicts. For the current application version, this approach is suitable because only two entities may be changed, deleted or added by the mobile version and these operations are rarely performed by different user at the same piece of data. However, we agree that in more complex environment other more refined techniques must be used.

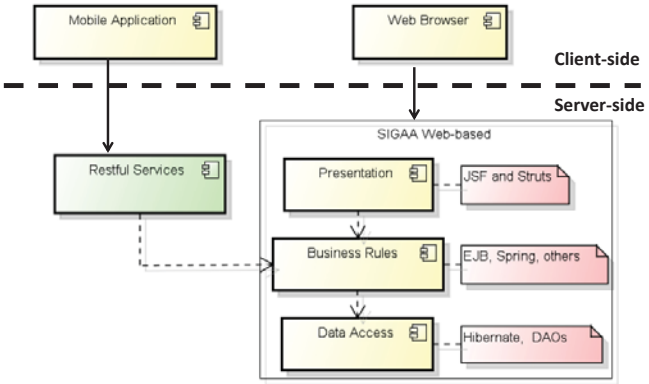


Figure 3 - Solution to integrate existing web-based SIGAA with new mobile application.

C. *UI-related Approaches*

Mobile applications tend to provide relevant advantages to their users in terms of design and usability [9]. For this reason, it requires new interaction approaches and own visual identity with the aim of differing the mobile version from the web-based system. A dangerous approach is to try to imitate the existing system. Famous applications, e.g. Facebook, Twitter and Instagram, have own appearances and usability on their mobile version. Therefore, an important strategy is to think of it as a mobile application, even if it was born from a web system, and to use the typical usability and design approaches performed by well-known applications.

Based on this understanding, we used some UI-related practices in SIGAA mobile projects. First, we define a logo to be used as icon to the application. Second, each functionality was reviewed in terms of steps to perform, quantity and arrangement of fields. Finally, the most important action we have performed was the usability evaluation. This task was done by a group of specialist following well-known techniques [2]. A particular, work done by this group was the evaluation of three different usability solutions for an important functionality. They used the methodology of Participatory Design and Prototyping Techniques, where three different prototypes have been developed and tested with the aim of investigating and evaluating the consequences of each one approaches.

After this evaluation, we found several important improvements to do. The main finds were solved and a new version (the second) was published with these improvements.

IV. RESULTS OF NEW DEVELOPED PRODUCT

The project has two task forces related to the development of mobile applications for the two main existing platforms: Android and iOS. In this paper we relate only the Android version experience, because the iOS version was not published yet.

The Figure 4 presents the architecture of the SIGAA Android application. The View layer is where all activities, i.e. components which interacts directly with the user presenting the graphical interface and manipulating the user events. It uses a layer of abstraction for the business rules that are hosted in the server called Business Delegate. This layer uses a communication channel which implements the REST pattern to communicate with the server-side of the system. Moreover, it manipulates the Cache Data Access Layer, storing and recovering data, depending of the connectivity status, to allow the offline use capability already explained early.

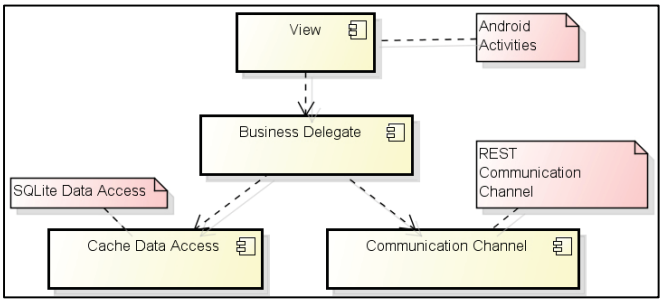


Figure 4 - Software Architecture of Android SIGAA Application.

Here we will present some statistics about the SIGAA Android uses. With Android Developer Console[5], information about download and quantification was extracted considering the following informations: active installations, device model, country, Android version, and application version. Nowadays, SIGAA Android is installed into about 2,400 devices (active installations), whose growing is presented on Figure 5. It was uploaded to Google Play at 04/25/2012 and the download number is increasing ever since.

Figure 5 - SIGAA Android active installations.

Considering device model installations, it is noted that Samsung Galaxy devices dominates with 1,069 devices, as presented on Figure 5. Observing country installations statics, it is perceived that Brazil dominates the number of downloads and installations, but analyzing Figure 6 we realize that there are other countries on the statistics, like United States of America, China, Spain, France and Portugal. This fact happens because undergraduate and postgraduate UFRN's students are participating on international programs related to the academic mobility and inter-institutional relations. Categorizing active installations by Android versions, it is discovered that Gingerbread (version 2.3) dominates the devices version with 1,310 devices, followed by Ice Cream Sandwich (version 4.0) with 477 devices as presented in Figure 8.

[5] Tool provided by Google to publishing and administrating the Android applications.

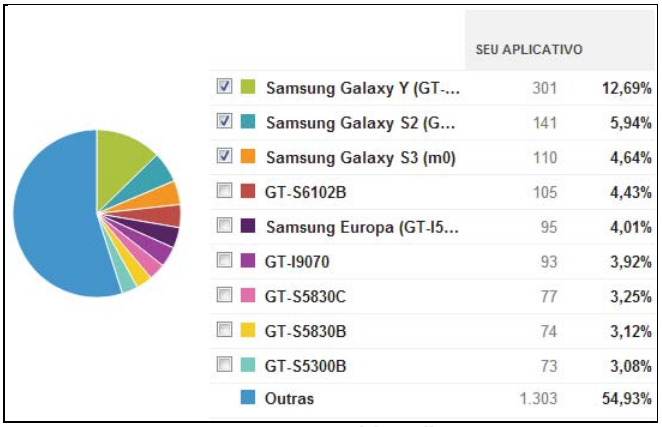


Figure 6 - Devices model installations.

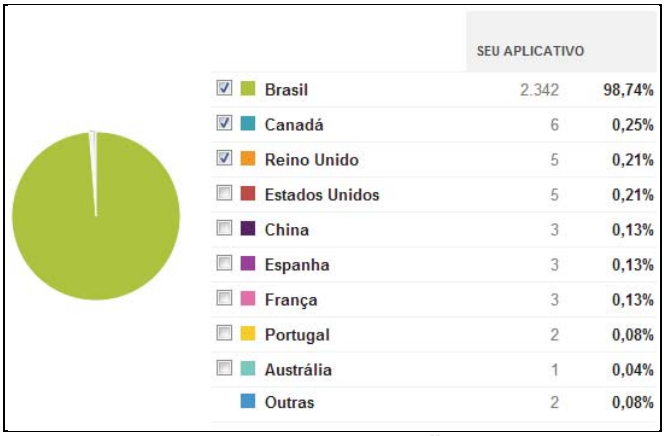


Figure 7 - Countries active installations.

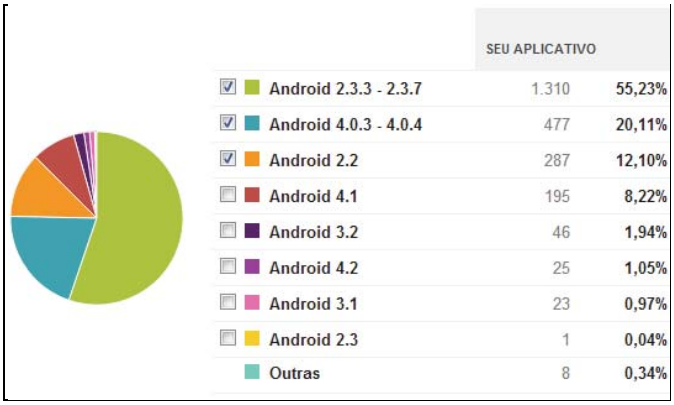


Figure 8 - Android versions active installations.

Continuing on Android Developers Console analysis, the third version of the application contains the higher active installation number, with 2,299 installations. SIGAA Android has 3 versions (1, 2 and 3) where actually the version 1 has 15 installations, version 2 has 58 installations. All these statistics data about downloads and active installations can be obtained using Android Developer Console, which is the tool for distributing Android applications.

After the active installations analysis with Developer Console, we used SIGAA log infrastructure to discover the average daily access and most used functionalities. SIGAA has a data base log controlled by an asynchronous process of persisting information about every user operations. Analyzing daily access logins into SIGAA Android, we discovered that since its release, this application has 124,643 logins and an average about 410 daily logins. We noted the increasing in login numbers at the beginning and the end of the academic activities period, as presented on the graph at Figure 8. For example, the month of vacation the number of access decreases, but immediately before and immediately after the number of access increases.

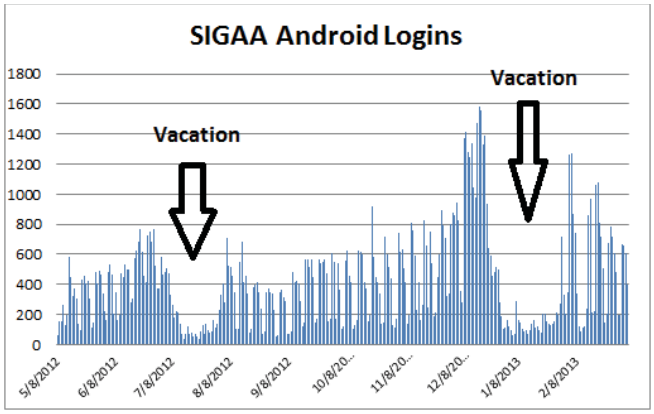


Figure 9 - SIGAA Android logins.

V. Concluding Remarks and Future Works

Although this paper report the specific experience of SIGAA Mobile project, we believe all presented strategies could be generalized and applied on other similar projects, particularly in projects with the aim of evolving an existing web-based system with the construction of its mobile version. However to be more confident of their effectiveness, a more in-depth study of the strategies and their applicability should be made. For this reason, we are evaluating the current reported strategies in other similar projects in order to collect more evidences of about the theme. Moreover, the next direction of this research would be the elaboration of a systematic process to be used in similar contexts.

Based on this experience we intend to produce a set of development recommendations, which involves best practices for projects involving the construction of mobile-based systems from existing web-based. We also could give more information, as code examples, about how a particular functionality may be adapted to the mobile application, including details on how the input was adapted, the possible differences in the workflow of both versions, the architecture components affected and other details related to this process.

Finally, we believe that a valuable result produced could be architectural solutions for this context, in special design and architectural patterns, frameworks and reusable components.

REFERENCES

[1] "Gartner Says Worldwide Mobile Phone Sales Declined 1.7 Percent in 2012" (Press release). Gartner. February 13, 2011. Retrieved March 09, 2013.

[2] Anamaria de Moraes and José Guilherme Santa Rosa. 2008. Avaliação e Projeto no Design de Interfaces (1st Edition). 2AB, Teresópolis, RJ.

[3] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. 1996. Pattern-Oriented Software Architecture: A System of Patterns. John Wiley & Sons, Inc., New York, NY, USA.

[4] ICT Data and Statistics Division. The World in 2011 – ICT Facts and Figures. Technical report, International Telecommunication Union (2011).

[5] Kent Beck and Cynthia Andres. 2004. Extreme Programming Explained: Embrace Change (2nd Edition). Addison-Wesley Professional.

[6] Lakshitha de Silva, Dharini Balasubramaniam, Controlling software architecture erosion: A survey, Journal of Systems and Software, Volume 85, Issue 1, January 2012, Pages 132-151, ISSN 0164-1212, 10.1016/j.jss.2011.07.036.

[7] Len Bass, Paul Clements, and Rick Kazman. 2003. Software Architecture in Practice (2 ed.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[8] Majid, R.A.; Noor, N.L.M.; Adnan, W.A.W.; Mansor, S., "A survey on user involvement in software Development Life Cycle from practitioner's perspectives," Computer Sciences and Convergence Information Technology (ICCIT), 2010 5th International Conference on, vol., no., pp.240,243, Nov. 30 2010-Dec. 2 2010.

[9] Nayebi, F.; Desharnais, J.-M.; Abran, A., "The state of the art of mobile application usability evaluation," Electrical & Computer Engineering (CCECE), 2012 25th IEEE Canadian Conference on , vol., no., pp.1,4, April 29 2012-May 2 2012.

[10] Roy Thomas Fielding. 2000. Architectural Styles and the Design of Network-Based Software Architectures. Ph.D. Dissertation. University of California, Irvine. AAI9980887.

ANDRIU. A Technique for Migrating Graphical User Interfaces to Android

Ricardo Pérez-Castillo, Ignacio García-Rodríguez de Guzmán, Rafael Gómez-Cornejo, Maria Fernandez-Ropero and Mario Piattini

Instituto de Tecnologías y Sistemas de Información (ITSI) at University of Castilla-La Mancha
Paseo de la Universidad 4 13071, Ciudad Real, Spain
{ricardo.pdelcastillo, ignacio.grodriguez, rafael.gomezcornejo, marias.fernandez, mario.piattini}@uclm.es

Abstract—Nowadays, pervasive environments force maintainers to provide agile solutions for migrating legacy information systems to mobile applications. While business knowledge can be easily reused in tier-based modularized systems, the migration of user interface tiers to a mobile application entails a bigger (but usually ignored) challenge. This paper presents AndrIU, a reverse engineering tool based on static analysis of source code for transforming user interface tiers from desktop applications to mobile ones. AndrIU has been specially developed for migrating traditional systems to Android applications. AndrIU is generic and extensible since it manages all the embedded knowledge in a common, standard repository according to the Knowledge Discovery Metamodel. This metamodel represents legacy knowledge in a platform-independent way. The main advantage is that AndrIU is designed to be extended for different migrations to others mobile platforms.

***Keywords*—Migration, User Interface, Android.**

I. INTRODUCTION

Pervasive environments are very common in our lives, which allow us to freely interact with a vast amount of services through a great variety of interactive devices (e.g., smartphones, laptops, pads, game consoles or digital television among other) [15]. For example, today's smartphones serve as email readers, calendars, car navigation systems or even entertainment systems, and they can provide almost constant connectivity between people via texting, voice calls, and video conferencing [3].

Pervasive environments force software producers to carry out agile developments to obtain just-in-time software applications for different devices [11]. Indeed, *App Store* and *Google Play* (two of the most important mobile applications markets) offer today more than 400,000 applications (with a growth of 20% per month) and they approximately consider 80,000 software vendors [16].

Recently, researchers and engineers have provided many methods and techniques to alleviate the challenge of agile software development for pervasive environments. Firstly, there exist multi-platform programming languages like Java. Secondly, various architectural design patterns have been proposed to facilitate the reuse of software such as three-tier architecture [4] or model-view-controller architecture [7]. The tier-based architectures encapsulate business rules in an isolated tier that minimize the coupling with other tiers in charge of user interface and data persistency.

Tier-based architectures facilitate the migration of software applications in pervasive environments since the core, business tier can be almost fully reused, while the remaining tiers has to be adapted. In fact, the sole migration of the part of an application that is interacting with the user is sometimes enough to migrate an application to different devices [2].

This paper addresses the migration of the graphical user interface (GUI) tiers of traditional applications to the user interfaces of mobile applications by integrating them with other source code tiers. This technique follows the model-driven development approach. The technique is particularly based on a static analyzer of Swing/AWT user interfaces which represent the information in a common, standard repository according to KDM (Knowledge Discovery Metamodel) [10]. KDM is the ISO/IEC 19506 standard for representing all the information retrieved by reverse engineering from every legacy software artifact (e.g., source code, databases, user interfaces, etc.). After that, the technique transforms the user interface model represented in the KDM repository to a user interface model for Android [8] applications based on a set of XML (eXtensible Markup Language) files.

The technique is supported by an Eclipse™-based tool, which was specially developed for Android applications due to their widespread use and open source nature, which facilitated the research. However, the technique is generic because it is based on the KDM standard, and therefore, additional transformations from KDM to others platforms such as iOS (based on Xcode) could easily be provided. The developed tool allows the applicability and adoption by the industry of the proposed technique for migrating user interfaces from Java-based applications to Android applications.

The remaining of the paper is organized as follows. Section II presents in detail the technique to migrate user interface tiers to Android applications. Section III introduces the supporting tool. Finally, Section 0 discusses conclusions and future work.

II. MIGRATION TECHNIQUE

AndrIU facilitates the migration of graphical user interfaces from desktop applications to mobile applications. On the one hand, the underlying process follows model-driven

development principles, i.e., (i) it treats all the involved artifacts as models in accordance with particular metamodels, and (ii) it provides automatic transformations between such models at different abstraction levels. On the other hand, AndrIU is based on KDM [10] to represents all the extracted information in a platform-independent and standardized way.

AndrIU is specially developed to the migration of AWT/SWING user interfaces of desktop applications to Android user interfaces based on XML files (see Figure 1). However, AndrIU is generic due to the use of KDM, and may easily be extended for other platforms. AndrIU considers four different artifacts and a path of three progressive transformations between them.

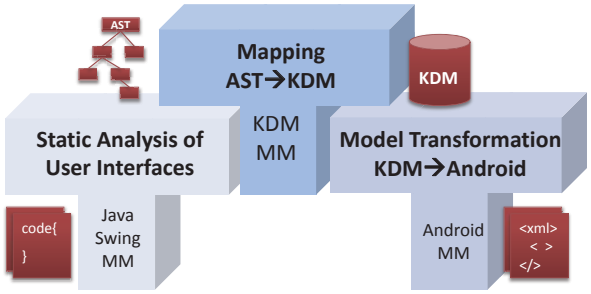


Figure 1. The user interface migration process

A. *Static Code Analysis of AWT/SWING interfaces*

In the proposed method, the first transformation is characterized by the use of static analysis as a reverse engineering technique to retrieve user interface information from the legacy source code. Static analysis consists of syntactically analyzing the source code of source files that belongs to the legacy system. Such static analysis detects code elements of the user interface tiers and ignores non-relevant pieces of source code belonging to different tiers.

This transformation is specifically tuned to analyze Java-based systems. Therefore, while the static analysis is digging up the information from a Java source file, a source code model is built on the fly according to the Java SWING metamodel. This metamodel contains some elements to represent containers such as *JFrame* or *JDialog*, and other elements to represent components within containers, e.g., *JTextField*, *JButton*, *JLabel*, etc. The parser searches for such elements in each source code file and builds the respective AST which represents a specific-platform model (PSM) (see Figure 1).

B. *Integration into the KDM repository*

After obtaining various ASTs from source code files, which contain relevant user interface elements, they are mapped to KDM elements to be integrated in the KDM repository. The mapping between Java SWING elements and KDM elements distinguishes three concerns: (i) the containers and control elements mapping; (ii) the actions mapping; and (iii) the navigability mapping.

Container and control elements mapping. The first mapping transforms container and control elements from SWING AST models to KDM UI models. Regarding containers, the mapped GUI elements are: windows (*JFrame*) and panels (*JPanel*) to *Screens*, and dialogs (*JDialog*) to *Report*. Concerning control elements, for example, buttons (*JButton*) and toggle buttons (*JRadioButton* and *JCheckBox*), which are transformed into *UIResource* elements; labels (*JLabel*) and text fields (*JTextField* and *JTextArea*) that are transformed into *UIField* elements in the KDM UI model. Since the KDM UI model has to be platform-independent, this mapping simplifies the semantics of user interface models. In fact, there are only four basic elements in the KDM UI model (*Screen*, *Report*, *UIResource* and *UIField*) to represents all the information. However, in order to avoid a semantic loss, such KDM elements incorporate an *Attribute* element that has a tagged value 'kind' which contains the classifier name of the SWING element, e.g., *JLabel*, *JButton*, etc.

Action Mapping. Since GUI elements are used to interact with other tiers of the system (e.g., the business domain tier), actions triggered under occurred events associated with GUI controls have to be collected in the KDM UI model. Such pairs of events-actions are handled in Java SWING through action listeners added in each GUI element. These elements are represented in the KDM UI model as *UIAction* elements, which have a feature *implementation* which contains a reference to a *CallableUnit* element within the KDM Code model. The implementation feature therefore represents a reference to the method that implements the triggered action. In this way, the UI and Code model are integrated within the KDM repository and feature location techniques could be used. Besides the action, *UIAction* elements contain a *kind* feature containing the type of the event (e.g., *actionPerformed*, *onMouseClick*, etc.).

Navigability mapping. Finally, the mapping between the SWING AST model and the KDM UI model is completed with the navigability mapping. This mapping uses the *UIFlow* elements, as added to the *UIAction* elements to define a navigability relationship between two windows. *UIFlow* elements contain the features 'from' and 'to' that respectively represent the references to the source and target *UIDisplay* elements (*Screen* or *Report*).

In order to detect the navigability between two different windows the parser of the previous transformation searches for calls to the method *setVisible (true)* (open a certain window) and *setVisible (false)* (close a window). These calls represent that a window can be launched from another window.

C. *Generation of Android interfaces*

Once the information retrieved from the user interface tier is integrated into the KDM repository, it can be used for migrating the KDM UI model to Android-based user interfaces.

Android applications follow a Model/View/Control (MVC) architecture [7], which consists of four different components: activities, services, intents and resources. Graphical user interfaces of Android applications therefore consist of a set of Activities classes, which are built using *View* and *ViewGroup* objects. There are many types of views and view groups, each of which is a descendant of the *View* class. On the one hand, the *View* objects are the basic units of user interface expression on the Android platform and is the base for subclasses called 'widgets', which implements user interfaces controls (e.g., text fields and buttons).

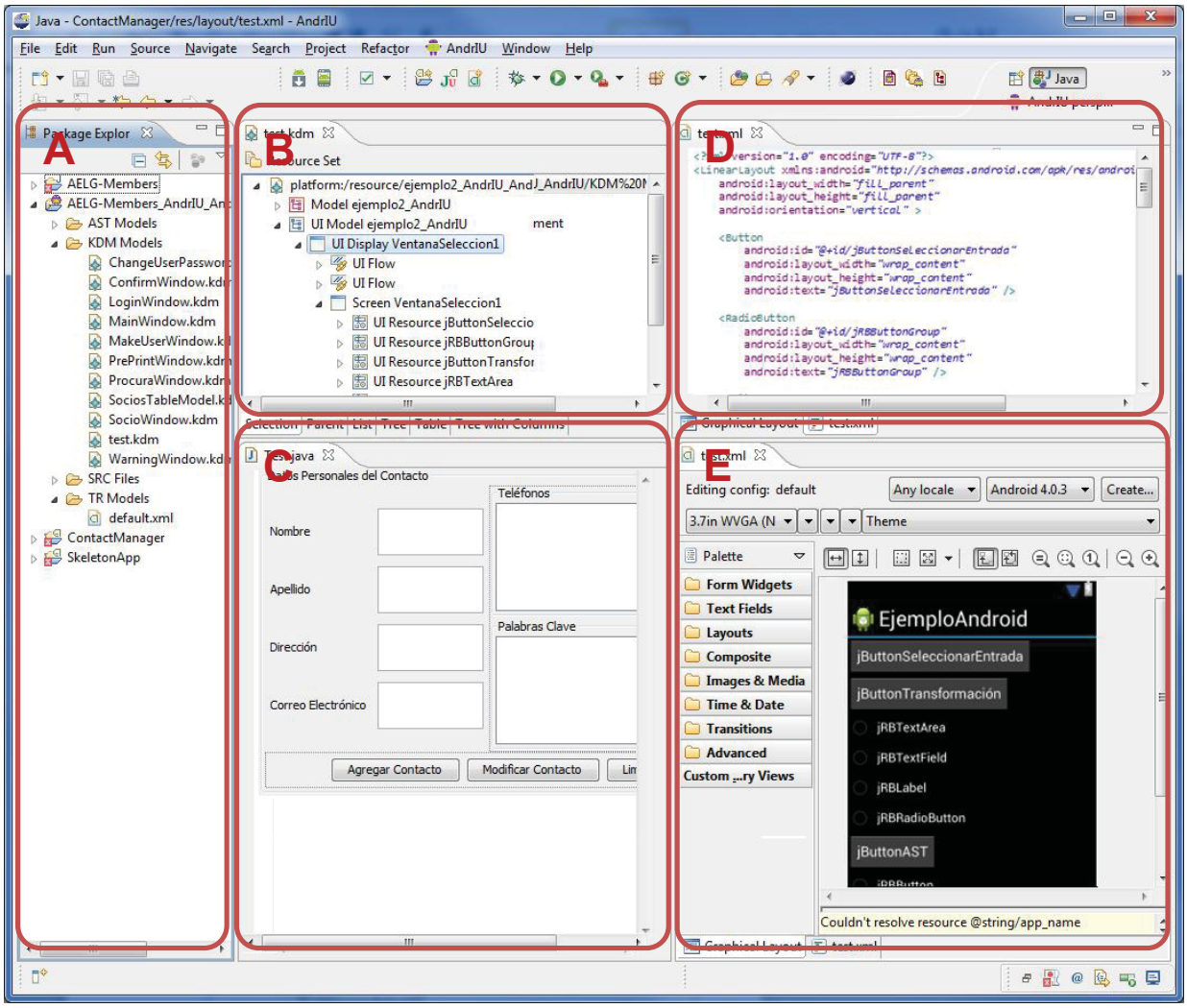


Figure 2. AndrIU modules and functionalities.

On the other hand, the *ViewGroup* class serves as the base for subclasses called 'layouts', which offer different strategies for distributing widgets such as linear, tabular, relative, and so on. Android user interfaces are depicted through XML files together a java class (*R.java*) with the definition of all the UI resources. The XML files, depicting the Android user interfaces, are built by transforming the KDM UI model almost directly. For each *UIDisplay* element this transformation creates a *Layout* element in the Android user interface. After that, it transforms all the child elements of *Screen* or *Report* elements to different Android controls (e.g., *Button*, *EditText*, *TextView*) included within the previous *Layout* element.

III. ANDRIU TOOL

AndrIU [1] is a tool based on the Eclipse platform especially developed to automate the underlying migration process (see Figure 2). AndrIU allows maintainers to complete the entire technique, since it automates the three proposed model transformations. AndrIU additionally aids a manual post-intervention by maintainers and developers through various graphical editors so that the migration can be tuned or adapted for each case.

A. Technologies Involved

This tool has been developed for Java-based legacy systems and can be used to carry out case studies involving applications with AWT/SWING user interfaces. The tool is based on four key technologies. The first technology is *JavaCC*, which is a parser and scanner generator for Java [14]. It is used to develop the static analyzer. The second technology is EMF (Eclipse Modeling Framework), which is a modeling framework and code generation facility for building tools and other applications based on structured data models [6]. This framework makes it possible to build specific metamodels according to the ECORE meta-metamodel (i.e. ECORE is the metamodel proposed by the Eclipse platform to define metamodels). Then, from these metamodels, EMF provides tools to produce a set of Java classes for the model, along with a set of adapter classes that enable viewing and command-based editing of the model as well as a basic editor. Another

Eclipse framework, such as GMF (Graphical Modeling Framework), is also used together with EMF to generate graphical editors from the ECORE metamodels. Finally, the fourth technology is XMI (XML Metadata Interchange), which is a model-driven XML integration framework for defining, manipulating and interchanging XML data and objects [12]. Every model involved in the proposed technique becomes persistent with an XMI file.

B. *Tool Modules*

AndrIU is divided into five panels supporting different functionality. Firstly, the panel A (see Figure 2) provides a project explorer to navigate through all the different artifacts involved in the three transformations (e.g., source code of the input java application, KDM models, Android files, etc.).

To support the first transformation, a tool module to carry out static analysis was developed. In this case, the module was built specifically for parsing Java source code. This tool module was developed through JavaCC from the EBNF (Extended Backus–Naur Form) grammar of Java 1.5 [13]. This module takes a Java file as input and then generates an XMI file as the output that represents the Java code model, a PSM model in L1. The second module executes a set of QVT transformations to obtain a KDM model in L2 from the Java code model obtained previously. The transformation is executed using the open source *Medini QVT* [9], a model transformation engine for QVT Relations. Panel B (see Figure 2) provides a tree view editor, which was built through EMF, to visualize and manipulate KDM models. At the same time, Panel C shows the graphical representation of the existing GUI. The third module also executes a QVT transformation to support the third transformation based on pattern matching. Panel D (see Figure 2) visualizes in an XML editor the file that represents the target user interface according to the Android platform. Finally, Panel E allows maintainers to visualize in parallel the first sketch of the target Android-based GUI so that maintainers can refine it. Panels C and E can be used by maintainers for checking manually the results between the input and output graphical user interfaces.

IV. CONCLUSIONS

This paper presents AndrIU a static analysis-based tool for migrating the GUI layer of legacy, desktop application to mobile applications. AndrIU follows the model-driven development principles and uses the KDM standard to represent the intermediate information. The usage of KDM has two important advantages. Firstly, AndrIU considers a common KDM repository in which back-end parsers for different graphical user interfaces (e.g., SWING, GTK, etc.) can store the extracted information in the common KDM repository. In turn, many front-end analyzers could be plugged in the KDM repository to migrate interfaces to different platforms. Secondly, another advantage of the KDM repository is that it facilitates the integration of additional information related to other software artifacts such us source code, databases, event model. This information may be used to exploit synergies between different artifacts during user interfaces migrations. In this sense, feature location techniques [5] may be used by mapping, for example, user interface elements with database model in order to know which data is accessed from particular user interface controls.

The work-in-progress deals with the conduction of various experiments with several industrial, Swing applications in order to demonstrate the applicability of AndrIU. Additionally, the future work will address some open issues, e.g., the adaptation of user interfaces to different devices and screens; transform layout of desktop application to mobile devices; recognize GUI elements in legacy applications with *spaghetti* code.

ACKNOWLEDGMENT

This work was supported by the FPU Spanish Program and the R&D projects PEGASO/MAGO (TIN2009-13718-C02-01) and GEODAS-BC (TIN2012-37493-C03-01).

REFERENCES

[1] Alarcos Research Group. AndrIU v1.0. Eclipse Marketplace 2012 [cited 2012 28-06-2012]; Available from: http://marketplace.eclipse.org/content/andriu.

[2] Bandelloni, R., G. Mori, F. Paternò, C. Santoro, and A. Scorcia, Web User Interface Migration through Different Modalities with Dynamic Device Discovery, in 2nd International Workshop on Adaptation and Evolution in Web Systems Engineering (AEWSE'07). 2007: Como, Italy. p. 58-72.

[3] Ebling, M.R. and M. Baker, Pervasive Tabs, Pads, and Boards: Are We There Yet?, in IEEE Pervasive Computing Magazine. 2012. p. 42-51.

[4] Eckerson, W., Three Tier Client/Server Architecture: Achieving Scalability, Performance and Efficiency in Client Server Applications. Open Information Systems, 1995. 10(1): p. 3.

[5] Eisenbarth, T., R. Koschke, and D. Simon, Aiding Program Comprehension by Static and Dynamic Feature Analysis, in Proceedings of the IEEE International Conference on Software Maintenance (ICSM'01). 2001, IEEE Computer Society. p. 602.

[6] EMF, Eclipse Modeling Framework Project. http://www.eclipse.org/modeling/emf/. 2009, The Eclipse Foundation. IBM Corporation

[7] Gamma, E., R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software. Longman Publishing Co. ed. 1995, Inc. Boston, MA, USA: Addison Wesley.

[8] Google Inc. Android (http://www.android.com/). 2012 [cited 2012 05/04/2012].

[9] ikv++, Medini QVT. http://www.ikv.de/index.php?option=com_content&task=view&id=75&Itemid=77. 2008, ikv++ technologies ag.

[10] ISO/IEC, ISO/IEC 19506. Knowledge Discovery Meta-model (KDM), v1.1 (Architecture-Driven Modernization). http://www.iso.org/iso/iso_catalogue/catalogue_ics/catalogue_detail_ics.htm?ics1=35&ics2=080&ics3=&csnumber=32625. 2012, ISO/IEC. p. 302.

[11] Martin, R.C., Agile software development: principles, patterns, and practices. 2003: Prentice Hall PTR.

[12] OMG, XML Metadata Interchange. MOF 2.0/XMI Mapping, v2.1.1. http://www.omg.org/spec/XMI/2.1.1/PDF. 2007, OMG.

[13] Open Source Initiative, Java 1.5 grammar for JavaCC. https://javacc.dev.java.net/files/documents/17/3131/Java1.5.zip. 2009.

[14] Open Source Initiative, JavaCC 4.2. A parser/scanner generator for java. https://javacc.dev.java.net/. 2009.

[15] Schmidt, A., B. Pfleging, F. Alt, A.S. Shirazi, and G. Fitzpatrick, Interacting with 21st-Century Computers, in IEEE Pervasive Computing Magazine. 2012. p. 22-31.

[16] van Agten, T. Google Android Market Tops 400,000 Applications. 2012 January 3, 2012 04/04/2012].

Using a Partially Instantiated GQM to Measure the Quality of Mobile Applications

Luis Corral, Alberto Sillitti, Giancarlo Succi
Center for Applied Software Engineering
Free University of Bozen-Bolzano
Bozen-Bolzano, Italy
Luis.Corral@stud-inf.unibz.it, {Alberto.Sillitti, Giancarlo.Succi}@unibz.it

***Abstract*— Mobile application markets manage hundreds of thousands of products involving millions of downloads. The quality of these applications is normally controlled by market policies; however, there is no link between the quality goals of the mobile software market and the practices that have to be exercised to assure a compliant application. In this paper, we propose a GQM-based strategy to supply the mechanisms to measure the quality of mobile software products. By partially instantiating a GQM structure, we can have a schema previously furnished to consider beforehand the conditions of the mobile domain that typically impact the quality of the final product. Our approach offers to apply a strong quality model in a field that requires strong quality metrics to assess products generated in short development cycles, with a potential impact of millions of users, executed in a very limited environment.**

Keywords- Assessment, Evaluation, GQM, Metrics, Mobile, Product, Quality, Standards

I. INTRODUCTION

Mobile devices have evolved from being a communication tool toward becoming the primary end-user computing equipment. The introduction of smartphones and tablets promoted a massive growth in the creation of mobile applications, thanks to their operative organization and the capability of installing and removing applications at user space, opening a development opportunity previously bounded to telephone manufacturers and carrier companies. Handheld devices are able to carry out computer operations that cover a variety of purposes, from traditional voice communication and entertainment to business applications, mobile banking, and other complex operations.

The offer and demand of mobile software applications spans in very large ranges: major mobile distribution channels manage hundreds of thousands of applications involving millions of downloads. This allows developers to distribute mobile "apps" in a wide scale, while users have access to a huge range of products. In order to maintain a standard quality level, application storefronts count on publishing guidelines that must be observed for an app to be considered for inclusion in market (mainly criteria related to adequacy of content, privacy, copyright protection, etc.). In this way, mobile apps are evaluated before being accepted for distribution and rated by other users after being downloaded. However, up to the day, mobile developers do not have a way to appraise the quality of their apps from an objective and quantitative point of view. Developers, for example, would like to know if an application meets market and user's needs, and may benefit from knowing in a detailed fashion how a successful mobile application was implemented.

In this paper, we propose a GQM-based strategy to supply the mechanisms to facilitate the definition of metrics to monitor the quality of mobile software products. By partially instantiating the definition of a GQM structure, we can have a schema that considers beforehand the conditions of the mobile domain that typically impact the quality of the final product. The rest of the paper is organized as follows: Section II reviews the related work; Section III presents our research questions; Section IV introduces our implementation strategy in the form of the Partially Instantiated GQM; Section V shows sample applications of our approach; Section VI outlines our future work and Section VII provides a summary and conclusions.

II. RELATED WORK

Under the premise of "high quality processes deliver high quality products", mobile Software Engineering has produced an extensive body of knowledge aiming to determine what are the processes and practices that facilitate the creation of high quality, successful products. As the software product specializes in one domain, these processes may be customized and adapted to suit the specific needs of the environment [1, 2]. Moreover, software metrics should be designed according to the needs of the business and the possibilities of the particular ecosystem [3, 4, 5]. When speaking about mobile software, the diverse and heterogeneous factors in which these applications rely (e.g., autonomy, connectivity, resource limitation, etc.) build up an environment that is complex and fault prone [6]. In consequence, mobile software developers should consider customized development practices that aid to ensure the quality of their applications when performing in such a limited and particular environment.

To overcome this need, there are research works that aim to furnish comprehensive frameworks to assure and evaluate the quality of mobile software product, both from process and

product standpoints. From the software process point of view, different development models have been proposed to accommodate specific needs of the mobile environment throughout activities, tasks and reviews conducted during the development process. A number of these software development methodologies concur on the fact that Agile-inspired practices are the most appropriate for conducting mobile software projects [7]. The suitability of Agile for the fulfillment of objectives mobile software development for mobile devices has been discussed in detail [8, 9], and Agile practices (Pair Programming, Refactoring, Test Driven Development) that have been analyzed have shown benefits in large industrial and experimental settings [10-15]. Some of these practices were adapted and incorporated to the mobile software domain. Building on top of this solid approach, we can point out Mobile-D [16], Hybrid Development [17], MASAM [18], Scrum [19] and Scrum/Lean Six Sigma [20].

Although the proposed methodologies show a thorough effort on process assurance, they miss to consider the target environment, disregarding environment-specific conditions that might enrich their quality and measurement activities. Focusing on product quality, research works are less extensive. Current proposals focus on methods to assess the quality of mobile applications [21]. Dantas et al. [22] presented a review of testing requirements particular to applications developed for mobile devices that can be easily adapted, incorporating detailed activities of product measurement. Spriestersbach and Springer [23] identified relevant challenges in the development of mobile web application and related such challenges into the quality characteristics described by the ISO/IEC 9126 quality standard. Mantoro [24] suggests adjustments to the same ISO quality model for assuring the quality of context-aware applications. These perspectives are highly relevant to the matter at hand, as many conditions found on the mobile web and context-aware applications also hold on general purpose mobile applications.

By analyzing the process and product methodologies already present in literature, we note that while we have a variety of works concerning mobile software quality, we still miss the link between the quality goals of the mobile software and the processes that have to be exercised to develop applications that fulfill such expectations. In addition, there is a lack of works regarding how to measure the level of accomplishment of a mobile application by using quality attributes and domain-specific software metrics.

III. RESEARCH PROBLEM

To be able to evaluate or predict the quality of a mobile software product, it is necessary to count on reference points that permit to compare the product against a series of or expected characteristics present on successful applications, in such way that a given product can be evaluated objectively and quantitatively.

Regardless of the platform or operating system used (e.g., Android, iOS, etc.), mobile devices have to cope with several physical and infrastructure limitations (memory, CPU, battery, networks, etc). Furthermore, a mobile application typically encompasses heterogeneous technologies that have to interact appropriately to provide a satisfactory experience. As a consequence, traditional Software Engineering practices should bear additional considerations. Understanding the context of the mobile domain and its limitations is of vital importance to guarantee the success of a mobile software project; a solid knowledge on these aspects enables developers to deliver better applications, and allows users to select and purchase them in a more conscientious way.

Several mature and recognized quality standards have been made available: ISO/IEC 9126, ISO/IEC 25010, McCall, and others; however, their effectiveness and applicability on the mobile software domain can be challenged as they do not take into account some of the conditions of mobile software that are not present in traditional desktop software products. Thus, it is necessary to consider the customization of these mature frameworks to fit the particular needs of the domain. For instance, there are development and appraisal models for Open Source Software (OSS), (e.g., QualiPSo OMM [25], MOST [26] or QSOS [27]). These models pick up from other well-grounded software quality models like CMMi, and introduce customizations to provide the necessary procedures to evaluate Open Source Software. The strategy followed by these models is establishing the boundaries of the OSS scope, pointing out and emphasizing domain-specific needs, and tailoring the quality model to enhance it in terms of applicability and effectiveness. Since we do not count on a quality model that defines goals, processes, policies and metrics that specifically fit the conditions existing in the mobile execution ecosystem, we can follow the example set by OSS to build a mobile-specific quality framework on top of mature software quality models, bearing in mind the scope, environment-specific concerns and, and platform-specific quality needs.

Given the high impact of the mobile software applications, it is of paramount importance to be able to analyze them to provide a sustained judgment, useful for the development, acquisition, recommendation, or deployment of mobile software products. To achieve this, we need to learn from traditional and innovative quality frameworks that have proven their value on other environments, (e.g., desktop, business, embedded) and modify them incorporating domain-specific requirements (e.g., usability matters, market awareness, etc.), and target-specific needs (e.g., physical and infrastructure limitations) To appraise the quality of the mobile software product, it is compulsory to have a clear understanding of the attributes that drive it, and to have solid foundations about the expectations of the stakeholders. With the purpose of finding these solutions, we formulated a research question:

RQ: How mobile software products can be analyzed to provide a solid and objective notion of its quality? This means that we set as the main objective of this work to achieve the capability of associating the critical requirements and success factors of a mobile product with quality characteristics that can be measured and controlled.

To provide an answer, we need first to survey what is typically expected from a mobile application from the viewpoint of the involved stakeholders. Many of the expectations of traditional software hold in the mobile scope; however, it is essential to deepen this analysis into the mobile domain. After this, it is necessary to analyze what are the

conditions present in mobile ecosystem, not present in other environments, which have a significant impact on the quality of the software product. At this point, we will be in a position to set and find the answer of an important sub-question:

How can we measure the quality of a mobile software product? The answer of this sub-question requires the definition of a collection of metrics that help to approximate quantitatively the quality of a mobile product bearing in mind attributes of the product that are highly relevant for users, developers, execution environments and application markets.

IV. Implementation Approach

Our research question requires conducting a comprehensive survey to select the most important quality requirements set upon apps. Mobile application markets already count on policies that can be considered the minimum quality expectation for a mobile app. These publication guidelines are highly influential (i.e., applications not compliant with them are not included on the market, or discontinued). We may start from those rules to draw an outline of what a mobile application should look for, what constraints should it bear, and how should it perform. Therefore we may consider enhancing them with additional requirements surveyed from other sources (e.g., mobile software development experiences, processes etc).

Then, the answer to our research sub-question requires an accurate and comprehensive definition of domain-specific metrics. To this end, we propose to use the Goal-Question-Metric (GQM) approach. The GQM approach proposes to define a goal, refine such goal into questions, and introduce metrics that collect the necessary information to answer the questions. A goal represents the conceptual level: it is defined for an object for a variety of reasons, with respect to various models of quality, from various points of view and relative to a particular environment. Questions represent the operational level: A set of questions try to characterize the object of measurement with respect to a selected quality issue, and to determine its quality from a selected viewpoint. Finally, metrics represent the quantitative level: data is associated with every question to answer it in a quantitative way [28, 29].

To evaluate the quality of a product in the mobile domain, it is necessary to relate the mobile-specific characteristics with measurable attributes. For instance, from an operative point of view, we should be aware that mobile software runs on a slow processor, with limited input means, powered by a battery, dealing with high autonomy requirements. Among others, these constraints pose common requisites on the application's quality, and such constraints may be characterized to understand how the mobile environment, by itself, introduces new quality requirements that can be also analyzed and traced.

To maximize the profit of our approach, we propose to summarize the common requisites and prepare a GQM baseline that can be partially instantiated. This GQM should work as a template that can be detailed and customized depending on the scope of the desired quality assessment. We believe that a GQM can be preliminarily furnished to facilitate an efficient approximation of the quality from a viewpoint relevant to the mobile ecosystem, considering beforehand the conditions that typically arise on the mobile domain and that impact the quality of the final product. By applying the partially instantiated GQM, the time to produce a new metric can be significantly shortened and it will be ensured that such metrics will be beneficial to keep track of mobile-specific quality drivers.

Creating partial instantiation of a GQM allows us to establish a core, extensible model that can be tailored, depending on the focus of the analysis. The goal will always be the analysis of the quality of the product:

G.1: *"**Analyze** the mobile software product for the purpose of **evaluating** it with respect to the **quality**, from the view point of **developers and customers**, in the context **of execution environment and application markets**".*

The questions, on the other hand, can be preliminarily set to survey the characteristics of the application and to evaluate it with respect to an outstanding attribute (*X*) that relates directly to the issue that the GQM model attempts to solve:

Q.1: *"What is the current performance of the application with respect to attribute X?"*

Q.2: *"Is the current performance of attribute X satisfactory from the viewpoint of the developer and the user?"*

Q.3: *"Is the current performance of attribute X acceptable from the viewpoint of the application market?"*

Finally, it is not possible to supply beforehand a fully-equipped set of metrics, since they shall be proposed depending on each case. Then, it should be provided a data source that leads to answer the questions from an unbiased and quantitative point of view.

V. Sample Application

Let us consider two sample applications of our approach, to evidence its actual feasibility.

In our first example, we want to evaluate the quality of the Facebook Android App in terms of its suitability with the battery life. Based on our partial instantiation of the GQM, we translate it into the following interpretation:

G.1: *"Monitor the quality of the product for the purpose of measuring its energy consumption from the point of view of the user in the context of the mobile execution environment".*

Different questions need to be set, including:

Q.1: *"What is the current performance of the application with respect to energy consumption?"*

Q.2: *"Is the current performance of energy consumption satisfactory from the viewpoint of the user?"*

Q.3: *"Is the current performance of energy consumption acceptable from the viewpoint of the application market?"*

To solve these questions, a number of metrics can be calculated, for example:

M.1: *Number of milliwatts consumed by the mobile phone while the application is executed.*

M.2: *Percentage of battery drained while the application is executed.*

Such values will supply an accurate notion of the amount of energy spent by the application. To determine the answers to the questions and the accomplishment of the goal, the values obtained from the metrics should be evaluated with respect to a performance standard and the market policy concerning to energy consumption and battery drains.

In the second application, we want to evaluate the quality of the Trip Advisor Android App in terms of its observance of user's privacy while retrieving his or her physical position. From the partial instantiation of the GQM, we obtain:

G.1 *"Monitor the quality of the product for the purpose of assessing its compliance with the privacy policy from the point of view of the user in the context of the mobile execution environment"*.

Possible questions are:

Q.1: *"What is the current performance of the application with respect to privacy compliance?"*

Q.2: *"Is the current performance of the privacy compliance satisfactory from the viewpoint of the user?"*

Q.3: *"Is the current performance of the privacy compliance acceptable from the viewpoint of the application market?"*

We can define the following metrics:

M.1*: Amount of time spent on accesses to GPS data while the application is working.*

M.2: *Amount of time spent on network access while the application is working.*

In this way, one can analyze if the app is retrieving the user's position only under his or her consent. To complete the evaluation of the questions and the achievement of the goal, the values obtained from the metrics should be evaluated with respect to a performance standard, and the market policy concerning to privacy management.

Since the metrics may be hard to determine, we foresee the selection of a mature software assurance framework that can be tailored to the needs given by the mobile ecosystem and the mobile product. Our intention is to build on top of the ISO/IEC 25010:2011 quality standard [30], and analyze its quality characteristics and attributes, focusing in those that are applicable to the quality of the mobile software product context. ISO/IEC 25010:2011 defines quality characteristics grouped in two orthogonal dimensions: A "product quality model" that relates to static properties of software and dynamic properties of the computer system, and a "quality in use model" that relates to the outcome of interaction when a product is used in a particular context of use, making it very suitable to the viewpoint of our GQM.

VI. Future Work

There is still an extensive effort to carry out in order to accomplish the goals of the underlying work. With this proposal and further experimentation, we aim to contribute to state of the art on mobile Software Engineering by enabling mechanisms to assure and monitor the quality of the mobile software product, based on real-world quality expectations and real-world market awareness, delivering a methodology to identify metrics that enable substantiated quantitative analysis. We also envision to provide the means to assist in decision making to establish strategies or recommend practices to optimize the development processes in mobile software projects. This work also aims to share other relevant questions that have been marginally established at this stage, such as:

(a) What is the extent in which the execution environment affects the quality of a mobile application?

(b) Will the quality metrics of a given mobile app relate to the restrictions and constraints imposed by the application market (i.e., the most restrictive the app market is, the less defects should be found in an application downloaded from it?)

In a long-term scope, establishing a strategy for software quality assessment for mobile applications will be helpful for researchers, developers and users to:

(i) Determine the capacity of the mobile software application to meet specific requirements and demands.

(ii) Identify quality requirements addressable when designing and implementing the mobile software product, and implement metrics that allow to measure quality attributes and track quality characteristics.

To verify the accuracy and usefulness of this approach, it is necessary to conduct a piloted multi-dimensional evaluation of diverse applications using the partially instantiated GQM and its associated quality attributes. We will design and deploy a case study that will supply the empirical data required to analyze the effectiveness this approach a real industrial setting. The analysis will be executed by designing and implementing product metrics to analyze the applications developed by a specialized mobile software company. The definition of the metrics will be done after a partially instantiated GQM created in collaboration with project managers, developers, testers and pilot users. Metrics will be co-related to quality attributes present in ISO/IEC 25010:2011, to guarantee the implementation of both the product quality and quality in use models. Finally this study will be complemented with developer's data collected non-invasively, to consider the impact of different levels of experience and seniority [31-35].

As means of validation, we propose that the product quality metrics obtained after the analysis shall be compared to the number of downloads and to the user's rating obtained by the product in the application store, to identify if there is correlation between the metric-oriented product assessment and the user-provided product assessment.

VII. Conclusions

The expansion of mobile platforms as high-end, ubiquitous computing equipment requires to understand the mobile environment to create the best strategies to assure the quality of the mobile software product from a domain-specific point of view. Our approach pursues to apply a specific, partially instantiated GQM that helps to identify goals, set questions and design effective metrics considering in advance the conditions that exist on the mobile domain and that impact significantly the quality of the final product.

Mobile applications stand out from other traditional software products due to a highly competitive market and a potential impact on millions of users. The need of assuring the development of high-quality mobile products becomes an imperative, and demands extensive investigation and experimentation from the academic and practitioner point of view. The answers to our research question will enable us to relate the real-world expectations on the mobile software products with measurable characteristics, so that developers can deploy quality strategies for their mobile software projects, and final customers can conduct trustable evaluation efforts before purchasing or deploying a new mobile application.

REFERENCES

[1] Maurer, F., Succi, G., Holz, H., Kötting, B., Goldmann, S., Dellen, B.; Software process support over the Internet. In Proceedings of the 21st International Conference on Software Engineering, pp. 642-645. 1999.

[2] Benedicenti, L., Succi, G., Vernazza, T., Valerio, A.; Object oriented process modeling with fuzzy logic. In Proceedings of the 1998 ACM Symposium on Applied Computing, pp. 267-271. 1998.

[3] Pedrycz, W., Succi, G., Chun, M.G.; Association analysis of software measures. International Journal of Software Engineering and Knowledge Engineering, 12 (3) , pp. 291-316. 2002.

[4] Scotto, M., Sillitti, A., Succi, G., Vernazza, T.; A relational approach to software metrics. In Proceedings of the ACM Symposium on Applied Computing, pp. 1536-15408. 2004.

[5] Pedrycz, W., Succi, G., Musïlek, P., Bai, X.; Using self-organizing maps to analyze object-oriented software measures. Journal of Systems and Software, 59 (1) , pp. 65-82. 2001.

[6] Corral L., Georgiev A. B., Sillitti A., Succi G.; A Method for Characterizing Energy Consumption in Android Smartphones. 2nd International Workshop on Green and Sustainable Software (GREENS 2013) in connection with ICSE 2013, pp. 38-45. 2013.

[7] Corral L., Sillitti A., Succi G.; Software Development Processes for Mobile Systems: Is Agile Really Taking Over the Business?. 1st International Workshop on Mobile-Enabled Systems (MOBS 2013) in connection with ICSE 2013, pp. 19-24. IEEE. 2013.

[8] Abrahamsson, P., Warsta, J., Siponen, M., Ronkainen, J.; New directions on Agile methods: A comparative analysis. In Proceedings of the International Conference on Software Engineering. 2003.

[9] Abrahamsson, P.; Mobile software development: the business opportunity of today. In Proceedings of the International Conference on Software Development, pp. 20-23. 2005.

[10] Russo B., Scotto M., Sillitti A., Succi G.; Agile Technologies in Open Source Development. IGI Global, USA, ISBN 978-1-59904-681-5. 2009.

[11] Di Bella E., Fronza I., Phaphoom N., Sillitti A., Succi G., Vlasenko J.; Pair Programming and Software Defects – a large, industrial case study. Transactions on Software Engineering, IEEE. To appear (DOI: 10.1109/TSE.2012.68). (n.d.)

[12] Sillitti, A., Succi, G., Vlasenko, J.; Understanding the impact of pair programming on developers' attention: a case study on a large industrial experimentation. In Proceedings of the International Conference on Software Engineering. pp. 1094-1101. 2012.

[13] Moser R., Sillitti A., Abrahamsson P., Succi G.; Does refactoring improve reusability?. 9th International Conference on Software Reuse (ICSR-9). 2006.

[14] Moser R., Abrahamsson P., Pedrycz W., Sillitti A., Succi G.; A case study on the impact of refactoring on quality and productivity in an agile team. In Proceedings of 2nd IFIP CEE-SET. 2007.

[15] Janes A., Remencius T., Sillitti A., Succi G.; Managing Changes in Requirements: an Empirical Investigation. Journal of Software: Evolution and Process, Wiley, to appear. (n.d.)

[16] Abrahamsson, P., Hanhineva, A., Hulkko, H., Ihme, T., Jäälinoja, J., Korkala, M., Koskela, J., Kyllönen, P., Salo, O.; Mobile-D: An Agile approach for mobile application development. In Proceedings of OOPSLA'04. 2004.

[17] Rahimian, V. Ramsin, R.; Designing an Agile methodology for mobile software development: A Hybrid Method engineering approach. In Proceedings of the 2nd International Conference on Research Challenges in Information Science. 2008.

[18] Jeong, Y.J., Lee, J.H., Shin, G.S.; Development Process of Mobile Application Software Based on Agile Methodology. Proceedings of the 10th International Conference on Advanced Communication Technology, pp. 362-366. 2008.

[19] Scharff, C., Verma, R.; Scrum to support mobile application development projects in a just-in-time learning context. Proceedings of the 2010 Workshop on Cooperative and Human Aspects of Software Engineering (CHASE 2010) in connection with ICSE, pp. 25-31. 2010.

[20] Cunha, T.F.V., Dantas, V.L.L., Andrade, R.M.C.; SLeSS: A Scrum and Lean Six Sigma integration approach for the development of software customization for mobile phones. In Proceedings of the 25th Brazilian Symposium on Software Engineering, pp. 283-292. 2011.

[21] Ryan, C., Rossi, P.; Software, performance and resource utilisation metrics for context-aware mobile applications. In Proceedings of the 11th IEEE International Software Metrics Symposium. 2005.

[22] Dantas, V. L. L., Marinho, F. G., da Costa, A. L., and Andrade, R. M. C.; Testing requirements for mobile applications. In Proceedings of the 24th International Symposium on Comp. and Information Sciences, pp. 555-560. 2009.

[23] Spriestersbach, A. and Springer, T.; Quality attributes in mobile web application development. In Proceedings of the 5th International Conference Product Focused Software Process Improvement, LNCS vol. 3009, pp. 120-130. 2004.

[24] Mantoro, T.; Metrics Evaluation for Context-Aware Computing. In Proceedings of the 7th International Conference on Advances in Mobile Computing & Multimedia, 2009.

[25] Petrinja, E., Nambakam, R., Sillitti, A.; Introducing the Open Maturity Model. In Proceedings of the 2nd Emerging Trends in FLOSS Research and Development Workshop in connection with ICSE. 2009.

[26] Del Bianco, V., Lavazza, L., Morasca, S., Taibi, D.; Quality of Open Source Software: The QualiPSo Trustworthiness Model. In Proceedings of the 5th International Conference on Open Source Systems. 2009.

[27] Petrinja E., Sillitti A., Succi G.; Comparing OpenBRR, QSOS, and OMM Assessment Models. In Proceedings of the 6th International Conference on Open Source Systems. 2010.

[28] Basili, V, Caldiera, G., Rombach, D.; The Goal Question Metric Approach. Chapter in Encyclopedia of Software Engineering. Wiley, ISBN: 1-54004-8. 1994.

[29] Basili, V.; Applying the goal question metric paradigm in the experience factory. In Proceedings of the 10th Annual Conference of Software Metrics and Quality Assurance in Industry. 1993.

[30] International Organization for Standardization. ISO/IEC 25010:2011 Systems and software engineering. Quality Requirements and Evaluation System and software quality models, www.iso.org. 2011.

[31] Scotto, M., Sillitti, A., Succi, G., Vernazza, T.; A non-invasive approach to product metrics collection Journal of Systems Architecture, Vol. 52 (11) pp. 668-675. 2006.

[32] Corral, L., Sillitti, A., Succi, G., Strumpflohner, J., Vlasenko, J.; DroidSense: a mobile tool to analyze software development processes by measuring team proximity. Proceedings of TOOLS 2012, pp. 17-33. LCNS vol. 7304. 2012.

[33] Scotto, M., Sillitti, A., Succi, G., Vernazza, T.; A non-invasive approach to product metrics collection Journal of Systems Architecture, Vol. 52 (11) pp. 668-675. 2006.

[34] Sillitti, A., Succi, G., Vlasenko, J.; Toward a better understanding of tool usage: NIER track. In Proceedings of the International Conference on Software Engineering, pp. 832-835. 2011.

[35] Fronza, I., Sillitti, A., Succi, G., Vlasenko, J.; Understanding how novices are integrated in a team analysing their tool usage. Proceedings of the 2011 International Conference on Software and Systems Process, pp. 204-207. 2011.

Locating and Understanding Concurrency Bugs Based on Edge-labeled Communication Graphs

He Li[1,2], Mengxiang Lin[1,3], Tahir Jameel[1,2], and Zhenyuan Jiang[1,2]

[1]State Key Laboratory of Software Development Environment, Beihang University, Beijing, China
[2]School of Computer Science and Engineering, Beihang University, China
[3]School of Mechanical Engineering and Automation, Beihang University, China
{lihe, mxlin, tahir, zyjiang}@nlsde.buaa.edu.cn

Abstract

Concurrency bugs are difficult to locate and understand. This paper presents LUCON, a novel edge-labeled communication graph based technique, which locates the concurrency bugs and presents buggy access patterns and bug triggering scenarios to help programmers understand the bugs. The buggy access pattern gives the essence of the bug and the bug triggering scenario shows how the bug happens with the information of call stack and thread creation. A set of experimental studies have been conducted to evaluate the effectiveness of LUCON. The preliminary results show that LUCON can locate concurrency bugs accurately and the bug reports provided by LUCON can really help programmers understand the bugs.

1 Introduction

Locating and understanding concurrency bugs is more challenging than that in sequential programs for at least two reasons. First, concurrent programs can have potential astronomically large number of thread interleavings and only rare particular interleavings can trigger the concurrency bugs. This makes it difficult for programmers to expose and locate these bugs. Second, the manifestation of concurrency bugs usually involves complicated interactions among multiple threads, and is therefore hard to understand.

A variety of fault locating techniques for concurrent programs have been proposed [6][7][11][12][17]. Among them, we are particularly interested in Recon [11], which not only can locate concurrency bugs but also presents short execution fragments to try to help understand these bugs. However, the results provided by Recon are still not well enough for programmers to clearly understand bugs. For example, for atomicity violation, which usually involves three or more statements, Recon often gives two of them. Actually, the missing statements are also necessary to understand concurrency bugs. In addition, Recon may rank the bug low, which means programmers have to take time to check some results unrelated to the bug.

In this paper, we propose LUCON, a new communication graph based technique to help locate and understand concurrency bugs. We extend a communication graph to an edge-labeled communication graph by associating each edge with labels. Specifically, one label represents whether the edge is relevant to buggy behavior and the other provides the edge's run-time information. Based on the edge-labeled communication graph, our method finds buggy access patterns which give the essence of bugs and constructs bug triggering scenarios which show how bugs happen to help programmers understand concurrency bugs. For a concurrency bug, due to the difference of execution contexts, this bug may be represented by several different edges in different communication graphs. We call these edges as *reduplicative edges*. In addition, if one edge is irrelevant to buggy behavior, we consider it as a *redundant edge* in fault locating. By integrating reduplicative edges and reducing redundant edges, our technique locates concurrency bugs accurately.

The main contributions of this paper are as follows: (1) We extend a communication graph to an edge-labeled communication graph, which characterizes properties of communications between threads; (2) Our technique proposes the idea of buggy access pattern and bug triggering scenario to help programmers understand concurrency bugs; (3) We have implemented LUCON for C/C++ programs and evaluation results show that LUCON can rank the bugs top on our benchmark.

2 Motivation

2.1 Concurrency bugs

In this paper, we focus on atomicity violations and order violations which are pervasive non-deadlock concurrency

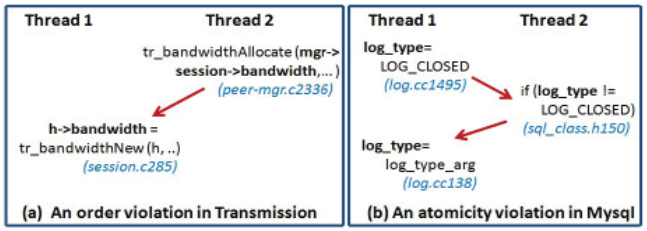


Figure 1. Bugs in Transmission and Mysql.

bugs in practice [8]. An atomicity violation happens when memory accesses supposed to be executed atomically are interrupted by the memory accesses in other threads. An order violation happens when memory accesses in different threads execute in an unexpected order.

For example, Figure 1(a) shows an order violation in `Transmission`, i.e. the read access to shared variable `h->bandwidth` in thread 2 is expected to be execute after the write access in thread 1, but actually not. Under the wrong order, the program reads an uninitialized null pointer and crashes later. Figure 1(b) shows an atomicity violation in `Mysql`, i.e. the two write accesses to the shared variable `log_type` in thread 1 are expected to executed atomically, but actually the read access in thread 2 occurs between the two write accesses. This unexpected read access causes loss of some records. The blue part represents the access's position in source file. For example, `session.c285` in Figure 1(a) means the statement is at line 285 in file `session.c`.

2.2 A motivating example

Figure 2 shows the result provided by Recon, which is called an aggregated reconstruction, for the atomicity violation in Figure 1(b). A node in the aggregated reconstruction is a memory access statement represented by its position in source file. For example, the top left node `myopen.c97` means the statement that executes the memory access is at line 97 in file `myopen.cc`. The two nodes `log.cc1495` and `sqlclass.h150`, which are connected by the black bold line, make up the buggy edge. The nodes in the red, blue and green circles are the memory accesses that happen immediately before `log.cc1495`, between `log.cc1495` and `sqlclass.h150`, and immediately after `sqlclass.h150`, which are called `prefix`, `body` and `suffix` in Recon respectively. Recon proposed to use the aggregated reconstruction to help understand bugs. Figure 2 shows that the buggy edge related to the bug is indeed included in the aggregated reconstruction. However, understanding the bug is still not easy due to the following reasons:

(1) The reconstruction only gives two memory accesses involved in the bug: `log.cc1495` and

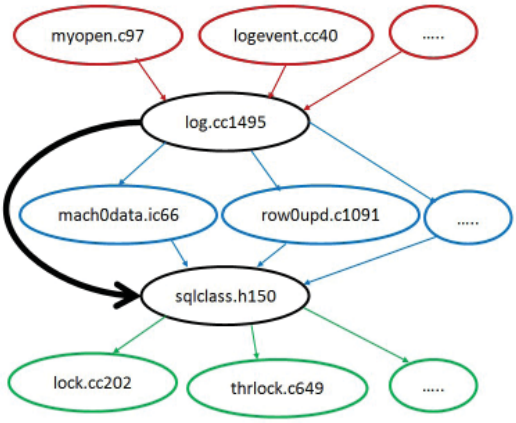


Figure 2. The result of Recon.

`sqlclass.h150`. According to these two memory accesses, it is difficult to determine whether the bug is caused by order violation where `log.cc1495` and `sqlclass.h150` are executed in an unintended order or by atomicity violation where an atomic region that contains one node of {`log.cc1495`, `sqlclass.h150`} is interrupted by other nodes. Even if programmers suppose the bug to be an atomicity violation, it is very difficult to find the remaining part. In fact, the node `log.cc138` that should be executed atomically with `log.cc1495` is far from `log.cc1495` in the source file, i.e. they are at line 138 and line 1495 in file `log.cc` respectively. Such a long distance makes it difficult for programmers to find node `log.cc138`.

(2) The aggregated reconstruction does not provide an effective execution fragment. The aggregated reconstruction mixes the information of several executions without making a distinction. Extracting one concrete execution from the aggregated reconstruction is impossible. For example, programmers cannot know whether the nodes `myopen.c97` and `logevent.cc40` in `prefix` appear in the same execution. Even if programmers use one single reconstruction, the nodes in `prefix`, `body` and `suffix` cannot show useful information about the bug. For example, the node `logevent.cc40` is an irrelevant memory access to `log.cc1495` and is less helpful.

Besides, this reconstruction is ranked 34th[1] which means programmers need to check 33 irrelevant reconstructions before it.

3 Approach

Given some buggy and nonbuggy executions, LUCON proceeds in three steps: (1) locate and rank the buggy mem-

[1]We will show this rank value in Section 4.1.

ory access pair; (2) find buggy access patterns that contain the buggy memory access pair; (3) build a bug triggering scenario based on the buggy access pattern.

3.1 Locating buggy memory access pair

Our method is based on the communication graph that represents concurrent program execution, where nodes represent the memory accesses and edges represent the inter-thread communications between nodes via shared memory. The edge in the buggy graphs that is strongly correlated with the occurrence of buggy behavior is most likely suspect. However, edges unrelated to buggy behavior influence the ranking accuracy of the real buggy edge. Moreover, the lack of run-time information about the edge does not facilitate programmers to understand the bug. To this end, we extend the definition of communication graph as follows:

Definition 1. ***(Edge-labeled communication graph)*** *An edge-labeled communication graph is a system* $G = (V, E, I, F)$*, where*
(1) V *is a finite set of nodes and each node* $v \in V$ *is a tuple* (s, ctx)*, where* s *is the statement that executes the access and* ctx *is the execution context of this memory access;*
(2) $E \subseteq V \times V$ *is a finite set of edges, and each edge* $e = (u, v) \in E$ *is a communication via a shared memory;*
(3) $F : E \rightarrow \{True, False\}$ *is a mapping that associates with each edge* e *a truth value;*
(4) $I : E \rightarrow TS \times T \times B \times TS \times T \times B \times M \times P$ *is a mapping that associates with each edge* e *a tuple , where* TS *is the set of timestamps* ts*,* T *is the set of threads* t*,* $B = \{R, W\}$ *is the set of access types* b*,* M *is the set of shared memories* m *and* P *is the set of processes* pid*.*

The edge label F is introduced to mark the edges that are not relevant to the buggy behavior. For example, edges, whose one node occurs before the time that the main thread creates the first thread, will not cause concurrency bug. Redundant edges can influence the ranking accuracy, but the nodes contained in them may be useful in understanding the bug. About label I, for an edge e, $I(e) = (ts_1, t_1, b_1, ts_2, t_2, b_2, m, pid)$ is the run-time information of e, where (ts_1, t_1, b_1) is the execution time, thread id and access type of the source node respectively, (ts_2, t_2, b_2) is the similar information of sink node, m is the shared memory that nodes in e accesses and pid is the process identifier.

For each edge e that may be relevant to the bug and appears in the buggy executions, we use the method proposed in [11] to count the edge's frequency in the buggy and nonbuggy executions, i.e. $buggyCount[e]$ and $nonbuggyCount[e]$ respectively. These edges compose a set called $edgesInBuggyEexecution$.

Besides redundant edges, reduplicative edges also influence the ranking accuracy. Reduplicative edges are different edges whose difference is due to the difference of execution contexts but they correspond to the same bug. Separately counting their frequency will lower the bug's rank. For example, one edge (s_1, ctx_1, s_2, ctx_2) appears in some buggy graphs and another edge (s_1, ctx_3, s_2, ctx_4), which has the same memory access pair (s_1, s_2) but slightly different context, appears in the other buggy graphs. They both do not appear in the nonbuggy graphs and correspond to the same bug, i.e. the buggy communication between s_1 and s_2. Separately counting the two edges' frequency will lower their ranks.

In order to reduce the impact of reduplicative edges, LUCON integrates their frequencies to the frequency of their common memory access pair (s_1, s_2), i.e. for each edge $e = (s_1, ctx_1, s_2, ctx_2) \in edgesInBuggyEexecution$, the frequency of memory access pair $mp = (s_1, s_2)$ whose initial value is 0 is calculated by:

$$freq(mp) = \begin{cases} freq(mp) + buggycount[e], & \text{if } nonbuggycount[e] = 0, \\ freq(mp), & \text{if } nonbuggycount[e] \neq 0. \end{cases}$$

During the integrating process, for each mp, when it is updated by an edge e for the first time, we associate it with $I(e)$, which presents the run-time information for the memory access pair.

At last stage, for each memory access pair, mp, LUCON computes the suspiciousness by the following equation:

$$suspiciousness(mp) = \sqrt{freq(mp)^2 + C(mp)^2} \quad (1)$$

where $freq(mp)$ is the frequency of mp and $C(mp)$ proposed in [10] is a disparity in communication behavior between buggy and nonbuggy graphs, and then LUCON ranks these memory access pairs by their suspiciousness.

3.2 Finding buggy access pattern

Many techniques [9][13][14][15][16] characterize faults by likely interleaved sequences of operations, i.e. *buggy access pattern* which is called *pattern* for simplicity in this section. Based on our insight, patterns are valuable for concurrency bug understanding because they contain all the memory accesses involved and the interleaving orders among these accesses. Based on the memory access pair, LUCON tries to find the possible patterns that contain the memory access pair.

For convenience, we denote a memory access to a shared variable by $b_{t,s}$, where b is the memory access type, i.e. either read (R) or write (W), t is the thread that executes the access, and s is the corresponding program statement. For example, R_{1,s_1} indicates a read access by statement s_1 of thread 1. When t_1 is different from t_2, memory access pair $(b_{1_{t_1,s_1}}, b_{2_{t_2,s_2}})$ is denoted as (s_1, s_2) in brief.

The following frequently-used patterns are considered:
Order violation:

(1) $R_{1,s_1} \rightarrow W_{2,s_2}$, (2) $W_{1,s_1} \rightarrow R_{2,s_2}$, (3) $W_{1,s_1} \rightarrow W_{2,s_2}$;
Atomicity violation:[2]
(4) $R_{1,s_1} \rightarrow W_{2,s_2} \rightarrow R_{1,s_3}$, (5) $W_{1,s_1} \rightarrow W_{2,s_2} \rightarrow R_{1,s_3}$,
(6) $W_{1,s_1} \rightarrow R_{2,s_2} \rightarrow W_{1,s_3}$, (7) $R_{1,s_1} \rightarrow W_{2,s_2} \rightarrow W_{1,s_3}$.

For example, pattern (1) means read access R_{1,s_1} wrongly occurs before the write access W_{2,s_2}. Pattern (4) means two read accesses R_{1,s_1} and R_{1,s_3} should be executed atomically, but are wrongly interrupted by the write access W_{2,s_2}.

Given a memory access pair (s_1, s_2) and its run-time information $I(e)$, LUCON tries to find the possible patterns that contain (s_1, s_2) by the following steps.

For atomicity violation, the access type of the memory access pair (s_1, s_2) is extracted. Based on the type, LUCON further searches the potential patterns in one concrete execution indicated by $I(e)$. For example, suppose that the access type of s_1 is W and the access type of s_2 is R, then the extracted type is $W \rightarrow R$. Since the type $W \rightarrow R$ involves in three atomicity violation patterns, LUCON successively searches each of them. For the first pattern $R_{1,s_1} \rightarrow W_{2,s_2} \rightarrow R_{1,s_3}$, LUCON tries to find the memory access R_{1,s_1} according to the following conditions: (1) it occurs before the W_{2,s_2} and R_{1,s_3} which can be checked by the timestamp ts; (2) it should be a read access; (3) it should also access the shared memory m; (4) it should belong to the same thread as R_{1,s_3}; (5) it should not be too far from R_{1,s_3}[3]. For the remaining two patterns, LUCON handles them similarly.

In the absence of atomicity violation pattern, LUCON checks the order violation pattern. Different from atomicity violation, nonbuggy executions are used to search for pattern $R_{1,s_1} \rightarrow W_{2,s_2}$. In most cases, this pattern will cause the program to crash when the read access R_{1,s_1} happens. Therefore, W_{2,s_2} cannot be found in the buggy executions. Conversely, the nonbuggy edge $W_{2,s_2} \rightarrow R_{1,s_1}$ usually appears in the nonbuggy executions.

3.3 Building bug triggering scenario

Buggy access pattern gives the essence of the bug, but still it cannot show how this bug happens. For instance, from the buggy access pattern in Figure 1(b), it is difficult for programmers to know how the execution process of thread 2 reaches to `sql_class.h150`, because `sql_class.h150` is executed in a small function and this small function is invoked by many functions. Meanwhile, programmers also do not know whether thread 2 is created between the two memory accesses in thread 1 or not.

LUCON reports a bug triggering scenario to help bug understanding. The bug triggering scenario provides a summary of functions and threads when programs fail. Specifically, call stack of each memory access in the pattern and the creation relationship between threads compose the scenario. The call stack gives the detailed function invoke process from thread start function to the memory access. The creation relationship between threads shows whether threads are independent and when the creation happens.

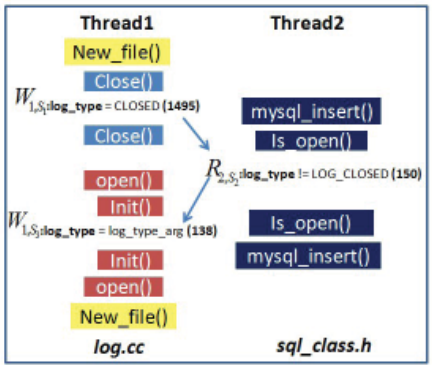


Figure 3. A bug triggering scenario example.

From the simplified bug triggering scenario in Figure 3, programmers can know the following information. Thread 1 executes the function `new_file()`. `new_file()` first invokes the function `close()` and `close()` executes the W_{1,s_1}. When `close()` returns, `new_file()` continues to invoke a new function `open()` and `open()` finally executes the W_{1,s_3} through function `init()`. But during the interval of W_{1,s_1} and W_{1,s_3}, thread 2 invokes the function `mysql_insert()` to execute R_{2,s_2}. With above information, programmers can roughly know most important events during the bug triggering process.

LUCON collects the information of call stack and thread creation during constructing the communication graphs. For each thread, every function call and return event are recorded with the timestamp that indicates when the event occurs. According to the timestamp of the memory access, all alive functions that have been called and have not returned, from thread start function to current memory access, can be found. Simultaneously, the parent thread and the parent thread's call stack when thread creation happens are stored. By recursively finding the parent thread, the relationship between threads can be determined.

4 Empirical study

We implemented LUCON based on Pin [2] and empirically evaluated it on a suit of buggy concurrent C/C++ programs. Two categories of buggy programs were selected. One is the bug kernels, which are constructed artificially (`Bankaccount`) or extracted from full version of Mozilla (`Httpconnectionw`, `Readwriteproc`). The other is the real client/server applications, including `Transmission`, `Pbzip2`, two versions of `Mysql` and

[2]We currently only consider the single-variable atomicity violation.
[3]The search distance can be preset.

Table 1. Evaluated applications and Locating results of LUCON and Recon

Program	LoCs	Bug Type	Recon		LUCON	
			Rank	Edges	Rank	Edges
Httpconnectionw	63	R→ W	2	25	1	1
Readwriteproc	60	W→ W	2	31	1	1
Bankaccount	136	R→ W→ W	24	46	1	3
Transmission	139k	R→ W	2	4386	1	2124
Pbzip2	2k	W→ R	2	484	1	175
Mysql-3596	415k	R→ W→ R	26	5062	1	4013
Mysql-791	372k	W→ R→ W	34	5142	1	3984
Apache	188k	W→ W→ R	23	3864	1	2314

`Apache`. The lines of code and the bug type are shown in Column 2-3 in Table 1. We ran our experiment on a computer with an 8-core 2.27GHz Intel Xeon processor and 12GB of memory, whose OS is Linux 2.6.32. Details about the implementation can be found in [1].

4.1 Effectiveness of LUCON

The goal of this study is to investigate how well LUCON ranks bugs. For a fair comparison to Recon, LUCON also collects 25 buggy runs and 25 nonbuggy runs. `sleep` primitive was inserted into the programs to increase the frequency of the bug appearance.

Table 1 shows the locating results of LUCON and Recon. Experiments of Recon are conducted on the prototype tool provided by Recon's developer. For all programs, LUCON locates the buggy access pattern accurately and ranks all these patterns top. Recon performs not well in some applications. For example, in `Mysql-791`, the real bug is ranked 34th. Column 5 and Column 7 in Table 1 show the number of edges used in bug locating processes of Recon and LUCON respectively. The result shows that the number of edges unrelated to the buggy behavior is rather high in most cases.

4.2 Bug report of LUCON

Figure 4 provides LUCON's report about the bug in `Transmission`. The bug report consists of two components: one is the overall description, the other is the bug triggering scenario.

`Transmission case study.` The first part (Lines 1-4) is the overall description. Line 1 shows that this bug may be an order violation. Line 2 shows the bug type is `R→W`. Line 3 and Line 4 give the information about the two nodes contained in this order violation. For example, for the first read access, the position of this access is `peer-mgr.c:2336`, i.e. it is at line 2336 in file `peer-mgr.c`. The string in the parenthesis is the name of the function which the access belongs to.

```
1: This may be an order violation bug
2: One possible buggy access pattern is: R⟶W
3: Fist access: peer-mgr.c:2336(bandwidthPulse())
4: Second access: session.c:285(tr_sessionInitFull())

5: The source access's call stack is
6: In trevent.c:305: timerCallback() call bandwidthPulse()
7: In event.c:387: event_process_active() call timerCallback()
8: In event.c:539: event_base_loop() call event_process_active()
9: In event.c:463: event_loop() call event_base_loop()
10: In event.c:401: event_dispatch() call event_loop()
11: In trevent.c:249: libeventThreadFunc() call event_dispatch()
12: In platform.c:123: ThreadFunc() call libeventThreadFunc()
13: Start Function of thread 1 is: ThreadFunc()(platform.c:114)
14: pthread_create()
15: In platform.c:154 tr_threadNew() call above function
16: In trevent.c:268 tr_eventInit() call above function
17: In session.c:268 tr_sessionInitFull() call above function
18: In cli.c:359 main() call above function
19: Parent thread Id is: 0

20: The sink access's call stack is
21: In cli.c:359: main() call tr_sessionInitFull()
22: Start Function of thread 0 is: main()
```

Figure 4. Bug Report for Transmission.

The second and the third parts(Lines 5-19) are the information of call stack and thread creation for the first read access. For these two parts, we should look at them from bottom to top. Lines 6-13 are the call stack information in the read access's thread. The function invoke process shows programmers how the execution of thread 1 reaches to the read access from the start function `ThreadFunc()` in `platform.c`. Through function names, programmers can roughly know the access happens during the process when thread 1 handles an event. Lines 14-19 show that thread 0 created thread 1. And the thread creation process is `main`→`tr_sessionInitFull()`→`tr_eventInit()` → `tr_threadNew()`→`pthread_create()`.

The fourth part (Lines 20-22) is the call stack information for the write access. It shows that the write access is executed in function `tr_sessionInitFull()` by thread 0. Because it is thread 0, it has no thread creation information.

From the thread creation information of thread 1(part3) and the write access's call stack information(part4), programmers know that thread 0 creates an asynchronous thread by invoking the function `tr_eventInit()` in function `tr_sessionInitFull()` of thread 0. When `tr_eventInit()` returns, thread 0 sequentially executes the write access in function `tr_sessionInitFull()`. Program crashes when the read access in the created thread 1 happens before the write access in thread 0.

For this bug, although Recon ranks it 2nd, Recon only gives the read access. It is hard for programmers to find the corresponding write access. Even if a tool can provide the two memory accesses to programmers, it is still hard

to understand how the bug happens due to the complexity of function calls. However, the above bug report shows that LUCON finds both the read and the write access, and further LUCON gives a bug triggering scenario to clearly show how the bug happens.

5 Related work

Concurrency bug locating and debugging. Ctrigger [13] uses some profiling runs to collect potential atomicity violations, forces the top ranked potential atomicity violations to execute and checks whether the bug appears. DefUse [16] monitors the definition-use pairs, extracts definition-use invariants from program runs and reports the invariant violations related to bugs. CCI [6] monitors and samples shared memory locations and locates the buggy locations using statistical method. Falcon [15] directly ranks patterns to identify potential concurrency bugs. Context-aware communication graph based method is proposed in [10][11] to debug concurrency bugs. The graph method considers buggy behaviors as abnormal edges in the graphs, i.e. edges that often appear in buggy graphs and never or rarely appear in nonbuggy graphs. Our work extends the context-aware communication graph to the edge-labeled communication graph, and improves locating accuracy by integrating reduplicative edges and reducing redundant edges.

Bug understanding. There exist some methods [3][4][5], which provide some kinds of execution contexts, such as data flow or control flow, to help understand the bugs in sequential programs. Recon [11] provides an aggregate reconstruction to try to help understand concurrency bugs. However, LUCON provides buggy access patterns and bug triggering scenarios to help programmers understand the bugs.

6 Conclusion

This paper presented LUCON, a new edge-labeled communication graph based locating technique to help understand concurrency bugs. LUCON first locates the buggy memory access pair and then finds buggy access patterns and constructs bug triggering scenarios to help programmers understand the bugs. There are two aspects of future work. One is to provide visual results to further facilitate programmers. The other is to handle multi-variable concurrency bugs.

Acknowledgement

This work was supported by the funds of the State Key Laboratory of Software Development Environment SKLSDE-2011ZX-07 and SKLSDE-2012ZX-18.

References

[1] *http://code.google.com/p/lucon/*.
[2] *http://www.pintool.org/*.
[3] H. Cheng, D. Lo, Y. Zhou, X. Wang, and X. Yan. Identifying bug signatures using discriminative graph mining. ISSTA '09, pages 141–152, New York, NY, USA, 2009. ACM.
[4] H.-Y. Hsu, J. A. Jones, and A. Orso. Rapid: Identifying bug signatures to support debugging activities. ASE '08, pages 439–442, Washington, DC, USA, 2008. IEEE Computer Society.
[5] L. Jiang and Z. Su. Context-aware statistical debugging: from bug predictors to faulty control flow paths. ASE '07, pages 184–193, New York, NY, USA, 2007. ACM.
[6] G. Jin, A. Thakur, B. Liblit, and S. Lu. Instrumentation and sampling strategies for cooperative concurrency bug isolation. OOPSLA '10, pages 241–255, New York, NY, USA, 2010. ACM.
[7] W. Li and N. Li. A formal semantics for program debugging. *Sci China Inf Sci*, 55(1):133–148, Jan. 2012.
[8] S. Lu, S. Park, E. Seo, and Y. Zhou. Learning from mistakes: a comprehensive study on real world concurrency bug characteristics. ASPLOS XIII, pages 329–339, New York, NY, USA, 2008. ACM.
[9] S. Lu, J. Tucek, F. Qin, and Y. Zhou. Avio: detecting atomicity violations via access interleaving invariants. ASPLOS XII, pages 37–48, New York, NY, USA, 2006. ACM.
[10] B. Lucia and L. Ceze. Finding concurrency bugs with context-aware communication graphs. MICRO 42, pages 553–563, New York, NY, USA, 2009. ACM.
[11] B. Lucia, B. P. Wood, and L. Ceze. Isolating and understanding concurrency errors using reconstructed execution fragments. PLDI '11, pages 378–388, New York, NY, USA, 2011. ACM.
[12] A. Muzahid, N. Otsuki, and J. Torrellas. Atomtracker: A comprehensive approach to atomic region inference and violation detection. MICRO '43, pages 287–297, Washington, DC, USA, 2010. IEEE Computer Society.
[13] S. Park, S. Lu, and Y. Zhou. Ctrigger: exposing atomicity violation bugs from their hiding places. ASPLOS XIV, pages 25–36, 2009.
[14] S. Park, R. Vuduc, and M. J. Harrold. A unified approach for localizing non-deadlock concurrency bugs. ICST '12, pages 51–60, Washington, DC, USA, 2012. IEEE Computer Society.
[15] S. Park, R. W. Vuduc, and M. J. Harrold. Falcon: fault localization in concurrent programs. ICSE '10, pages 245–254, New York, NY, USA, 2010. ACM.
[16] Y. Shi, S. Park, Z. Yin, S. Lu, Y. Zhou, W. Chen, and W. Zheng. Do i use the wrong definition?: Defuse: definition-use invariants for detecting concurrency and sequential bugs. OOPSLA '10, pages 160–174, New York, NY, USA, 2010. ACM.
[17] F. Sorrentino, A. Farzan, and P. Madhusudan. Penelope: weaving threads to expose atomicity violations. FSE '10, pages 37–46, New York, NY, USA, 2010. ACM.

Multiple Coordinated Views to Support Aspect Mining Using Program Slicing

Fernanda Madeiral Delfim and Rogério Eduardo Garcia
Departamento de Matemática e Computação
Faculdade de Ciências e Tecnologia
Universidade Estadual Paulista "Júlio de Mesquita Filho" – UNESP
Presidente Prudente - SP, Brazil
fer.madeiral@gmail.com, rogerio@fct.unesp.br

Abstract

Aspect Mining and Refactoring to Aspects aim to identify crosscutting concerns and encapsulate them in aspects, respectively. Aspect Mining remains as non-automatic process, i.e., the user needs to analyze and understand the results generated by techniques/tools, and confirm crosscutting concerns to refactor them to aspects. In this paper we propose a visual approach that deals with results generated by two aspect mining techniques proposed in the literature. By coordinating visual mappings, different levels of detail to explore software artifacts support aspect mining facilitating their interpretation for further refactoring to aspects. The model to coordinate multiple views was implemented ($SoftViz4AspectMining$ tool) and in this paper are presented the visualizations obtained, how to interpret them and lessons learned.

Keywords: Program Understanding, Aspect Mining, Refactoring to Aspects, Program Slicing, Software Visualization, Software Evolution.

1. Introduction

Modularization of *Crosscutting Concerns* (CCs) is still a challenge, since their implementations tend to be *scattered over* several system units and *tangled* with others concerns [11, 20, 14], what is a problem to understandability of software system, and as result make difficult the maintainability [4]. *Aspect-Oriented Programming* [10] provide mechanisms and abstractions to modularize CCs, encapsulating them in a separate code unit named *aspect*, increasing the modules cohesion [5] and possibility to code reuse [11]. Increasing modules cohesion makes easier program understanding activities, since it helps to focus on comprehending specific functionality implemented in separated units, supporting the software maintenance.

The evolution of existing software systems to aspect-oriented technology is also a challenge. First, it is necessary to identify CCs in non-aspect-oriented program that potentially could be turned into aspects and, then, decide about that. *Aspect Mining* is a research area aimed at CCs identification. Several techniques have been proposed to aspect mining [1, 2, 3, 8, 9, 12, 13, 22, 23]. In general, such techniques have similar limitations, like: *poor precision* of results (i.e., the percentage of relevant aspect candidates reported by a given technique is relatively low); *subjectivity* to analyze the results because the ambiguity on results produced (what on software engineer as CC candidate, other would not); *difficulty on comparing* the results of different techniques; and *difficulty on composing* of results from multiples aspect mining techniques. *Inadequate presentation of results* is pointed out as possible cause of such limitations [15]. In this context, approaches to deal with limitations of current techniques for mining aspects can be useful.

Software Visualization can be helpful on interpreting and analyzing results of aspect mining techniques, making explicit how the source code is organized, how different levels of abstraction are associated and how such associations take place in source code. Several approaches and tools have been proposed to support program comprehension using software visualization techniques, like Asbro [18], CRISTA [19] and $SoftViz_{OAH}$ [6], but they do not support CC identification.

In this paper we propose a visual approach to support aspect mining from existing object-oriented program. The proposed approach includes visualize program slicing results, allowing the visualization of control and data dependencies at statements level. The multiple coordinated views are used to allow analyzing the program structure, the associations at class level and method level based on callings, and how the slices are composed across multiple classes. We choose program slicing and added some features of the fan-in analysis in order to facilitate understanding and analyzing. Our focus is investigate whether visu-

alization contributes on understanding the results generated by those techniques (program slicing and fan-in analysis) in a complementary way. A Software Visualization tool, named $SoftViz4AspectMining$, was developed to support the visual approach proposed.

The remainder of this paper is organized as follows. In Section 2 is presented an overview about aspect mining techniques chosen. In Section 3 is presented the visual approach proposed (the coordination model) and some consideration about the developed tool. In Section 4 are presented the visual results obtained and we show how to interpret them in Section 5. Finally, the conclusions and future works are summarized in Section 6.

2. Related Works

There are several techniques proposed for mining aspects in the literature [1, 2, 3, 8, 9, 12, 13, 22, 23]. Katti et al. [9] discussed about the use of program slicing to help the process of inserting aspects in a object-oriented program. According to them, the slicing process consists of identifying the statements that form the slice (slice identification) and isolating these statements into an independent program (slice extraction). Thus, they seems very much similar to the Aspect Mining and Refactoring to Aspects process. We consider program slicing visualization promising because visual presentations can externalize slice overlapping and user interaction can be useful to deal with. Due to pages restriction, we do not present slicing techniques (see [21]).

Additionally, we also use fan-in metric mapped to visual attribute. Fan-in analysis tries to capture the scattered code at method level [16, 17]. Scattered code is a symptom of crosscuttingness because called methods might implement CCs and, then, those methods are called from many places, resulting in high fan-in value. In an aspect-oriented reimplementation of such concerns, the method would constitute an advice, and the call would correspond to the context that needs to be captured using a pointcut. The steps for fan-in analysis are: compute fan-in metric for all methods; filter relevant methods; and analyze them to determine which are CC candidates. To the last step, guidelines can be used to look for consistent invocations of methods with high fan-in value from call sites that could be captured by a pointcut definition – for example, the calls always occur at the beginning or at the end of a method [13].

3. The Coordination Model Proposed

In this paper we propose a visual approach to support aspect mining implemented in a desktop Software Visualization tool, named $SoftViz4AspectMining$. The $SoftViz4AspectMining$ architecture is organized in three layers as depicted in Figure 1.

The proposed visual approach consists in a set of visual presentations organized into *Visualization Layer* (see Figure 1) to show program features – *Aggregate View*, *Treemap View*, *Class Level Hyperbolic View*, *Method Level Hyperbolic View*, *Bars and Stripes View* and *Bytecode View*. Using a program bytecode as input, at *Data Layer* are performed static and dynamic analyzes to gather information from code, kept into data structures used to generate the visual presentations at *Visualization Layer* and to coordinate them by *Control Layer*. The multiple coordinated views enable a visual exploration of software in different levels of detail.

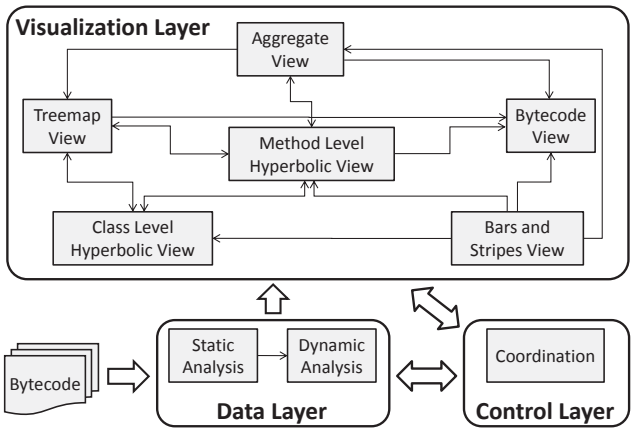


Figure 1. $SoftViz4AspectMining$ **architecture.**

The *Aggregate View* shows graphs representing instructions (vertices) linked by control flow and control and data dependency (edges) projected into regions (representing methods). Slices can be visualized and, by interacting with the *Aggregate View*, the user can choose a visual item that represents a specific statement (slice criterion) and slice type (static/dynamic and backward/forward). Then, another *Aggregate View* is created to present the slice using the parameters chosen.

The colors of visual items are predefined in each visual presentation, but the user can modify such color definition. In addition, there is a color mapping based on method calls used across several views – for each unit (i.e., packages, classes and methods) is computed the fan-in value and a color gradient from light gray to red is used to represent the obtained values (red represents the greater value). One may filter code units based on fan-in value by defining a threshold and observe them in projections and, then, it is possible to visualize units with high frequency of callings.

4. Results: Visual Presentations

The results obtained with $SoftViz4AspectMining$ are shown using a oriented-object program of an elevator simulation [7]. In Figures 2 and 3 are shown two hyperbolic projections based on method calls: the first at class level and the second at method level. The nodes represent units (classes or methods), and the direct edges represent the calling between them. The hyperbolic projection shows a node in focus (chosen by user) larger than the other and positioned in center. The neighbor nodes are colored in pink, and other nodes are no colored. Such projections help to have evidences of possible CCs, and if user decide to refactoring methods to aspect, help identify which units are affected – by analyzing the calls shown at method level it is possible to observe the method to be considered on creating advice and the ones that should be crosscut.

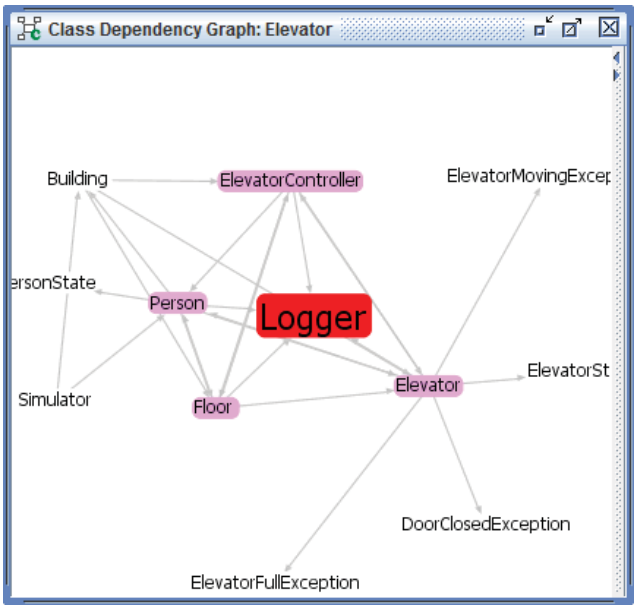


Figure 2. *Class Level Hyperbolic View*.

The *Treemap View* presents the program structure hierarchically by nested rectangles (see Figure 4). The outer rectangle represents the whole program (root). Other unit levels are packages, classes, methods and test cases, represented by inner rectangles. The rectangles are colored using a simple linear gradient of gray (dark gray represents whole program). Each leaf rectangle represents a method and its size is proportional to its amount of bytecode instruction.

In the *Aggregate View*, program instructions are represented by nodes – each node is a group of bytecode instructions. Also, the nodes are aggregated by methods, represented by curved polygon (named *aggregate region*) using specific color by method (we used transparency because of overlapping). There are two types of nodes: physical

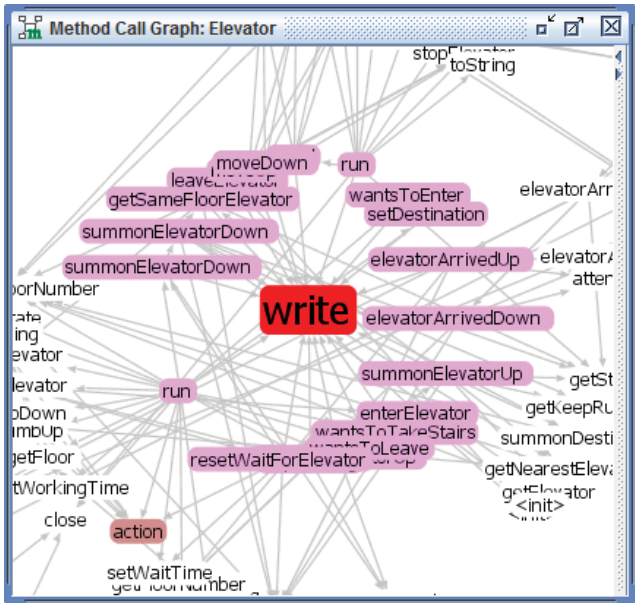


Figure 3. *Method Level Hyperbolic View*.

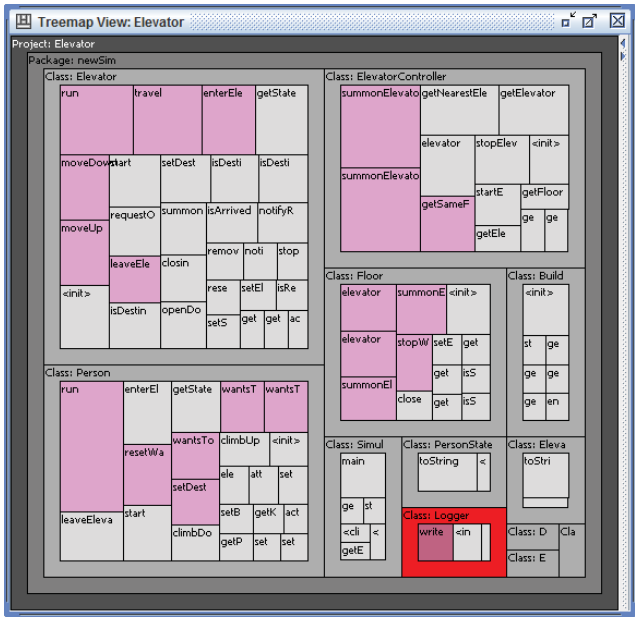


Figure 4. *Treemap View*.

nodes and virtual nodes. The physical nodes are program statements (colored in pink). The virtual nodes are introduced to represent methods entries (colored in dark green), methods exits (colored in light green) or parameters (colored in black). The edges represent control flow (continuous lines colored in black), exception (dashed lines colored in black), control dependency (continuous lines represent intra-method and dashed lines represent inter-method, both

colored in blue) or data dependency (continuous lines represent intra-method and dashed lines represent inter-method, both colored in green). *Aggregate View* allows visual exploration at statement level – statements are projected within the methods in which they are. *Aggregate Views* are generated to selected method – in Figure 5 is depicted the *Aggregate View* to method *main*. Focusing on specific node, the user can apply filters to visualize slices obtained both static/dynamic and backward/forward as subgraph.

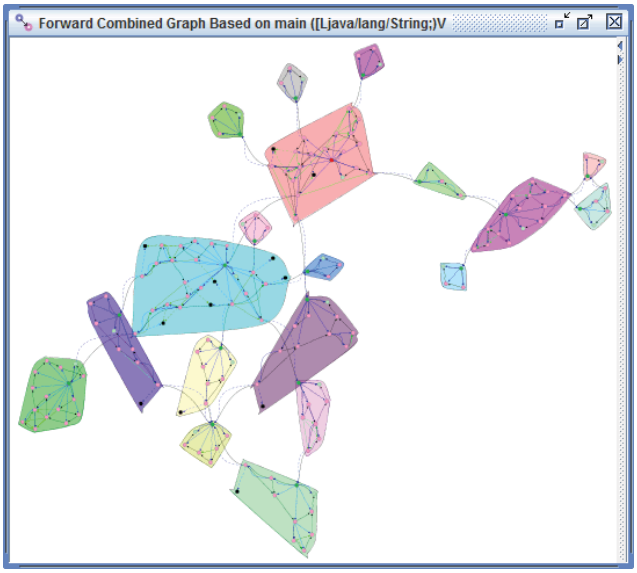


Figure 5. *Aggregate View*.

In Figure 6 is shown the *Bars and Stripes View*. The bars represent classes – the height of each bar is proportional to amount of bytecode instruction. The stripes represent program instructions using different colors assigned to each slice. Such view provides an overview of how a slice is distributed across multiple classes. Also, the $SoftViz4AspectMining$ allows visualizing bytecode instruction using two synchronized scrolling: right-scrolling to overview and left-scrolling to detail. In the right-scrolling the rectangles represent a set of bytecode instruction defined by labels in bytecode program. In the left-scrolling are shown bytecode instructions of selected rectangles in right-scrolling, grouped method which instructions belong to. Similar to *Aggregate View*, each method is shown by different color: in Figure 7 is shown the method `Logger.write (Ljava/lang/String;)V` represented in red, and other method is represented in blue. The color mapping is the same of *Aggregate View*.

5. Analyzing the Visual Presentations

In this section we show how to analyze the visual presentations. The frequency method calls is computed at level of

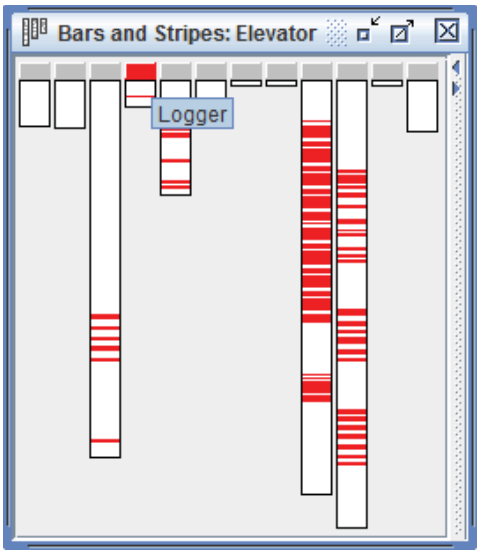


Figure 6. *Bars and Stripes View*.

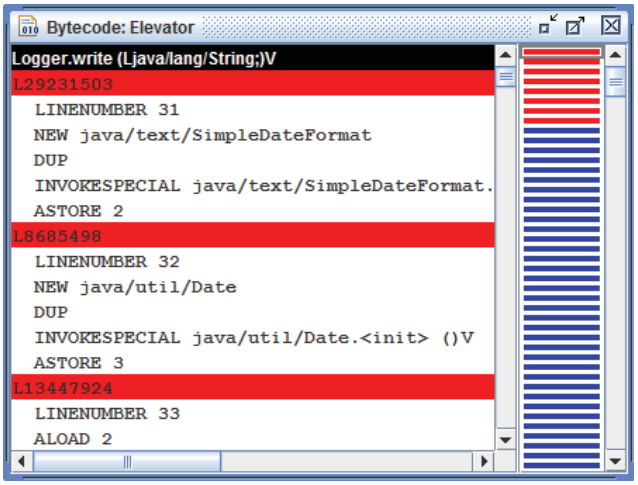


Figure 7. *Bytecode View*.

package, class/inner class/test case and method in order to create the color gradient aforementioned. In the *Class Level Hyperbolic View* (Figure 2) is shown the most called class (`Logger`) highlighted in red. The user can refine the results of class level view and methods are highlighted at *Method Level Hyperbolic View*. At method level it is also possible to use another color scale to highlight methods with high frequency call. In addition, such color scale can be used to establish a minimum threshold value to frequency call in order to discard methods. In Figure 3 are colored both `write` and `action` methods (it was considered 17 callings as minimum frequency[1]). Based on that, the user can

[1]The minimum frequency is defined empirically, taken into account the source code.

observe explicitly the frequency of calling to each filtered method and decide whether it can be considered a CC candidate.

Method Level Hyperbolic View has fundamental role in the coordination model. By selecting a method of interest, it is possible see the dependency of its class in the *Class Level Hyperbolic View* and, also it is possible to observe the selected method and the related ones at program hierarchical structure shown in *Treemap*. The *Treemap View* is not decisive for CCs identification, but it is interesting to have an overview of the program structure and observe the code units (classes and methods) that can be affected by refactoring. In Figure 4 is shown the hierarchical structure highlighting the method selected in Figure 3 (its class in red) and the related methods (called and callers) in pink. In addition, details of control and data dependency of selected method statements can be analyzed using *Aggregate View*, as well as the bytecode instructions in *Bytecode View* (see Figure 7).

The *Aggregate View* shows a graph at statements level. Each method has its own graph, and graphs can be combined by method calls statements from or to a selected method (forward or backward). A graph from a method is used to create intra-method slice, and combined graph is used to create a inter-method slice. The slices can be static or dynamic, from or to a selected node (forward or backward slice). In Figure 8 is shown a static backward slice generated from the backward combined graph based on method `write` (represents by *aggregate region* (1)). The slice criterion selected is the statement (2) and the entry node to `write` method is (3). In Figure 8 was used distance filter equal to two, from entry node (3). One may observed: 1) the statements that call directly the method owner of the slice criterion analyzing control dependency edges (blue); 2) the parameter used in the slice criterion by data dependency edges (green). Thus, the user can define pointcuts to capture joinpoints. In addition, an overview of how a slice is distributed across multiple classes is provided in Figure 6 (statements from a slice are colored in red, and the `Logger` class highlighted according color scale at class level).

By interacting with visual presentation, the user can gather information about methods and decide about refactoring to aspect or not. Methods considered CCs candidates, its class and methods callers, and how they are related must be analyzed at high and low level. At low level, the user can observe the control and data dependency, what is important to evaluate how a CC can be implemented into an aspect (for example, the aspect should have access to method data?).

In addition, we observed some considerations of program slicing. A program slicing algorithm convert the program statements into some alternative representation to calculate a slice. Since a slice consists of all the statements of a program that may affect the values of some variables in a set V at some point of interest p (slicing criterion), independently of the representation generated by program slicing algorithm, the control and data dependencies should be calculated. Thus, program slicing can be useful to help the Refactoring to Aspect process, not only to extract a slice, but also to help to define pointcuts (a step of Refactoring to Aspects).

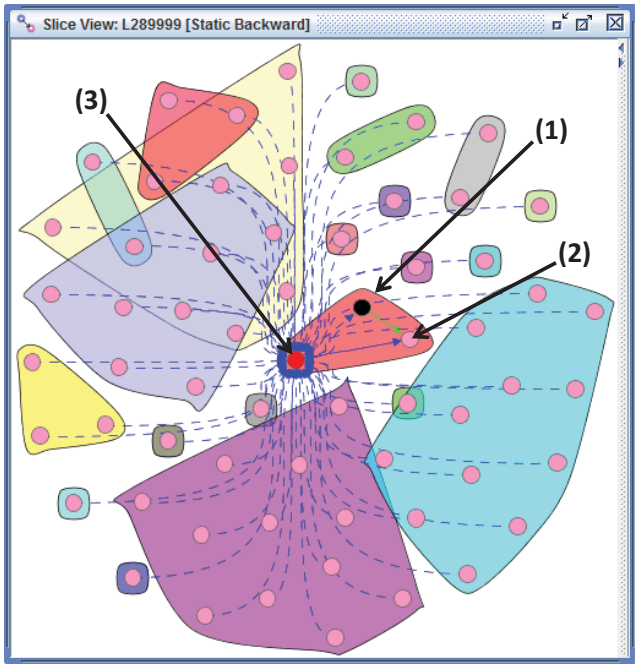


Figure 8. Slice Visualization in the *Aggregate View*.

6. Conclusions and Future Works

In this paper we propose a visual approach that consists in coordinating multiple views, using a color mapping based on fan-in metric and a slices visualization, enabling: 1) visual exploration of software in different levels of detail; 2) visualization of the methods more frequently called; 3) control and data dependency analysis to define pointcuts.

We show the $SoftViz4AspectMining$ tool that was developed to support the multiple coordinated views model. Our pilot study has shown that the model proposed supports analyzing by exploring different levels of detail, helps to deal with the limitations of projections (scalability of visualization techniques) by user interaction (applying distance filter in the graphs views, for example), and allows pointing out the visual results and observations (lessons learned) presented here. However, to evaluate the effectiveness of coordinated visual model proposed, a controlled experiment has been planned, aims to identify CCs in object-oriented program. Also, we intend to evaluate the tool focusing on

defining pointcuts (how effective is the help to refactor to aspects). The initial study was very important to identify possible treats to controlled experiment.

In addition, we intend to implement other aspect mining techniques, to combine them in a complementary way. For example, *link analysis* [8] identifies CCs at class level, and may be advantageous make explicit the analysis result on *Class Level Hyperbolic View*, as well as the result of *clone detection* in the *Aggregate View*, with slices information.

References

[1] E. S. Abait, S. A. Vidal, and C. A. Marcos. Dynamic Analysis and Association Rules for Aspects Identification. In *Proceedings of the II Latin American Workshop on Aspect-Oriented Software Development (LA-WASP '08)*, pages 31–39, October 2008.

[2] S. Breu and J. Krinke. Aspect Mining Using Event Traces. In *19th IEEE International Conference on Automated Software Engineering (ASE '04)*, pages 310–315, Washington, DC, USA, September 2004. IEEE Computer Society.

[3] M. Bruntink, A. van Deursen, R. van Engelen, and T. Tourwe. On the Use of Clone Detection for Identifying Crosscutting Concern Code. *IEEE Transactions on Software Engineering*, 31(10):804–818, 2005.

[4] M. Ceccato, M. Marin, K. Mens, L. Moonen, P. Tonella, and T. Tourwé. Applying and combining three different aspect Mining Techniques. *Software Quality Control*, 14(3):209–231, September 2006.

[5] C. Clifton, G. T. Leavens, and J. Noble. MAO: Ownership and Effects for More Effective Reasoning About Aspects. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP '07)*, pages 451–475. Springer-Verlag, 2007.

[6] A. F. d'Arce, R. E. Garcia, R. C. M. Correia, and D. M. Eler. Coordination Model to Support Visualization of Aspect-Oriented Programs. In *Proceedings of the 24th International Conference on Software Engineering and Knowledge Engineering (SEKE '12)*, pages 168–173, 2012.

[7] H. Do, S. G. Elbaum, and G. Rothermel. Supporting Controlled Experimentation with Testing Techniques: An Infrastructure and its Potential Impact. *Empirical Software Engineering: An International Journal*, 10(4):405–435, 2005.

[8] J. Huang, Y. Lu, and J. Yang. Aspect Mining Using Link Analysis. In *Proceedings of the 5th International Conference on Frontier of Computer Science and Technology (FCST '10)*, pages 312–317, Washington, DC, USA, 2010. IEEE Computer Society.

[9] A. Katti, V. Bingi, and V. Chavan. Application of Program Slicing for Aspect Mining and Extraction - A Discussion. *International Journal of Computer Applications*, 38(4):12–15, January 2012.

[10] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold. An Overview of AspectJ. In *Proceedings of the 15th European Conference on Object-Oriented Programming (ECOOP '01)*, pages 327–354, London, UK, UK, 2001. Springer-Verlag.

[11] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-Oriented Programming. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP '97)*, pages 220–242. Springer-Verlag, 1997.

[12] J. Krinke and S. Breu. Aspect Mining Based on Control-Flow. In *Proceedings of the 7th Workshop Software Reengineering (WSR '05)*, volume 25, pages 39–40, Bad Honnef, Germany, May 2005. GI-Softwaretechnik-Trends.

[13] M. Marin, A. V. Deursen, and L. Moonen. Identifying Crosscutting Concerns Using Fan-In Analysis. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 17(1):3:1–3:37, December 2007.

[14] P. Massicotte, L. Badri, and M. Badri. Towards a Tool Supporting Integration Testing of Aspect-Oriented Programs. *Journal of Object Technology*, 6(1):67–89, 2007.

[15] K. Mens, A. Kellens, and J. Krinke. Pitfalls in Aspect Mining. In *Proceedings of the 15th Working Conference on Reverse Engineering (WCRE '08)*, pages 113–122, Washington, DC, USA, October 2008. IEEE Computer Society.

[16] A. Mubarak, S. Counsell, and R. M. Hierons. An Evolutionary Study of Fan-in and Fan-out Metrics in OSS. In P. Loucopoulos and J.-L. Cavarero, editors, *Proceedings of the 4th IEEE International Conference on Research Challenges in Information Science (RCIS '10)*, pages 473–482. IEEE, May 2010.

[17] E. Nasseri, S. Counsell, and E. Tempero. An Empirical Study of Fan-In and Fan-Out in Java OSS. In *Proceedings of the 8th International Conference on Software Engineering Research, Management and Applications*, pages 36–41, Los Alamitos, CA, USA, 2010. IEEE Computer Society.

[18] J.-H. Pfeiffer and J. R. Gurd. Visualisation-Based Tool Support for the Development of Aspect-Oriented Programs. In *Proceedings of the 5th International Conference on Aspect-Oriented Software Development (AOSD '06)*, pages 146–157, New York, NY, USA, 2006. ACM.

[19] D. Porto, M. G. Mendonça, and S. C. P. F. Fabbri. CRISTA: A Tool to Support Code Comprehension Based on Visualization and Reading Technique. In *Proceedings of the 17th IEEE International Conference on Program Comprehension (ICPC '09)*, pages 285–286, 2009.

[20] C. N. Sant'Anna, A. F. Garcia, C. v. F. G. Chavez, C. J. P. de Lucena, and A. v. Staa. On the Reuse and Maintenance of Aspect-Oriented Software: An Assessment Framework. In *Proceedings XVII Brazilian Symposium on Software Engineering*, 2003.

[21] F. Tip. A Survey of Program Slicing Techniques. *Journal of Programming Languages*, 3(3):121–189, 1995.

[22] P. Tonella and M. Ceccato. Aspect Mining Through the Formal Concept Analysis of Execution Traces. In *Proceedings of the 11th Working Conference on Reverse Engineering (WCRE '04)*, pages 112–121, Washington, DC, USA, 2004. IEEE Computer Society.

[23] T. Tourwé and K. Mens. Mining Aspectual Views Using Formal Concept Analysis. In *Proceedings of the 4th IEEE International Workshop Source Code Analysis and Manipulation (SCAM '04)*, pages 97–106, Washington, DC, USA, 2004. IEEE Computer Society.

How Does Acquirer's Participation Influence Performance of Software Projects: A Quantitative Analysis

Yasha Wang[1,2,3], Jiangtao Wang[1,2], Jiakuan Ma[1,2], Bing Xie[1,2,3]
1 School of Electronics Engineering and Computer Science, Peking University, Beijing 10087, China
2 National Engineering Research Center of Software Engineering, Peking University, China
3 Beijing Beida Software Engineering Development Co., Ltd.
{wangys, xiebing, wangjt10, tanghao13, majk06}@sei.pku.edu.cn

***Abstract*- In custom software development projects, software acquirer always participates in software development processes acting as collaborator, reviewer or supervisor. The acquirer participation has a significant influence on project performance. In practice, the plans of acquirer participation are often made by project managers according to their personal experience or intuition, and the quality of the plan is difficult to guarantee. Moreover, because of lacking objective evaluation criterion for the plan, it's hard for the acquirer and the supplier to reach an agreement or compromise when they have different opinions. To solve this problem, this paper collects historical data of 25 software projects and establishes a quantitative model of relationship between the acquirer participation and project performance by leveraging a variable-selection-based regression analysis approach. The quantitative model provides guidance for making acquirer participation plans. Furthermore, a review of 9 experienced project managers was conducted to evaluate the model's validity.**

Keywords- software acquirer, project performance, quantitative analysis, software process

I. INTRODUCTION

Acquirer plays important role in software lifecycle processes especially for custom software development projects. According to the IEEE Std 12207-2008, acquirer is stakeholder that acquires or procures a product or service from a supplier. Commonly, the acquirer participates in software development processes, acting as collaborator, reviewer or supervisor. Literature survey indicates that acquire participates in many activities of software development processes. For example, Subramanyamon's study [4] of 117 software projects indicates that the acquirers take part in every phase of the software lifecycle and averagely 22% of the whole time duration of the project. A questionnaire survey conducted by [5] indicates that software acquirer participates in various activities of software development processes such as planning, requirement elicitation and coding. Some other existing works also demonstrate that the acquirer participation has a significant impact on project performance [2] [3] . Proper acquirer participation improves the project performance [6] [7] [8] , or in other words, has positive influence on the project performance, and on the other hand, over- or under participation has negative influence on the project performance[1] [9] .

Through a questionnaire among 35 software enterprises and 84 project managers, we found that in software projects, acquirers are not tend to lead the participation plan making task, due to their lack of professional knowledge on software technology. On the other hand, as having different perspectives or interests, the participation plans made by suppliers are often difficult to gain acquirers' approval. In these situations, an objective criterion to evaluate the plan is needed.

According to the above motivation, a quantitative model of the relationship between acquirer participation and project performance is built in the follow steps:

Step 1, data collection and quantification. Data are collected from 25 custom software development projects, which are already finished and from the same business domain, e-government.

Step 2, data analysis. In the quantitative model we want to establish, the dependent variable is the project performance and the independent variable is the degree that the acquirer participated in a specific activity in software development processes. This paper proposes a variable-selection-based regression analysis approach. It first selects the independent variables that are most related and influential to the dependent variables by leveraging dantzig selector [13] , then it eliminates those weak correlated independent variables, and build the model by leveraging ordinary least squares regression.

Step 3, interview and analysis. We invite 9 senior project managers to make comment on the model we built, discuss the connotations by taking data into consideration to prove the model's validity.

The rest of this paper is organized as follows. Related works are reviewed in section II. Data collection and quantification is described in section III . The data analysis approach and result are described in section IV. The interview with project managers is described in section V. Finally, the paper is concluded in section VI.

II. RELATER WORKS

Acquirer process, in contrast to supplier process, has relatively less studied. The acquirer of project is defined by IEEE Std 12207-2008 [10] , which provides acquisition process including a series of seven activities from acquisition preparation to closure. However, IEEE Std 12207-2008, as a process specification framework, does not provide detailed guidance for acquirer's participating in software development.

Exiting works chose a few macro indicators as independent variables and made some high-level conclusions. For example, [8] [9] select the degree of acquirer's participation as one single macro indicator to investigate the statistical correlation between this indicator and project satisfaction or overall performance. Though drawing valuable high-level conclusions, these researches are not able to provide detailed guidance for planning the acquirer's participation.

It is a common challenge in gene biography and some other disciplines to analyze data when the number of independent variables p is much smaller than sample size n. For example, there are tens of thousands DNA microarray while the number of patient samples is merely several thousand [11] [12] . In this case, variable selectors such as Danzig Selector [13] are able to reduce the number of variables by selecting main variables so as to lay the foundation for modeling. Inspired by this idea, this paper combines variables-selection with the regression approach in software process data analysis.

III. DATA COLLECTION AND QUANTIFICATION

A. Data Collection

From 11 Chinese software enterprises, we collect historical data in 25 custom software development projects in e-government domain. The project duration ranges from 10 months to 30 months, and the investment (or cost) ranges from 520 thousands Yuan to 3 million Yuan.

Data is collected by ways of questionnaire surveys, telephone and face-to-face interviews with people both from acquirers and providers, and data extraction from project documentations. The project documentations include project planning and tracking documents, meeting minutes of technical conversations and management reviews, project milestone reports and project conclusion reports.

The collected project data consist of three parts: project planning data, project performance data and acquirer's participation data. Project planning data include planning time duration and investment volume of the project, and the acquirer's preference on project performance. Project performance metrics includes time, costs and software quality [14] . Performance preference is the weight assigned to each of these three metrics, and the more important a metric is considered, the bigger weight is set to it. Project performance data include actual time duration, costs, completion rate and the acquirer's satisfaction degree to software product. Acquirer's participation data record what activities acquirers participated in, what role they played, and how much time they participated each time.

B. Project Performance Quantification

According to our collected data, many projects do not finish all functions that were planned to be built. Therefore, besides time, cost and software quality, we take project completion rate into account to evaluate project performance.

The project performance of project i, denoted as Yi is defined as follows:

$$Y_i = CR_i - COR_i \times w_{ci} - TDR_i \times w_{di} - QGR_i \times w_{qi}$$

(where $0 \le w_{ci}, w_{di}, w_{qi} \le 1$ and $w_{ci} + w_{di} + w_{qi} = 1$)

The w_{ci}, w_{di} and w_{qi} is the weight assigned to costs, time and software quality respectively. The metrics in the above equation is defined in table 1.

Table 1 metrics for the quantification of project performance

Metric	Meaning	Definition	Explanation
CR_i	Completion Ratio	$CR_i = \frac{CF_i}{PF_i}$	CF_i and PF_i are the finished and planned function points of project i respectively.
COR_i	Cost Overrun Ratio	$COR_i = \frac{AC_i - PC_i}{PC_i}$	AC_i and PC_i are the actual and planned cost of project i respectively.
TDR_i	Time Delay Ratio	$TDR_i = \frac{AD_i - PD_i}{PD_i}$	AD_i and PD_i are the actual and planned time duration of project i respectively.
QGR_i	Quality Grade Ratio	$QGR_i = \frac{MQG - QG_i}{MQG}$	MQS = 5 is the maximum score, and QS_i is the score of project i given by the acquirer.

The 25 project performance values of the 25 sample projects are formed to a 25-dimension project performance vector, denoted as Y_{25}.

C. Acquirer Participation Degree Quantification

Since the data are collected from different projects and different companies, the process model and terminology are different among projects. In order to solve this problem, we map the activities of the sample projects to the process framework specified in IEEE Std 12207-2008 [10] . Additionally, there are some common e-government-domain-specific activities in the sample projects, which are not defined in IEEE Std 12207-2008. In order to maintain the security and consistency of the data, the data migration activity needs the collaboration of both acquirers and suppliers. In this paper, 38 activities are concluded in 25 projects

For a given specific activity, acquirer may participate in different forms. We concluded 3 forms of acquirer participation, which is described in table 2.

Table 2 Three forms of acquirer participation

Form	Description	Example
Management	Participation to influencing project management	Tracing the project schedule
Technique	Participation to influencing technique solution	Attending technique solution review meeting
Business	Participation to influencing business domain knowledge	Providing domain knowledge in requirement elicitation

We define the concept of participation point as a 2-tuple of *(a, f)*, where *a* denotes a development activity that the acquirer participates while *f* denotes the participation form of a. There are 114 (38×3) total possible participation points in our sample data set. Furthermore, for a given participation point $PP_{i,j}$ (the participation point j in project i), we define its participation degree as $x_{i,j} = t_{i,j}/AD_i$, where $x_{i,j}$ is the sum of time that the acquirer spent in a certain participation point j of project *i*, and where AD_i is the actual duration of project i. The participation degree of 25 projects in 114 participation point can form a *Participation Matrix* $X_{25\times114}$.

In Matrix $X_{25\times114}$, if acquirer does not participate in activity a, the participation degrees of three related participation points are assigned as 0. Acquirers sometimes participate in an activity acting in multiple forms. In this occasion, we averagely assign the total participation time to corresponding participation points. In the sample data set, we found that there are 32 total participation points whose participation degrees are all zero for all 25 projects. These participation points are deleted from the participation matrix, and the matrix is reduced to $X_{25\times82}$.

IV. Data Analysis

A. Overview

In this paper, the goal of quantitative data analysis is to establish quantitative relationships between participation degree matrix $X_{25\times82}$ and project performance vector Y_{25}. The analysis approach is divided into two steps: Step 1: Dantzig Selector is used to select the most influential independent variables in participation degree matrix, and those non-selected variables are deleted from the data set. Step 2: Ordinary least squares regression (or OLS for short) is utilized to establish the quantitative model.

B. Variable-selection based on Dantzig Selector

The variable selection problem in this paper can be represented as the computation of a sparse vector $\beta \in R^{82}$, making $Y = X\beta + \varepsilon$, $\varepsilon \in R^{25}$ and $\varepsilon_i \sim i.d.d\ N(0, \sigma^2)$, $i=1, 2, ..., 25$.. As β is sparse, those non-zero elements indicate those selected variables. The Dantzig Selector proposed by Emmanuel Candes and Terence Tao in 2007 is adopted to solve this problem.

We use an open source implementation of Dantzig Selector (can be downloaded from http://www.acm.caltech.edu/l1magic/) to accomplish the computation of β. As required by the Dantzig Selector, the assumption has to be made about the number of variables needed to be selected (denoted as m). Based on [13] , if the number of samples data is n, the restriction of $m \le n/2$ has to be satisfied. In order to try our best to avoid the possible deletion of important variables, we set m=12, which is the maximum value permitted. After setting m, the following steps are performed:

(1) Initialize vector $result \in R^{82}$, set all its component to be 0;

(2) A column with all components to be 1 is added as the last column of $X_{25\times82}$, extending $X_{25\times82}$ to $X'_{25\times83}$;

(3) Randomly select 20 samples from the all 25 projects, and select corresponding lines or component of $X'_{25\times83}$ and Y_{25} to constructing $X^{sub}{}_{20\times83}$ and $Y^{sub}{}_{20}$;

(4) Select $x_0 = X^{subT} * (X^{sub} * X^{subT})^{-1} * Y^{sub}$, $e = 3 \times 10^{-4}$, $dtol = 5 \times 10^{-3}$, uses Dantzig Selector open-source implementation to compute $\tilde{\beta}'$ =l1dantzig_pd($x_0, X^{sub}, [\,], y^{sub}, e, pdtol$). $\tilde{\beta}' \in R^{83}$, as a sparse vector;

(5) Select the 82 left components of $\tilde{\beta}'$, making $\tilde{\beta} \in R^{82}$;

(6) Set the 12 biggest components of $\tilde{\beta}$ in absolute value to be 1, representing that corresponding independent variables are selected during this iteration, set other components to be 0.

(7) Denotes $result = result + \tilde{\beta}$;

(8) Executes (3)~(7) for 1000 times repeatedly, thus getting vector $result$.

The components of vector $result$ show total times that each variable has been selected in these 1000 random experiments (as shown in Figure 1). The 12 variables which are selected the most times have been chosen to be the selected variables.

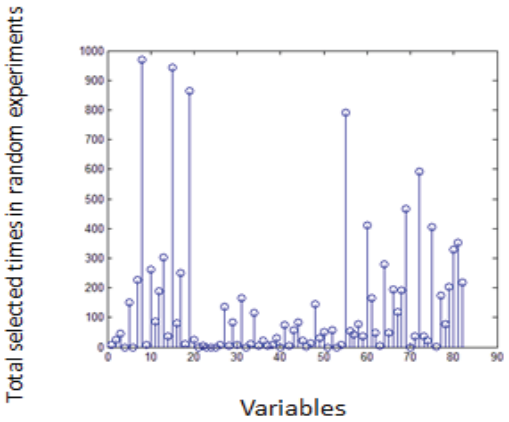


Fig.1Total time of each screened-out variable

C. Regression Analysis

Based on the variable selection result, we use 12 selected variables (denoted as a~l) to conduct OLS regression analysis, and the result is showed in figure 2.a. Coef. and _cons represents the gradient and intercept, respectively.

Source	SS	df	MS	Number of obs =	25
				F(12, 12) =	101.22
Model	.274882345	12	.022906862	Prob > F =	0.0000
Residual	.002715647	12	.000226304	R-squared =	0.9902
				Adj R-squared =	0.9804
Total	.277597992	24	.011566583	Root MSE =	.01504

y	Coef.	Std. Err.	t	P>|t|	[95% Conf.	Interval]
a	-.3550184	.0226927	-15.64	0.000	-.4044616	-.3055752
b	-.040897	.0188591	-2.17	0.051	-.0819874	.0001934
c	-.079719	.0077469	-10.29	0.000	-.0965981	-.0628399
d	.0184763	.0030358	6.09	0.000	.0118619	.0250908
e	.1323048	.0869781	1.52	0.154	-.0572043	.3218139
f	.0033093	.0101302	0.33	0.750	-.0187625	.0253811
g	.0440916	.0291368	1.51	0.156	-.0193921	.1075753
h	-.0340198	.0213832	-1.59	0.138	-.0806097	.0125701
i	-.0129583	.0031434	-4.12	0.001	-.0198072	-.0061095
j	-.0222482	.0277372	-0.80	0.438	-.0826823	.0381859
k	.0064459	.0035258	1.83	0.092	-.0012363	.0141281
l	.0278073	.0105363	2.64	0.022	.0048507	.0507639
_cons	.899255	.0175109	51.35	0.000	.8611019	.937408

(a) The 1st iteration

Source	SS	df	MS	Number of obs =	25
				F(5, 19) =	144.15
Model	.270468179	5	.054093636	Prob > F =	0.0000
Residual	.007129813	19	.000375253	R-squared =	0.9743
				Adj R-squared =	0.9676
Total	.277597992	24	.011566583	Root MSE =	.01937

y	Coef.	Std. Err.	t	P>|t|	[95% Conf.	Interval]
a	-.3447594	.0219539	-15.70	0.000	-.3907095	-.2988093
c	-.0977238	.0069554	-14.05	0.000	-.1122817	-.0831659
d	.0248959	.0026504	9.39	0.000	.0193486	.0304432
e	.2343965	.0240759	9.74	0.000	.1840051	.2847878
i	-.0190572	.0023798	-8.01	0.000	-.0240381	-.0140764
_cons	.9256581	.0137656	67.24	0.000	.8968463	.9544698

(b) The last iteration

Fig. 2 Results of the OLS regression

It is easy to observe that the P values of some variables are significantly bigger than others. The bigger the P value is, the less correlation between variable and project performance. In order to improve the accuracy of this model, we execute a simple algorithm as in figure 3 to further deleting variables. Here set θ=0.03, indicating that only when the statistical significance is bigger than 3%, a variable is deleted for being considered to have no relationship with the project performance.

1. Check whether there are variables with P value bigger than θ. If yes, go to step 2, or else exit.
2. Select variables with the biggest P value and delete it, then go to step 3.
3. Conduct OLS regression analysis based on rest of variables and return to step 1.

Fig.3 algorithm for deleting variables

Using the algorithm above, only five variable are left, and final model is PFMC=0.9357 − 0.3448·a − 0.0977·c+0.0249·d+0.2344·e − 0.0191·i And the Detailed explanations of variables are described in table 8.

Table 4 Detailed explanations of variables in conclusion

Variables	Detailed explanations
PFMC	project performance
a	Contract agreement in the form of business
c	Project planning in the form of management
d	Software requirements analysis in the form of business
e	Data migration regular meeting in the form of technique
i	Testing problem resolution in the form of management

D. Analysis of the model's mathematical properties

Accuracy of the model. As shown in figure 2.b R2 and Adj-R2 of OLS are close to 97%, indicating that these 5 variables have a good interpretation on y.

Completeness of the model. As a multivariate and linear regression analysis model is used, we have to check whether there are missing high-order terms. Ramsey RESET test is performed, and the result p = 0.9765 indicates that there are no missing high-order terms.

V. Interviews

The quantitative model in this paper shows that there are five participation points having strong correlation to project performance, two of which have positive influence while other three have negative influence. In order evaluate the validity of this model and understand its connotation, we invite 9 senior project managers and interview them through phone calls or e-mails. The contents of interview focus on three questions about the each of the five selected participation points: 1) Whether there is a strong correlation between the participation point and the project performance; 2) Is the influence positive or negative; 3) Why. The conclusion is organized as follows.

Participation point 1: Contract agreement in the form of business. Model shows a negative influence (Coefficient of - 0.3448). All of the 9 interviewee consider it is strongly relevant to project performance. Four of them think its influence is negative, because it tends to make the work falls into unnecessary details. However, other five interviewee consider that its influence can be positive if the degree of participation is proper. The model is not fully compatible with the interview, and the reason need to be further studied in a bigger data set.

Participation point 2: Project planning in the form of management. Model shows a negative influence (Coefficient of - 0.0977) and is fully compatible with the interview. All interviewee agree that it has negative influence because the acquirer often tends to make a unreasonable short time plan thus reducing the software quality.

Participation point 3: Software requirements analysis in the form of business. Model shows a positive influence (Coefficient of 0.0249) and is fully compatible with the interview. All interviewee consider it has positive influence, because commonly, the acquirer has better understanding of

the business knowledge, and its participation helps to increase the working efficiency.

Participation point 4: Data migration regular meeting in the form of technique. Model shows a positive influence (Coefficient of 0.2344), which is consistent with 7 interviewee's opinions. Because they think that data migration has high requirements on system reliability, data consistency and involving interaction with legacy systems, and acquirer can contribute their valuable knowledge of the data and legacy systems. Other two interviewees didn't think this participation point has strong correlation with the project performance. Through deeper investigation, we found these 2 interviewees do not have experience of large-volume data migration, and their point of view is only according to personal intuitions.

Participation point 5: Testing problem resolution in the form of management. Model shows a negative influence (Coefficient of -0. 0191), which is opposite to all interviewees' opinion. All interviewees hold the point that acquirer's participation helps to avoid unnecessary dispute and has positive impact on project performance. Through further data analysis, we find the related data of this participation point satisfies uniform distribution conditions and does not have obvious offsets. So the opposition between model and interviewee's opinions needs to be treated with caution.

VI. CONCLUSION

It is a challenging task to make acquirer participation plans. This paper proposes a method based on objective historical data, and establish a quantitative relationship model between the acquirer participation and project performance, which can provide guidance for making acquirer's participation plans. This method is approved to be valid in terms of mathematic properties study even in small sample size. The interview with project managers shows that the most conclusions of our model are consistent with their experience. But there are also some opposite cases, which deserve further study. Future work of this paper is to improve accuracy and trustworthiness of the model, by collecting more project data and combining with other variable-selection methods.

ACKNOWLEDGMENT

This work is supported by the Key National Science & Technology Specific Projects under Grant NO. 2011ZX01043-001-002, and High-Tech Research and Development Program of China under Grant No. 2013AA01A605.

REFERENCES:

[1] Zhang KL. Ananlysis on Current Status and Policy Suggestion of E-government Outsourcing in China, China Management Informationization, 2010, 13(21):55-57 (in Chinese with English abstract).

[2] Jun H, William K.The Role of User Participation in Information Systems Development: Implications from a Meta-Analysis. Journal of Management Information System, 2008, (3):301-331

[3] BachoreZ, ZhouL. A Critical Review of the Role of User Participation in IS Success, AMCIS 2009 Proceedings, Paper 659.http://aisel.aisnet.org/amcis2009/659.

[4] SubramanyamR, WeissteinF, KrishnanM. User Participation in Software Development Projects. Communications of the ACM, 2010, (3):137-141.

[5] McleodL, MacdonellS, DoolinB. User participation in contemporary IS development: an IS management perspective, Australasian Journal of Information Systems,2007, (1):113-136.

[6] BussenW, Myers M. Executive information systems failure: A New Zealand case study. Journal of Information Technology, 1997, (12):145–153.

[7] GarceauL, JancuraE, KneissJ. Object oriented analysis and design: A new approach to systems development. Journal of Systems Management, 1993, 44(3):25-33.

[8] McKeenJ, GuimaraesT, WetherbeJ. The relationship between user participation and user satisfaction: An investigation of four contingency factors. MIS Quarterly,1994, 18(4):427-451.

[9] HeinbokelT, SonnentagS, FreseM, StolteW, BrodbeckF. Don't underestimate the problems of user centredness in software development projects - there are many!, Behaviour and Information Technology, 1996, 15(4):226-236.

[10] The Institute of Electrical and Electronics Engineers.IEEE Std 12207:2008. Systems and software engineering-Software life cycle processes Information technology-Software life cycle processes. The Institute of Electrical and Electronics Engineers, 2008.http://ieeexplore.ieee.org/servlet/opac?punumber=447582

[11] Tibshirani R, Hastie T, Narasimhan B, Chu G. Class prediction by nearest shrunken centroids, with applications to DNA microarrays. Statistical Science, 2003,18(1):104-117.

[12] Fan JQ, Ren Y. Statistical Analysis of DNA Microarray Data in Cancer Research. Clinical Cancer Research, 2006,12(15):4469-4473.

[13] Emmanuel C, Terence T. The Dantzig selector: statistical estimation when p is much larger than n. The Annals of Statistics, 2007,35(6):2313-2351.

[14] Kasunic M. Performance Benchmarking Consortium, NDIA CMMI Technology and Users Conference. Denver, CO, 2006.

Synchronized Data Acquisition from Web Services Serving at Disparate Rates

D. R. Plante
Department of Mathematics and Computer Science
Stetson University
421 N. Woodland Boulevard
DeLand, FL 32723, USA
Email: dplante@stetson.edu

***Abstract*—Service-oriented architectures (SOAs) have provided improved means of serving data in business-to-business (B2B) communications. Usually, however, the flow of data from services is limited by the service provider, and different providers may limit to vastly different rates, leading to different quality of services (QoSs) for each. In some cases, the data from these various providers must be synchronized by the service consumer, for example, when that data comes from different marketplaces (e.g. Amazon or Ebay) and provides such information as prices, item sales ranks, and quantities. The acquisition of that data from each service must be reasonably synchronized if that data is to be reliably compared. This paper addresses the problems associated with this synchronization when QoS differs greatly between services but the quality of the data and the services themselves vary as well.**

***Index Terms*—Web services, quality of services.**

I. Introduction

A service-oriented architecture (SOA) [1] provides a loosely coupled architectural style that facilitates the exchange of data between software agents. In business-to-business (B2B) communications, service providers and service consumers exchange data conforming to specifications determined by the provider. In exchanging this data, the services of different providers may offer vastly different levels of quality of service (QoS). While network lag is one limiter of service, some services are simply not able to provide data at a level that other providers do. It is also the case that some providers purposefully limit or *throttle* consumer requests at specified rates to prevent abuse of the service and server overload. Both Amazon [2] and eBay [3] provide detailed and specific *throttling* or *call limits* on accessing data from their services. Large sellers on their respective marketplaces may receive increased limits commensurate with their increased sales. However, sales on international marketplaces may differ greatly (for example AmazonCA or AmazonUK), again with QoS for these services differing greatly.

As will be explained in Section II, if all data from all services are of equal importance, and if all services are also of equal importance, the rate of the slowest service produces a hard limit to the overall rate that synchronized data may be obtained from all services. Of course, data may be obtained more rapidly from higher QoS services. The problem, however, involves synchronizing each related data unit from each service with that from the other services.

As a concrete example of the problem at hand, assume a particular model camera is selling on eBayUS, eBayUK, AmazonUS, and AmazonUK and the service consumer wishes to obtain the lowest selling price of the camera from any seller on any of those marketplaces. As marketplaces and item prices are often highly dynamic, reliable price comparison requires requesting a *data unit* (e.g. lowest price of camera for given UPC or model number) from each of the four marketplaces at reasonably synchronized times. Note that the term synchronization here is somewhat relaxed, as the exact time of data acquisition is not required. However, it is important that the data is obtained at reasonably similar times. *Reasonably* here depends a great deal on the data unit itself, as some item prices on these marketplaces will hardly change at all for days or weeks, while some will change often, possibly in the order of seconds or minutes.

Given the dynamic nature of such marketplaces and QoS limits, it is impossible for any service consumer with large data unit needs to be in relative synchronization with the service provider. For a seller with millions of items for sale in its inventory, the problem can be quite severe. If such a seller wishes to list an item on the marketplace that provides the greatest profit at a given time, knowing other sellers' prices for items on those marketplaces is critical. Stale price data may lead to reduced profits or worse, to lost sales for not being competitively priced.

In Section II, we briefly provide some background and some related work, though we have found no other comparable work on this problem in the literature. Also in Section II, we describe the problem in more detail. We then describe our approach to a reasonable solution of the problem in Section III. In Section IV we demonstrate the implementation of algorithms developed in this paper. In Section V, we summarize the paper along with some future directions for work on this problem.

II. Background and Problem Description

Online marketplace auctions have been the subject of research for over a decade [4], [5] and continue to be actively studied. Most of this work is aimed at trying to understand underlying consumer behavior and/or price trends, unlike the

present work where we wish to simply have accurate and timely data from multiple services providing data at disparate rates. Also, while previous research has attempted to understand underlying web services, contracts they satisfy, and the QoS guarantees [6], [7], [8], this study focuses on the added constraint that the data obtained from the provider must be reasonably synchronized and obtained from all the services, not only some.

A. Disparate Service Rates and Quality

To illustrate the problem, assume 4 service providers hold 8 instances each of related data, $A, B, C, ..., H$ as shown in Figure 1. For example, data unit A may be the price of a particular model camera, B may be the price of a DVD, etc. Importantly, for all four services, A refers to the same item. While the price may differ, the price is tied to the particular item such that prices for that item may be compared across marketplaces. Assume the QoS of each service permits them

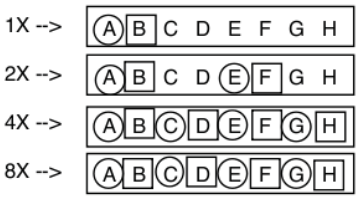


Fig. 1. Four services being consumed at rates that are powers of 2.

to serve data to the provider at a rate of 1, 2, 4, and 8 times (i.e. $1\times$, $2\times$, $4\times$, and $8\times$) the speed of the slowest service, which in the figure is the first or top service. As the fastest service provides data at 8 times the speed of the slowest, by the time the first is able to serve its second unit of data (i.e. data unit B), the fastest service will serve all 8 data units. The next slowest service will, in the same time frame, serve 4 units, and the next slowest will serve 2 units. One key constraint, however, is that to be useful for comparison, the data must be obtain at reasonably similar times. For example, if the service operating at $2\times$ speed is to first serve data units A and then H and the $1\times$ service serves unit A through H in that order, by the time the $1\times$ service gets to unit H, the corresponding data from service $2\times$ will be stale and possibly inaccurate. Of course, since the $2\times$ service is twice as fast as the $1\times$ service, it may serve the H data again to try and synchronize with the $1\times$ service when that service provides the unit H.

One important point to note here is that no matter how fast the other services are, while they may offer all of the data being provided by the service operating at $1\times$ speed, ultimately, the $1\times$ service is a bottleneck that cannot be overcome when the constraint of synchronization is imposed. Of course, the relaxation of this constraint will help. If, for example, the $2\times$ service serves data units A and B in the time the $1\times$ service serves only A, if the $1\times$ service then serves B to the consumer, the consumer may deem the data unit B from the $2\times$ service to be "recent enough" to use when it receives the related data from the $1\times$ service. By categorizing the data units based on some *quality* criteria (e.g. the sales rank of the item on the marketplace or the user feedback on that item) and *rate of change* of that data (e.g. how quickly the price of the item is expected to change), intelligent choices may be made on how often data should be updated.

In this paper, we will not address the issue of how delayed data may be related. Instead, we assume the service consumer requests data from the number of services and then compares it. One constraint that is relaxed, however, is that of service quality. In Figure 1, assume that the $1\times$ service is slow partly because it is a "lower quality" service. For example, it may be that it is a marketplace with fewer sales (e.g. eBayUK having fewer sales per day than AmazonUS). Possibly fewer resources are dedicated to the slower service because it is less active. In any case, it may be that the service consumer deems the *quality* of the $1\times$ server to be lower than that of the $2\times$ server. Note that the $2\times$, $4\times$, and $8\times$ services can all serve data units A and E in the time it takes the $1\times$ service to serve only A. The data units A and E are now synchronized between all but the $1\times$ service. Likewise, the $4\times$ and $8\times$ services both can serve the units A, C, E, and G in the time it takes the $1\times$ service to serve only A and the $2\times$ service to serve only A and E. The *circled* data units in the Figure 1 indicate these elements. The $8\times$ service could have all 8 data units circled, but the other 4 units would not be synchronized with any other service. (Obtaining that data for analytics purposes would be reasonably, of course.) This may be considered an acceptable solution in some cases, namely to allow services $4\times$ and $8\times$ to keep the 4 data units A, C, E, and G synchronized while the $2\times$ service keeps A and E synchronized with the faster services and the $1\times$ service only keeping the unit A synchronized.

In Figure 1, the *squared* data units show the same scenario for the next data requests, namely when the $1\times$ service requests element B. We see that in the time it takes the $1\times$ service to obtain two data units, the 4x and $8\times$ services have obtained all 8 data elements in relative synchronization, and the $2\times$ service has obtained half the data units in synchronization with the two faster services. Here it is clear that the orders of speed are important, since having respective speeds of, for example, $1\times$, $1.7\times$, $2.3\times$, and $7.4\times$ would not permit such synchronization. Data units are acquired more rapidly from the faster services, but keeping them aligned at these rates is problematic. However, using other integer powers (e.g. $1\times$, $3\times$, $9\times$, $27\times$) will also allow for synchronization, especially in cases where the disparate QoS of services are even more dramatic. Therefore, when relaxing the constraint of *all* services having *all* data units synchronized, the above method of acquiring data at rates that are integer powers of an integer base is useful.

B. Variable Data Quality

Not all data units served by a service are of equal quality. Using our example of online marketplaces, some items are top sellers, for example a new release DVD of an Oscar

winning movie or a New York Times top selling book. Also, some may even have metrics to indicate their importance on a marketplace (e.g. Amazon Sales Ranks or buyer feedback ratings). As already described, while the slowest service being accessed by a service consumer provides a bottleneck when comparing synchronized data, some improvements to the most restrictive model are possible. By taking into account data quality, another alternative means of acquiring synchronized data is possible.

Assume a single service with the eight units of data from the previous section, A through H. If units A, B, and C are fast-changing and high quality (e.g. hot selling movie DVDs), by cycling through the data in sequence, each unit will only be requested and updated once each in one cycle, the same amount as the other data units. If, however, units G and H are very slow-moving inventory items (e.g. old B-rated movie DVDs) whose prices are fairly unchanging, we may be willing to obtain that data fewer than one time per eight requests so that we may obtain the higher-quality data more often.

In Figure 2, we provide data quality measures to each data unit. From this data, we see that A is considered $4\times$ as important as H. We see that rather than obtain each of the eight data elements in one cycle, we can use this metric to request A four times as often as H. The total number of requests for units A through H is 16. So while A will be requested $4\times$ in the 16 requests, H will only be requested once. However, given the importance of A, this may be a reasonable choice. For the proposed quality measures, a single cycle is now 16 requests rather than the original 8, so the expansion factor for one pass is 16/8 = 2.

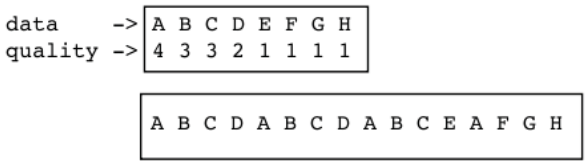


Fig. 2. Data of different quality/importance being served more often.

III. Proposed Methodology

As Figure 3 shows, N service providers serve data units to one service consumer at the request of the consumer. Each provider is considered to have a service quality given by

$$\mathcal{S} = \{s_1, s_2, ..., s_N\}. \tag{1}$$

Service quality here refers to the value placed on the given service and the collective importance of its data, not quality of an individual data unit. The rate at which each service returns data is given by its velocity,

$$\mathcal{V} = \{v_i : i \in \{1, ..., |\mathcal{S}|\}\}. \tag{2}$$

The velocity is a real-valued measure of the number of times faster each service is than the slowest service. By definition, the slowest service will have a velocity of 1.0 while other

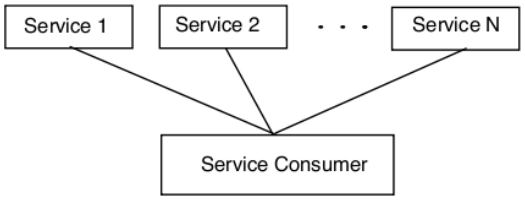


Fig. 3. Service consumer and producers of data units.

services will have velocities greater than or equal to 1. As detailed in Section I, since synchronization of data acquired from the different services is key, velocity in the form provided by $\mathcal{V}$ is not as useful a measure as is the *order* of the service. The order of the services is defined as

$$\mathcal{O} = \{o_i : i \in \{1, ..., |\mathcal{S}|\}\} \tag{3}$$

where for base b,

$$o_i = \{b^j : j \in \mathbb{N}^0, v_i < b^{j+1}, v_i \geq b^j\} \tag{4}$$

In general, a base of 2 provides the greatest flexibility in optimizing synchronization and speed of services, though higher valued bases may be chosen.

Each service is capable of delivering M units of data that are to be compared to related data delivered by the other services, where the data provided by service j is given by

$$\mathcal{D}_j = \{d_{1j}, d_{2j}, ..., d_{Mj}\}. \tag{5}$$

This data may take any form that may be parsed and processed on the receiving end of transmission. For example, data may be flat or comma separated or more structured, such as JSON, or XML. Each unit of data also has a corresponding quality

$$\mathcal{Q}_j = \{q_{1j}, q_{2j}, ..., q_{Mj}\}. \tag{6}$$

We require that each unit d_{ij} be aligned across all services j for each unit i. For example, the first unit of data for service j, namely d_{1j}, should correspond to the same data for all services. An example of this may be the price for a given item with a given UPC or ISBN. While the price of that item may differ between different services, the data should be for the identical item (i.e. UPC or ISBN) across services. Likewise, the quality of that unit of data, in this case q_{1j}, should also align with the data for the given service. Using the example of an item with UPC or ISBN, quality is numeric and may be the item's sales rank on a marketplace or user feedback value for that item. As their values are clearly tied to each other, the data and its quality are combined to produce pairs defined by

$$\mathcal{P} = \{(d_{ij}, q_{ij}) : i \in \{1, ..., |\mathcal{D}|\}, j \in \{1, ..., |\mathcal{S}|\}\} \tag{7}$$

We use the differences in qualities to request data from services in a manner that allows improved performance for higher quality data and services at the expense of lower quality data and services. To do this, we first require that qualities are normalized, namely

$$\hat{q}_{ij} = \frac{q_{ij}}{\sum_{k=1}^{N} q_{kj}} \tag{8}$$

$$\hat{s}_i = \frac{s_i}{\sum_{j=1}^{N} s_j} \tag{9}$$

If all services are deemed to be of comparable quality, the normalized quality of each service is $\hat{s}_j = 1/|S|$ for $j \in 1, ..., |S|$. While data quality thus far has referred to data from each different service, for decision making on which data to request more often, an overall quality metric is needed for each data unit collectively across services. This is achieved by weighting normalized data quality by normalized service quality, with

$$\hat{Q}_i = \sum_{j=1}^{N} \hat{q}_{ij}\hat{s}_j \tag{10}$$

$$\sum_{i=1}^{M} \hat{Q}_i = 1 \tag{11}$$

Clearly for collections of data where M is large, each value of $\hat{Q}_i$ will be necessarily quite small. A mapping is desired that will provide a relative multiple of updates per data unit in one complete pass through all data from each service. One way to accomplish this is to map the distribution of $\hat{Q}_i$ to the interval $[0, 1]$ using a parameterized function and to scale this mapping. Assuming $\hat{Q}_{min} = \min \hat{Q}$ and $\hat{Q}_{max} = \max \hat{Q}$, we have

$$f(x) = D[\hat{Q}_{min}, \hat{Q}_{max}](x) \to [0, 1] \tag{12}$$

$$c_i = \lfloor (C_{max} - 1) f(\hat{Q}_i) \rfloor + 1 \tag{13}$$

$$C = \sum_{i=1}^{M} c_i. \tag{14}$$

Here, C_{max} is chosen to be the maximum number of times any data item may be requested during a full pass through all data from all services. For a uniformly distributed set $\hat{Q}$, we note that $D[\alpha, \beta](x) = U[\alpha, \beta](x)$. The number of data units requested for one complete pass through $\mathcal{D}$ from all services will be expanded by a factor of $T = C/M$.

One sample mapping is to use a simple linear function,

$$D[\hat{Q}_{min}, \hat{Q}_{max}](x) = \frac{x - \hat{Q}_{min}}{\hat{Q}_{max} - \hat{Q}_{min}}. \tag{15}$$

Such a mapping may not be desirable, however, if data of higher quality collectively is far preferred over data of much lower quality. In such cases, a mapping such as a logistic function may be preferred, for example

$$D[\hat{Q}_{min}, \hat{Q}_{max}](x) = \frac{1}{1 + e^{-\left[\frac{x - \hat{Q}_{min}}{\hat{Q}_{max} - \hat{Q}_{min}}\right]}}. \tag{16}$$

Since for each data unit d_{ij} it is required that index i refers to the same data description (e.g. d_{1j} is the same book for each service, with only its variable properties, such as price, differing for each service j), we define the *request description* to be r_i. Therefore, a request for data r_1 will result in data unit d_{11} being provided by service 1, d_{12} being provided by service 2, and so on. The request units and their pairings with their respective counts are given by

$$\mathcal{R} = \{r_i : i \in \{1, ..., |\mathcal{D}|\}\} \tag{17}$$

$$\mathcal{C} = \{(r_i, c_i) : i \in \{1, ..., |\mathcal{D}|\}\} \tag{18}$$

Algorithm 1: Algorithm to create ordered list of request units.

Data: set $\mathcal{C}$; number of data units M; max count for a request C_{max}; total count C
Result: $requestList$ ordered list of requests for services

```
requestMultiplicities[] ← OrderRequests(C)
numBins ← C_max
binSize ← ⌊C / numBins⌋
bins ← string[][numBins]
remainderBin ← string[]
requestList ← string[C]
for i ← 1 to M do
    request ← requestMultiplicities[i].r
    count ← requestMultiplicities[i].c
    jump ← ⌊numBins / count⌋
    startBinNum ← 1
    while startBinNum ≤ numBins and
    size(bins[][startBinNum]) ≥ binSize do
        startBinNum ← startBinNum + 1
    end
    binNum ← startBinNum
    lastBin ← startBinNum + (count − 1) * jump
    if lastBin > numBins then
        jump = 1;
    end
    for j ← 1 to count do
        if binNum ≤ numBins then
            place copy of request in bins[][binNum]
        else
            place copy of request in remainderBin[]
        end
        binNum ← binNum + jump
    end
end
for k ← 1 to numBins do
    foreach bins[][k] as request do
        requestList[] ← request
    end
end
foreach remainderBin[] as request do
    requestList[] ← request
end
return requestList
```

A. Constructing Request List of Multiples

In Section II-B, we gave a simple example of using the quality of data to construct a new list of data unit requests.

In Algorithm 1, we provide one approach to distributing the multiple quantities of data in $\mathcal{C}$ based on each element's respective count, c_i. The method described here assumes that given the sizes defined by $l = \lfloor \max \mathcal{O} \rfloor$ and $n = \lfloor \frac{C}{\max \mathcal{C}} \rfloor$, we require that $n >> l$. Physically, n is the number of request units in each of the $\max \mathcal{C}$ bins that data will be placed into and l is the maximum factor of velocity (e.g. 2, 4, 8, etc.) indicating how much faster the fastest service is than the slowest service. Therefore, the required condition will always be met assuming there is a substantial amount of data that is required from the services.

The algorithm works as follows. We first call `OrderRequests(`$\mathcal{C}$`)` so that the data/count pairs in $\mathcal{C}$ are in decreasing order based on the value of c_i so that the first pair has the highest count and the last pair has the lowest count. Since multiple pairs will have the same count, the order of the multiples does not matter as long as they all are grouped together. We then create $numBins$ bins and one overflow bin, $remainderBin$. Since the number of bins is equal to the maximum multiplicity of request units, we ensure that multiple copies of requests are dispersed across the request list to allow price updates to be separated temporally. For each request-count pair (r_i, c_i), a $jump$ factor is calculated to allow maximal dispersal of higher-priority requests, indicating how many bins to jump before placing the next copy of the request in the list. An example demonstrating the algorithm is provided in Section IV.

B. Traversing Request List of Multiples

For Algorithm 2, the $requestList$ constructed by Algorithm 1 is used to construct each service's request group. An example of this is illustrated in Figure 1 for a simplified case, where the circled elements form the request group for the services in the first step, and the squared elements are those requested in the second step. This process is stepped along until the slowest service traverses the entire $requestList$ once. The $jumps$ being made for each service indicates how many elements of $requestList$ to step over using the Algorithm 2. Conditionals in the inner loops work to restrict the number of elements being grouped based on speed of service and running beyond the length of $requestList$.

Part of the request synchronization takes place here at the end of the algorithm, where the requests are made to the services. All requests are made by the consumer, and only once all results are returned by the service providers does the algorithm step along to the next set of requests. Service failure is not addressed and is somewhat application dependent, as business logic must be used to decide how best to handle the problem. It should be noted that one advantage to the approach developed here is that since service velocities are always dropped to match either 2^i, 3^i, or other elements of a power series, some flexibility is already built into the model to handle some network lag or delays of these faster services. When the slowest service suffers from some (possibly serious) delays, synchronization of all services becomes problematic in any case as that service already is the bottleneck.

Algorithm 2: Algorithm for traversing request list.

Data: set $\mathcal{O}$; request list data $requestList$; number of data units M; number of services N

```
maxOrder ← max O
for i ← 1 to N do
    step[i] ← maxOrder / o_i
end
for j ← 1 to M do
    for i ← 1 to N do
        jumps ← int[o_i]
        k = 1
        repeat
            jump ← (k − 1) ∗ step[i];
            if jump < maxOrder then
                jumps[k] ← jump
                k ← k + 1
            end
        until jump ≥ maxOrder;
        for k ← 1 to size(jumps) do
            index ← jumps[k] + j
            if index ≤ M then
                requests[k][i] ← requestList[index]
            end
        end
    end
    request requests[][] data from services
end
```

IV. Example

While in general, the number of data units M in real-world applications will be very large, we illustrate the implementation of the two algorithms from Section III with a small dataset to understand its essence. In Table I, we simplify the structure of the requests using the notation of `letter-number`, where `letter` refers to a group of requests having a particular count (e.g. `A` has a count of 6 copies in the final request list), and `number` refers to each instance of a unique request (e.g. `A4` is the fourth unique request that has a count of 6.) In Figure 4, we show how the requests from Table I are loaded into the final request list by binning that data. As the maximum count of any unique request in this example is 6, Algorithm 1 begins by creating 6 bins plus an overflow bin of undetermined sizes.

TABLE I
Request-count pairs (r_i, c_i) with $M = 30$ unique requests, $C = 105$ total list entries and expansion factor of $T = 3.5$.

r	c	Number of Unique Requests	Number of Copies In Request List
A1 → A10	6	10	60
B1 → B5	4	5	20
C1 → C5	3	5	15
D1 → D10	1	10	10

The algorithm places the multiple instances of a unique request unit into bins by *jumping* an appropriate number of bins. In the present case, for the counts of 6, 4, 3, and 1, the jumps are $\lfloor \frac{6}{6} \rfloor = 1$, $\lfloor \frac{6}{4} \rfloor = 1$, $\lfloor \frac{6}{3} \rfloor = 2$, $\lfloor \frac{6}{1} \rfloor = 6$. Therefore, `A` units will jump 1 bin, `B` units will jump 1 bin, `C` units will jump 2 bins, and `D` units will jump 6 bins. As we see in Figure 4, `A` units are placed in bins 1 through 6, and `B` units are placed in bins 1 through 4. However, `C` units are placed 2 bins apart if possible, and once units `C1` and `C2` are placed in bins 1 and 3, those bins have $\lfloor \frac{105}{6} \rfloor = 17$ elements each and are considered full. Therefore, the algorithm steps to the next available bin, bin 2, and places units `C3` and `C4` in bins 2 and 4, which are spaced by 2 bins. At this point, bins 1 through 4 are filled, leaving only bins 5, 6 and the overflow bin available. Since jumping 2 bins from the next available bin, bin 5, would move beyond the last of the 6 bins, the jump is changed to 1, and `C5` is placed in bin 5 then 6. Now, since the last `C5` unit cannot be placed in one of the 6 primary bins, it is placed in the overflow bin. The 10 `D` units are placed in bins 5, 6, then the overflow bin one at a time as bins fill up. These bins are then sequentially loaded into the array $requestList$ and returned for Algorithm 2 to request the data from services. This method of placing units gives maximum priority to the highest-ranked data units, with lower-ranked units being placed less optimally if required, as is the case for `C5` since its 3 copies are not as well spaced across bins.

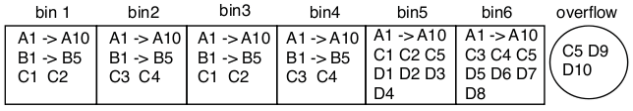


Fig. 4. Bin loading.

With the placement of requests in $requestList$, the first 17 elements (i.e. those originating from bin 1) are given in Figure 5. If we assume three services operating at speeds (i.e. orders) of 1, 2, and 8, the respective $jumps$ in Algorithm 2 are 8, 4, and 1 respectively and specified in the figure next to each service. Circled units are those that would be grouped for each service in the first iteration of the loop through the M unique request elements, while the squared elements show which units are grouped during the ninth step through the loop. An important point to note here is that all units in each group are and will always be unique as long as the order of speed is much smaller than the number of data units.

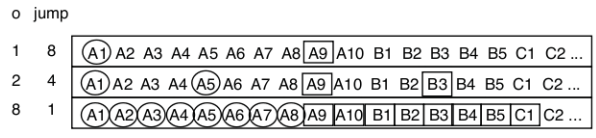


Fig. 5. Bin traversal.

V. Conclusions and Future Work

In this paper, the problem of a service consumer acquiring data from a collection of service providers operating at disparate rates in a synchronized manner is addressed. We provide one possible solution to this problem by using service and data quality to modify how the data is requested. In this initial approach, variable network lag and service provider failure is not addressed. In future work, the issue of variable network lag for the various services should be addressed, though as already pointed out, some flexibility exists in the present approach to account for some lag. Should serious service quality degradation or inversion take place, for example if the slowest service actually becomes faster than other services due to lag or failure of the other services, the present approach will perform less optimally. The issue of service failure is a more difficult problem to address as how to handle this is somewhat business-model oriented. However, for certain business decisions such as *ignore any service that is down*, improvements to the present approach are possible, and such effects will be addressed in future work. Also, the present model assumes that all services have all data units available to provide to the service consumer, though this is often not the case in real service applications and is also the subject of future work. Finally, decoupling of the service provider's synchronized request structure into separate consumer request modules or services may be preferred, with each module requesting data from the providers separately as fast as each can provide it and storing it for use by a data-synchronizing aggregator.

References

[1] Thomas Erl. Service-oriented architecture. Prentice Hall, 2004.

[2] Amazon Marketplace Web Service (Amazon MWS) Developer Guide 2012, https://images-na.ssl-images-amazon.com/images/G/01/mwsportal/doc/en_US/bde/MWSDeveloperGuide._V386854335_.pdf

[3] Trading API Guide, Verson 811, *ebay developers program*, 2013. http://developer.ebay.com/Devzone/XML/docs/PDF/eBayXMLAPIGuide.pdf

[4] Jungpil Hahn. "The dynamics of mass online marketplaces: a case study of an online auction," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 317-324. ACM, 2001.

[5] Rayid Ghani. "Price prediction and insurance for online auctions," in *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pp. 411-418. ACM, 2005.

[6] Fu, Xiang, Tevfik Bultan, and Jianwen Su. "Wsat: A tool for formal analysis of web services," in *Computer Aided Verification*, pp. 394-395. Springer Berlin/Heidelberg, 2004.

[7] Abdelzaher, Tarek, Ying Lu, Ronghua Zhang, and Dan Henriksson. "Practical application of control theory to web services." In American Control Conference, 2004. Proceedings of the 2004, vol. 3, pp. 1992-1997. IEEE, 2004.

[8] Wang, Xiaoling, Kun Yue, Joshua Zhexue Huang, and Aoying Zhou. "Service selection in dynamic demand-driven Web services," in *Web Services, 2004. Proceedings. IEEE International Conference on Web Services*, pp. 376-383. IEEE, 2004.

A Dialogue Game Approach to Collaborative Risk Management

Fabrício S. Severo, Lisandra M. Fontoura, Luís A. L. Silva
Programa de Pós Graduação em Informática
Universidade Federal de Santa Maria (UFSM)
Santa Maria, Brazil
{severo.fabricio, lisandramf, silva.luisalvaro}@gmail.com

***Abstract*—Risk management aims to reduce surprises and increase the chances of success of a project. Although collaborative tasks of risk management have a crucial role in the capture and reuse of different experiences in risk management, limited attention has been dedicated on the argumentative interaction steps among project participants engaged on the development of risk management debates. In this paper, we describe an argumentation-based collaborative approach to risk management. Grounded on the notion of dialogue games, this approach presents a new risk management communication protocol defined by its locution acts and interaction rules. Along with this protocol, a web-based collaborative Risk Discussion system is offered. Experiments were developed to evaluate both the approach and the system. Results from these collaborative risk discussion experiments show positive evidence for the applicability of the protocol and the usability of the system.**

Keywords- Risk Management, Collaboration, Dialogue Games.

I. INTRODUCTION

Risk management is based on the identification of key risks in a project and the elaboration of plans to prevent or mitigate events that could interfere on the project or organization goals. In essence [1], innovative projects are likely to have high risks but also high benefits. According to [1][2], risk management tasks ought to be developed collaboratively and with reuse of information from previous projects. Collaboration activities ensure that experiences are exploited in a project, taking into consideration different perspectives about the risks and their plans. In addition, the reuse of risk information captured on these collaborative tasks allows stakeholders to avoid repeating plans that have failed in the past.

In this paper, we present a novel argumentation-based approach to collaborative risk management. Due to the dialectical nature of collaborative risk management debates, this argumentation approach for risk management is grounded on the notion of "dialogue games" [3]: knowledge representation structures which aim to identify and represent meaningful steps of human interaction that are typical of debates. Our approach exploits the knowledge elicitation and representation of key risk information in a project and the consequent organization of a risk discussion memory. We also present a web-based Risk Discussion (RD) system, which offers a collaborative environment where stakeholders can interact and record those risk discussions in the memory, allowing future reuse of that information. Experiments were developed to evaluate our approach and system, in which results show positive evidence for its applicability and usability.

In this paper, Section 2 presents background information; Section 3 presents our dialogue game-based risk management approach; Section 4 presents experimental results; finally, Section 5 presents conclusions and future works.

II. BACKGROUND

Boehm [2] described risk management as a set of risk principles and practices directed to the identification, analysis and treatment of risk factors. Among other goals, these tasks aim to increase the likelihood of success of a project. Risk management consists of: the identification of possible future problems and their causes; the analysis of those risks and their prioritization; the elaboration of plans to deal with the risks; and the monitoring and execution of plans, when necessary in the project life cycle.

Collaborative tasks of risk management can be modeled as a process of "argumentation" as one can observe when typical argumentation systems are considered (e.g. [4][5]). Argumentation studies the structure of arguments and the process of arguing [6]. The main characteristic of arguments is the fact that an argument presents (in its simplest format) a statement that may or may not be true in a point in time. Arguing (or argumentation) is the act of using arguments to explain or justify a point of view. In argumentation, dialogues can be expressed by means of "dialogue games". According to [3], the main structures defining a dialogue game are: (1) start and finish rules – describing when and how a dialogue should start and end; (2) locution acts – defining the types of actions for participant interaction; and (3) combination rules – describing the context where each locution is exploited so that these locutions could organize the progress of the dialogue. Protocols like these are often implemented by means of logic-based formalisms instead of semi-formal representation models directed to support the development of broader tasks of knowledge engineering in less formal application problems such as risk management.

III. A DIALOGUE GAME APPROACH TO RISK MANAGEMENT

This paper discusses a new approach to engage projects stakeholders on the development of risk management debates. This approach is grounded on a well-defined communication protocol, tailored for answering risk management needs. As a result of collaborative risk management tasks, information exchanged by participants involved on this kind of structured

dialogue is recorded in a risk management memory. Further details of this approach are described in the following sections.

A. Definitions

In our argumentation-based interaction protocol, and corresponding system that implements this protocol, a risk management discussion memory M consists of a set of projects $PR = (pr_1, pr_2, \ldots, pr_n)$. Each project pr_i is a tuple (C, D) where C is a typical set of factual project characteristics $(c_1, c_2, \ldots, c_n)$ – a list of attributes and values – and D is a set of risk management discussions $(d_1, d_2, \ldots, d_n)$. Each discussion d_i is formed by a set of speech acts $A = (a_1, a_2, \ldots, a_n)$. Importantly, these locution moves a_i represent different arguments advanced in the discussion by a participant $p_i \in P$, where P is a set of discussion participants $(p_1, p_2, \ldots, p_n)$ involved in the dialogue. In this knowledge elicitation model, S is a set of statements $(s_1, s_2, \ldots, s_n)$ where the content of these statements is presented in a textual format by participants. In summary, the dialogue game protocol for risk management $£_{RM}$ is defined as a tuple (L, R) where L is a set of predefined locution acts $(l_1, l_2, \ldots, l_n)$ and R is a set of combination rules $(r_1, r_2, \ldots, r_n)$. In this case, a rule r_i is represented as a tuple (l_i, l_j) meaning that l_j can be used as a response to l_i during a debate. In essence, the main role of the dialogue game protocol $£_{RM}$ is to organize the collaborative management of risks and its consequent recording of risk discussions.

B. The Locution Acts of the Dialogue Game

We developed a new set of locution acts L for risk management. This protocol is based on the integration of locutions for deliberation dialogues [3] with locutions for typical tasks of risk management. The L set consists of the locutions l_i:

start_discussion(d_i, p_i): it is used to start a discussion d_i by a participant p_i;

propose(t, a_i, s_i, p_i): it allows making a proposition of type t, where $t \in T$ = {risk, impact, probability, plan, consequence}, in the dialogue. This proposition involves a speech act a_i and contains a statement s_i which is advanced by a participant p_i;

select(t, a_i, p_i): it allows making a selection of type t, where $t \in T$ = {risk, impact, probability, plan, consequence}. This selection involves a speech act a_i which is advanced by a participant p_i;

withdraw(a_i, p_i): it is used to state that a speech act a_i is rejected by a participant p_i;

argument_pro(a_i, s_i, p_i): it is used when a participant p_i wants to state an argument in favor of a speech act a_i of the dialogue. It also describes this argument as a statement s_i;

argument_con(a_i, s_i, p_i): it is used when a participant p_i wants to state an argument against a speech act a_i of the dialogue. It also describes this argument as a statement s_i;

ask(a_i, s_i, p_i): it allows raising a question which is related to a speech act a_i of the dialogue. This question is presented as a statement s_i by a participant p_i;

inform(a_i, s_i, p_i): it is used to advance information related to a speech act a_i in the dialogue. This information is presented as a statement s_i by a participant p_i;

summarize(A', s_i, p_i): it allows summarizing a subset A' of speech acts, where $A' \subset A$, of the dialogue. This summarization is presented as a statement s_i by a participant p_i;

ask_position(a_i, p_i): it is used when a participant p_i needs to request an opinion about a speech act a_i of the dialogue;

opinion(a_i, s_i, p_i): it allows stating an opinion which is related to the speech act a_i of the dialogue. This opinion is presented as a statement s_i by a participant p_i;

end_discussion(d_i, p_i): it is used to close a discussion d_i by a participant p_i.

C. The Combination Rules of the Dialogue Game

The R set of combination rules can be summarized as the following: the *start_discussion* and *end_discussion* locutions are used once in a discussion d_i, opening and closing it, in this order; the *propose* locutions submitted with a type t_i = *risk* are the key elements of a discussion d_i – in this case, the *propose* locutions involving any other type than *risk* are used in response to a *propose* locution where the type is *risk*; the *argument_pro*, *argument_con*, *ask*, *inform* and *summarize* locutions are used in response to any other speech act, promoting the discussion of the topics; the *withdraw* locution can be used after a proposition (i.e. *propose* locutions) to withdrawing from it; the *ask_position* locution can be used after other locutions in order to ask whether participants either agree or disagree with a point of view. In response, participants should use the *opinion* locution to express their opinion.

D. An Example of the Dialogue Game Protocol

In order to understand how our dialogue game mediates a risk management debate, we can present an example of a risk discussion (due to the limits of space, this sample discussion was reduced to a single risk):

a1. **start_discussion**("Risk Management of a Software Project", p_1)

a2. **propose**("risk", a_1, "The schedule and budget are unreal", p_2)

a3. **inform**(a_2, "The project has a fixed budget and needs to be finished in a year", p_1)

a4. **propose**("probability", a_3, "High", p_2)

a5. **ask**(a_2, "Do we know if the project requirements are described properly?", p_2)

a6. **inform**(a_5, "The requirements are okay, but the time to develop this project is not enough.", p_3)

a7. **propose**("impact", a_2, "High", p_1)

a8. **propose**("plan", a_2, "The client should be more present during the project development.", p_2)

a9. **argument-con**(a_8, "This client doesn't have the time for this.", p_1)

a10. **withdraw**(a_8, p_2)

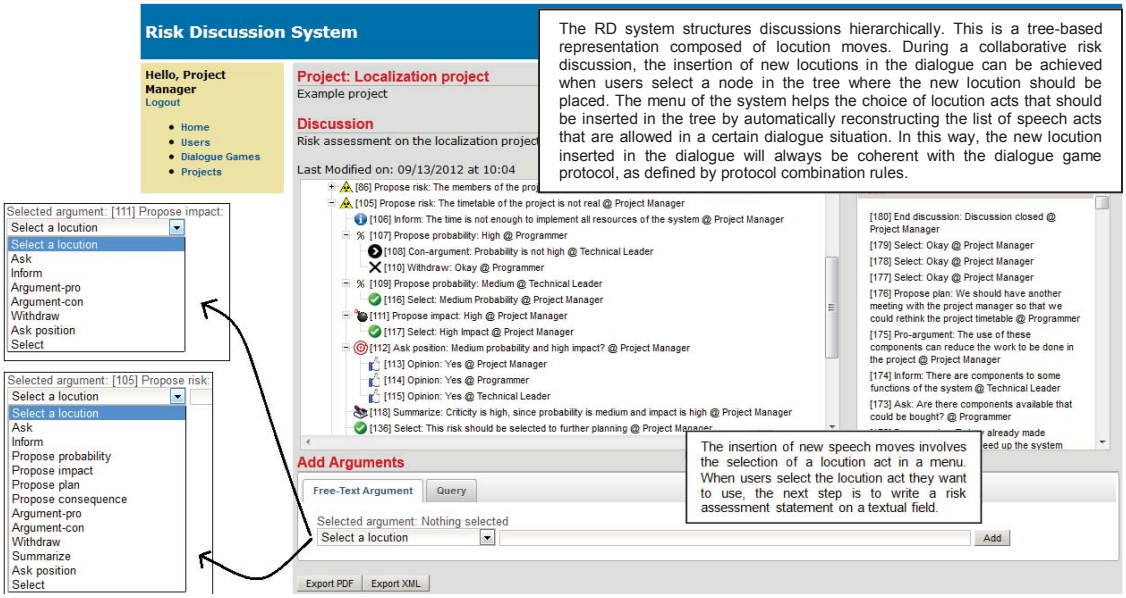


Figure 1. The Risk Discussion system

a11. **propose**("plan", a_2, "We should make periodic meetings in order to discuss problems in the project development and their solutions.", p_3)

a12. **propose**("plan", a_2, "We should improve productivity by changing some coding standards.", p_2)

a13. **summarize**([a_2-a_{12}], "An unrealistic budget and schedule is a risk in our project. This risk will be treated with two plans: the participants will make periodic meetings to discuss the project problems; and the productivity of the programmers will be improved by changing some coding standards.", p_1)

a14. **end_discussion**("Risk Management of a Software Project", p_1)

The key step in this debate is the proposition of risks. In the example, the risk stated was that "*The schedule and budget are unreal*". The probability and impact of each risk are analyzed. Then, plans to deal with the risk are proposed. In practice, multiples risk management plans can be advanced in the debate. In this way, participants are involved in the selection of plans that will be applied in the project. Other locutions can also be exploited during the discussion in order to capture alternative points of view regarding the risks being assessed.

E. A Web-Based Risk Discussion System

The Risk Discussion (RD) system (Fig. 1) supports the development of debates as described by the risk management dialogue game protocol. The representation elements of this protocol are described externally to the system in a XML representation. This is a XML-based representation containing all the locution acts and combination rules defined on the interaction protocol.

Through the RD system, users are able to store their risk discussions in a memory containing concrete experiences of collaborative risk management. The system also contains basic query resources into this memory, where users can search for particular risk management speech acts advanced in past problem situations. In practice, these locutions are the indexes to this memory of risk discussions (e.g. a dialogue participant searching for a list of "proposed plans" in past risk discussions).

In summary, the RD system contains resources for the elicitation and consequent representation of risk information in a structured memory. In practice, we believe these argumentation-based resources can enhance the data that is traditionally managed by other project management systems.

IV. VALIDATION EXPERIMENTS

In order to evaluate the proposed approach and the RD system, two experiments involving software development participants and computer science students were developed. The first experiment aimed to compare our dialogue game-based approach with a "traditional" collaborative risk management approach (based on paper-based templates for the recording of risk management information), where no supporting systems were used. This preliminary experiment involved 16 participants (divided on 3 different groups) on the development of risk management tasks of 2 projects. On the first stage of this experiment, when participants were asked to analyze the risks of the first project, they executed this task without using our argumentation-based system. As a result, they were asked to write a report about their overall risk management achievements. On the second stage, and now analyzing the second project, they were asked to use the RD system to record their risk management discussions. In the end, participants were asked to answer an evaluation questionnaire about their overall risk management experiences. The questionnaire results were not surprising: participants believed it was important to have a system to mediate the risk management discussions and the RD system was able to fulfill most of those discussion needs. Moreover, the RD system environment allowed participants to enhance the discussion coordination from the first stage to the second stage of the experiment. Consequently, the information

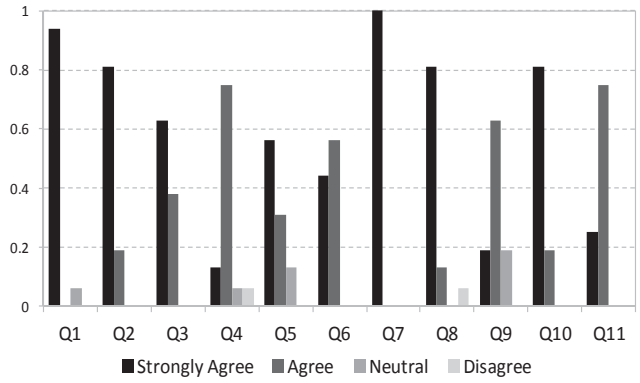


Figure 2. Questionnaire experiment results

recorded by the system was more complete than the risk information recorded on the initial risk management reports.

A second experiment was executed in order to further assess the relevance of our approach and RD system. In total, 16 participants (4 participants were there on the first experiment) were involved on this experiment where 63% claimed good experience in risk management while the remaining participants claimed little experience (although none of them claimed to be an expert in the field). This experiment focused on the utilization of the RD system by 4 different debate groups (contained 4 participants). The evaluation started with a short presentation of general risk management techniques. The dialogue game protocol for risk management was also presented to these participants. All participants were given a document containing basic information about risk management principles, including the main characteristics of a project and its possible risks. Then, the participants started discussing the risks of the project. This debate was fully developed through the RD system.

The overall feedback obtained with this second experiment was promising. Key results are presented in Fig. 2 (the "Strongly Disagree" option was not shown there since none of the participants selected this option). Questions 1 and 2 were designed to assess the relevant of our collaborative approach for risk management. In general, all the participants agreed that it is important to exploit collaboration in risk management (Q1), as well as the recording of risk management debates (Q2). Questions 3 to 6 aimed to assess the applicability of the approach. In this case, all the participants stated that our set of locution acts allowed them to express relevant aspects of risk management (Q3), and they also agreed that the protocol organizes such risk discussions appropriately (Q6). Although some participants disagreed that the effort to learn the protocol is low (Q4), the majority of the participants (88%) agreed that this learning effort is not overwhelming. The majority of the participants stated that the proposed communication protocol is adequate to the development of risk management tasks (Q5), but some of them (13%) selected the neutral questionnaire option related to this question.

Questions 7 to 11 were designed to evaluate the RD system. In this case, all participants agreed that it was worth to spend some time learning and using this collaborative system (Q7). However, they stated that it was not completely clear what was necessary to develop during the experimental setting (Q9). It may be because we had a limited time to explain relevant aspects of the system and the risk discussion experiment. Although most of the participants agreed that the effort to use the system can decrease with time (Q8), this agreement was not a consensus among them. However, all participants stated that it is important to have a system to mediate such risk discussions (Q10). Finally, participants stated positive answers when (Q11) was analyzed (a question regarding the usability of the overall RD system resources) although free-text participants' suggestions indicated that the visual interface resources of the system may need improvements (e.g. such as the need of additional editing options in the discussion tree).

V. CONCLUSIONS

This paper presents a collaborative approach to risk management which is grounded on the definition, application and evaluation of a new dialogue game protocol. This protocol aims to support the elicitation and structuring of risk discussions and the recording of those discussions on a risk management memory. A Risk Discussion system is also discussed in which project stakeholders are fully able to collaborate on the development of key risk management tasks.

The main contribution of this paper is the argumentation-based approach defined as a new dialogue game for risk management. The adoption of this approach can promote the engagement of stakeholders in risk management discussions, offering a communication environment in which they exchange information with the goal of achieving better decisions. The approach is central on the elicitation and consideration of opinions and experiences from these project participants in order to achieve the systematic development of risk management tasks. Another benefit is the construction of a structured risk discussion memory in which collections of data and arguments regarding different risk characteristics are recorded explicitly.

Future works will aim the development of additional evaluation experiments involving the risk discussion of projects in different organizations. We will also pursue the investigation of new ways of querying the risk management memory that is constructed when the argumentation-based approach for collaborative risk management is adopted.

REFERENCES

[1] DEMARCO, T., LISTER, T.: Waltzing with Bears: Managing Risks on Software Projects. Dorset House Publishing Co., Inc. (2003).

[2] BOEHM, B.W.: Software Risk Management: Principles and Practices. Management. (1991).

[3] MCBURNEY, P., HITCHCOCK, D., PARSONS, S.: The Eightfold Way of Deliberation Dialogue. International Journal of Intelligent Systems. 22, 95–132 (2007).

[4] TOLCHINSKY, P., CORTÉS, U., MODGIL, S., CABALLERO, F., LÓPEZ-NAVIDAD, A.: Increasing Human-Organ Transplant Availability: Argumentation-Based Agent Deliberation. IEEE Intelligent Systems. (2006).

[5] REED, C., WELLS, S.: Dialogical argument as an interface to complex debates. Intelligent Systems, IEEE. (2007).

[6] MOULIN, B., IRANDOUST, H.: Explanation and argumentation capabilities: Towards the creation of more persuasive agents. Artificial Intelligence Review. 169–222 (2002).

Maturity Model and Lesson Learned for improve the Quality of Organizational Knowledge and Human Resources Management in Software Development

Flávio E. A. Horita, Marco I. Hisatomi, Fernando H. Gaffo and Rodolfo M. de Barros
Computer Department, State University of Londrina, Londrina/PR, Brazil
{feahorita, marco.hisatomi, fernandogaffo}@gmail.com, rodolfo@uel.br

***Abstract*—Constant changes created by the global market have led software organizations to depend increasingly on their intellectual capital, its human resources. In this context, the lessons learned are presented as an important resource to aid in the preservation and control of this intellectual capital. This paper aims to present a process model focused on gradually implement activities to improve the human resources management. Furthermore, this is also used to manage and share a repository of organization's intellectual capital, through a stream of lesson learned management. This process model was applied in software house located in a Brazilian public university. Their results showed improvements in both human resource management and in preserving the intellectual capital of the organization. Thus, this paper contributes to the scientific community by presenting a process model focused not only on improve the human resources management but also to increase intellectual capital of the organization.**

Keywords-Human Resources Management; Maturity Model; Lesson Learned; Software Development; Process Quality.

I. INTRODUCTION

Changes in the technological context achieve a high degree of innovation and agility, contemplating men as the protagonist of a new organization history [14]. To assimilate them, a cohesive development process, it is necessary that the organization has a capable team with skills and to propose corrective measures in a timely and effective. These factors emphasize the high dependence of human resources for the development and maintenance of software projects [14], [15]. Several processes and methodologies have been developed to aid in the management of these resources [10], [5], [9].

In similar context, the use of Lessons Learned (LL) during the performance of activities represent an important resource to assist both in the analysis and understanding of patterns, customs and ways of operation of teams and the planning of future projects. In general, they should report the actual result and expected decision detailing the facts and deviations occurred during this journey [1]. Therefore, the quest for maturity of human resource management, the use of LL can bring significant results [4]. In this sense, the objective of this paper is to present a process model to implement activities which improve gradually the human resource management and, in parallel, to increase knowledge about the historic achievement of its tasks and projects developed through the LL. Moreover, the organization analyzing these LL can identify forms and patterns of development teams performance.

In the pursuit of this goal, this paper is organized as follow. First, it's introduced background concepts used in this study. The research method used to develop this paper is brief presented in Section 3. In Section 4 we present the proposed process model. Its initial application is presented in Section 5. Finally, we conclude and present future works in Section 6.

II. BACKGROUND

A. Human Resources Management on Software Development

Several studies demonstrate that holding the best technological tools, using the most efficient techniques and work models is not enough to guarantee the success of a software project [8], [14], [15]. It is necessary the existence, in parallel, of a human resources management (HRM) able to develop skills and guarantee the effective allocation of its members, in order to increase the quality of its process [12].

However, several managers attribute more importance to the technical and practical areas rather than the human resources, which end up by losing the focus in software development processes [15]. Moreover, during the development of a software project, the dynamic in business processes and the high turnover of technologies, and his members highlights the importance to manage intellectual knowledge with creating mechanisms to collect, store and share it [5], [12].

B. Maturity Levels

Maturity models seek to establish levels of development of processes, called maturity levels that characterize stages in the implementation of improvement processes in the organization [2]. Thus, at each step in this journey, the model recognizes and signals the gradual maturity of the organization. Several maturity models were studied, among which we may highlight:

- People Capability Maturity Model (P-CMM): it is a maturity model variant of Capability Maturity Model

(CMM) which has as focus to help in HRM. To do so, it offers a set of good manners to make provisions for the continuous growing of workforce abilities in the organization [6].

- GAIA Human Resources (GAIA-HR): it is a framework composed by a maturity model, services, and diagnostic assessment questionnaire which aims to develop processes and factors that influence on the HRM [9].

C. Lesson Learned (LL)

A lesson is a knowledge gained through experience. The experience can be positive (good practices) as a successful test, or negative, as a failure. Both of them are considered lessons. A lesson must be significant, impacting on daily operations [5]. Basically, it is an acquired knowledge by observation or adverse experiences that cause an improvement in organization or to a particular individual.

There are several benefits of applying Lessons Learned (LL) within an organization, Roe [13] and Goes et al. [7] cite some of them: a) Saves time in solving problems, since the solutions of common problems are centralized in one location for easy access by members, b) Helps reduce or avoid costs from rework to correct defects already discovered, and c) Encourages the use of best practices within the organization, which improves the chance of success of the projects.

Can still be characterized as LL, narratives that explain knowledge gained through experience, which can be both positive and negative [3]. The LL record is an excellent way to avoid the mistakes made previously and that the successes achieved in the projects can be copied in future projects.

III. Research Methodology

The research methodology used in this paper was a case study. We chose this methodology because it offers an empirical research that offers researchers an object of applied study in its natural context [16]. Table I presents the steps followed on the research methodology.

Table I. Key Steps of Research Methodology

Step 1	**Literature Review**
	Reviewed literature and identified approaches used to develop the HRM.
Step 2	**Process – First Release**
	From this analysis, we elaborated the process first version which is defined its workflow and elements.
Step 3	**Process Application**
	Using this version, the process was applied on a case study.
Step 4	**Results Analysis**
	The results obtained with the case study, we analysis the beneficiates and problems evidenced.
Step 5	**Process – Final Release**
	The items identified on the last step were used to increase or add new elements to the process.

IV. Process Model

The process aims to implement gradually activities to improve the HRM and, in parallel, it will increase knowledge about the historic achievement of its tasks and projects developed through the LL. Fig. 1 shows the process model structure composed by eight activities and a LL incremental process. It used an intuitive notation that represents the main objective of each activity. Besides this, the blue arrows represent a connection between the activities, the green arrow sets the flow when the evaluation of services implantation is approved, and red arrow when this it is rejected.

Besides this, we can highlight the conditional step represent by a rhombus shape and its alternatives: (1) if the activity *Services Implantation* is approved the organization follows the green arrow and goes to next activity, *Increase Organization Maturity Level*, or (2) if the activity *Services Implantation* is rejected the organization follows the red arrow and go back planning the services implantation. Next, the process model activities and LL incremental process will be present in detail.

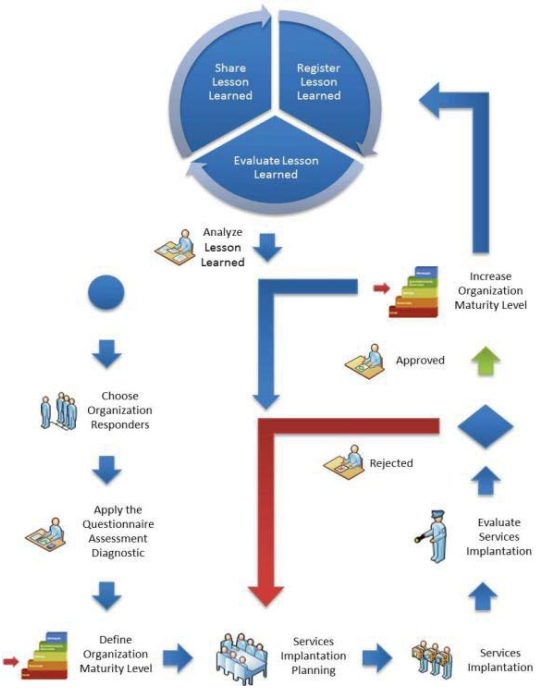


Fig. 1.Process Model for HRM Maturity using LL

A. Choose Organization Responders

Since this is a process model to assess the maturity of HRM in software teams, they represent the main actors in this scenario. With this assumption, the people involved this process should be committed to the organization, focusing on maximizing the knowledge and aware of their contribution to the improvement of this management [9].

As the first activity of the process, the choice of members of the organization who will participate in the implementation of improvements in the management of human resources has crucial role to ensure its success. Thus, this team should be prepared according to the following criteria: members aligned to the organization strategy, involved in day-to-day organization, and committed to knowledge dissemination.

B. Apply the Questionnaire Assessment Diagnotic

The purpose of this paper is the implementation of a process that demonstrates the practice of HRM, which can be verified by a suitable questionnaire. The questions should be based on principles that may assess over time. For this purpose we can evaluate both aspects: the skills of workers and the HRM process. For this, we used the models of Diagnostic Assessment Questionnaire (DAQ) presented by Gaffo e Barros [6] (The system used to Diagnostic Assessment Questionnaire (DAQ) can be found on this link: http://www.gaia.uel.br/gaia_ad/).

The result of this questionnaire should reflect the real situation of the organization as to the stage of HRM, this identification of the maturity level institutionalized in the organization, it is resulted by the low achieved index compared using the value range presented on the Table III of Gaffo e Barros [6] (an example of this can be found on Table II, the maturity level institutionalized on the initial application was two because the low achieved index of all services was the mobilize and staff).

Among his important points stand out: (1) human resources management, (2) motivation of the people involved in the project, (3) commitment of employees to project's success, (4) changes that affect people's performance, among others.

C. Define Organization Maturity Level

Based on the tabulation of the questionnaire results, the organization should be framed in a maturity level, thus establishing the landmark positioning of HRM. In this context, it was used the maturity model and services presented by Horita e Barros [9]. Further, the calculations used to define this maturity level were based on Gaffo e Barros [6].

D. Services Implantation Planning

The success of the HRM can be measured only after its effective implementation and evaluation, but the process itself must have a deployment plan. This activity aims to minimize the risk of failure during deployment. Resources should be sized to ensure that the deployment is successful, such as trained personnel and available computational resources, participation of other stakeholders, infrastructure, scheduling for each task, among others. The LL is also one of the techniques to be planned from the explicit knowledge to its spread, considering the experiences of success and failure.

E. Services Implantation

This is one of process model main activity, it is put into practice the tasks defined in the previous activities. Moreover, it is necessary to identify possible adjustments to maintain the initial goal. In this activity, the responsible persons should have a fundamental role to monitor the tasks safely and in line with the planning. However, it should also be prepared to take corrective action to adjust the faults that may occur during deployment.

F. Evaluate Services Implantation

The assessment roll is the activity in which you will be indicating whether the services were implemented properly and according to plan, i.e. it shows whether they are satisfactory, if adjustments need or are disapproved.

This review can happen through a new application of DAQ, through the application of personal interviews with selected staff in the activity, Choose Responders Organization, or applying a checklist evaluation [5].

G. Increase Organization Maturity Level

Improving HRM in the organization is a natural result of the evolution process. Therefore, it is necessary to perform new challenges to improve HRM, shortly after positive evaluation. Thus, this activity directs the organization to record the activities executed at Implantation Planning Services to define new services being deployed.

H. Lesson Learned Process

The activities proposed in this process model are based on the LL cycle. Planned in four stages, three of them are executed on a cycle form - register, evaluate, and share - and last one executed when a specific lesson are required. Initially, the registry is the explanation of the experience transformed into knowledge for possible use by others involved [13]. From the recorded knowledge, this is validated according to the requirements and criteria for the LL management.

Then the LL is disseminated to feed new lesson and ensure that the experiences will be useful to others involved in software development. According Alvarenga Neto [11], sharing the LL can happen by several media: internet, intranet, and groupware information repositories. On each evolution iteration of organization' maturity level, the LL are analyzed and effectively used to improve the software development process. Therefore, the process model provides a LL analysis to drive improvement in the organization.

V. PROCESS MODEL APPLICATION

In order to validate the process model on the case study, we selected one simple project of software house in a public university. This project was composed by seven use case defined with customers. Its development team was composed by four undergraduate, one master student with a medium knowledge on software development, test, and requirements elicitation. This team used a development process based on Project Management Body of Knowledge (PMBOK).

As defined by the process model, initially, DAQ was applied on the current members of development team and members of previous projects. These previous participants were selected using a simple questionnaire sent by email asking to these members participate of this questionnaire application. This application happened on Jan/2012. For this, we used the tool cited at Section IV.B.

.

In Aug/2012, a second DAQ application happened to the software house. This application aimed to validate and identify areas where improvements in services have been deployed and how to detect those needing improvement and redesigns. However, this time, were adopted as respondents, members other than those selected for the initial application, but they were involved in project used as case study. Table II shows the service rate achieved and a comparison with the initial rate.

Table II. Adherence Index on Final DAQ Application

Services	Initial Index	Final Index	Evolution of Index
Manage Human Aspects	32,81%	68,75%	35,94%
Manage Performance	29,06%	58,55%	29,49%
Knowledge Management	26,14%	49,24%	23,10%
Manage Training	31,64%	52,82%	21,18%
Mobilize and Staff	23, 67%	53,00%	29,33%
Human Resources Plan	33,12%	54,06%	20,94%
Review the Business Needs	25,93%	53,70%	27,77%

After analyzing the results of initial application of DAQ, it was evidenced the necessary improvement on three services (highlighted by underline). Because of low index of mobilize and staff it was gives to the organization the maturity level two. From data shown by Table II, after the implementation of the services suggested, it is highlighted the rates achieved by focus areas all of them showing a growth above 20%. The attendance rates achieved have also enabled the migration of HRM software house for the level of maturity of three maturity model since the lower rate is 49.24%.

Besides this, it was also used to analyze the LL one indicator that aims to provide growth in the level of knowledge managed by the organization after the framework implementation. To do so, are compared and analyzed their contents generated with those approved. This approval aims to ensure that they are stored only those relevant to the aid of the projects. Fig. 2 shows this indicator for the case study.

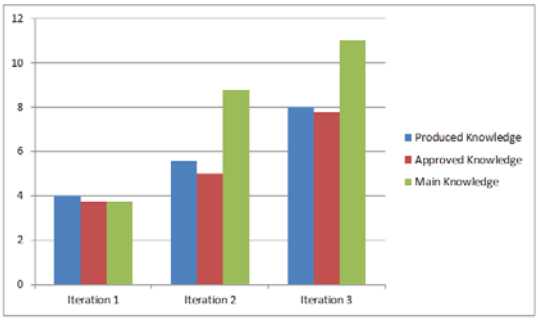


Fig. 2. Produced Knowledge vs. Approved Knowledge

As shown in Fig. 2 through the process model implantation, we identified an increase of 65 % in the project selected as case study. In addition, this analysis can emphasize the increased production of knowledge qualified to be stored and used in future projects of the organization.

VI. Conclusion and Future Work

This main contribution of this paper is to present a process model to improve the activities in human resource management and to increase the historic achievement of its tasks and projects developed through the LL. Moreover, LL analyzing aims to understand the performance patterns of development team and help on future projects planning.

The experience of process model applying was possible to evidence his efficiency on improving the activities of human resource management in software house used as case study. Furthermore, when it was used together on an evolutionary HRM process, the LL cycle showed efficient by analyzing the indicator associated. In future lines of work, we will try to apply the process model on other companies and we intend to integrate the multi-criteria analysis to help on identify, filter, summarize, and select the set of the best LL created.

References

[1] A. S. Al-Mudimigh, Z. Ullah, and T. A Alsubaie. A framework for portal implementation: A case for saudi organizations. *International Journal of Information Management*, 2011.

[2] Associação para Promoção da Excelência do Software Brasileiro. *Modelo de Referência para Melhoria de Processo do Software Brasileiro para Software (MR-MPS-SW)*, Agosto 2012.

[3] Kessler F. Brett, F. and D. Dressler. The 24 keys to high performance. In *Frontline Group Organizational Learning Division*, 2000.

[4] Patricia Carrillo, Kirti Ruikar, and Paul Fuller. When will we learn? improving lessons learned practice in construction. In *International Journal of Project Management*, 2012.

[5] Hefley B. Curtis, B. and P-CM Miller, S. P-cmm: People capability maturiry model. Technical report, Software Engineering Institute, June 2009.

[6] F. H. Gaffo and Rodolfo M. Barros. Gaia risks - a service-based framework to manage project risks. In *XXXVIII Conferencia Latinoamericana en Informática*, 2012.

[7] Anderson S. Goes, Marco I. Hisatomi, Bruno O. Mesquita, and Rodolfo M. Barros. Applying lessons learned as an improved methodology for software project management. In *IADIS International Conference Information Systems, 2013, Lisboa*, 2013.

[8] Orit Hazzan and Irit Hadar. Why and how can human-related measures support software development processes? *Journal of Systems and Software*, 81(7):1248 – 1252, 2008.

[9] Flávio E. A. Horita and Rodolfo M. Barros. Gaia human resources - an approuch to integrate itil and maturity levels focused on improving the human resource management in software development. In *25th International Conference on Computer Applications in Industry and Engineering (CAINE)*, 2012.

[10] Flávio E. A. Horita, Jacques D. Brancher, and Rodolfo M. Barros. A process model for human resources management focused on increasing the quality of software development. In *24th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, 2012.

[11] Rivadávia C. D. Alvarenga Neto and Chun W. Choo. The post nonaka concept of ba: eclectic roots, evolutionary paths and future advancements. In *Proceedings of the 73rd ASIS&T Annual Meeting on Navigating Streams in an Information Ecosystem*, 2010.

[12] Yimeng Qiu. Human Resource Management Based on Human Capital in Enterprises. *Personnel*, 2011.

[13] T. H. Roe. Establishing a lessons learned program: Observation, insights and lessons. In *Center for Army Lessons Learned*, 2011.

[14] Xiaohong Shan, Guorui Jiang, and Tiyun Huang. The optimization research on the human resource allocation planning in software projects. In *International Conference Management and Service Science (MASS) on*, pages 1 –4, aug. 2010.

[15] Hamid Tohidi. Human resources management main role in information technology project management. *Procedia Computer Science*, 3:925–929, January 2011.

[16] R. K Yin. *Case Study Research: Design and Method*, volume 5. Third edition edition, 2002.

Recovering Software Architectural Knowledge from Documentation using Conceptual Model

Mojtaba Shahin[1,2], Peng Liang[1]*, Zengyang Li[3]
[1] State Key Lab of Software Engineering, Computer School, Wuhan University
[2] Department of Computer Engineering, Neyriz Branch, Islamic Azad University
[3] Department of Computing Science, University of Groningen
mojtabashahin@gmail.com, liangp@sklse.org, zengyang.li@rug.nl

***Abstract*—Software architectural knowledge (AK) is the integrated representation of the software architecture (SA) of a software-intensive system, the architectural design decisions, and the external context/environment. AK annotation using AK conceptual model is used to recover formal AK from SA documentation, including architecture design as well as the design decisions, rationale, context, and other factors that together determine architecture solutions. But there is no evidence on how architects, especially junior architects, understand and annotate SA documents and recover formal AK from the documents using an AK model, which is right the case when a new architect jumps into a project, trying to understand the SA documents created by previous architects. This paper first presents AKRCM (AK Recovery using Conceptual Model) approach for recovering AK from SA documents. Second, we conduct a descriptive study using experiment to investigate how junior architects annotate SA documents and recover AK using AKRCM approach. We found that an AK conceptual model is beneficial for junior architects to get a fair understanding of SA documents, and to recover better-quality AK from SA documents.**

Keywords - architectural knowledge; knowledge recovery; knowledge annotation; junior architect; conceptual model

I. INTRODUCTION

Software architecture is considered of paramount importance to the software development life cycle [1]. It is a key artifact for the early analysis of the system, as it facilitates stakeholders' communication and understanding, and drives both system construction and evolution. Software architecting is meanwhile a knowledge-intensive activity, in which a large amount of knowledge is being continuously produced and consumed. In the field of software architecture (SA), a paradigm shift has occurred from describing the outcome of architecting process to describing the Architectural Knowledge (AK) created and (re)used during architecting process, including architecture design as well as the design decisions, rationale, assumptions, context, and other factors that together determine architecture solutions [4]. Traditional approaches to documenting SA are limited to capture this knowledge and partially result in AK vaporization. The architecturally significant information is lost during architecting process, especially in architecture evaluation and maintenance activities. This situation leads to many severe problems, such as expensive system evolution, lack of stakeholders' communication, and limited reusability of architecture [2]. The SA community, both in industry and academia, is therefore gradually acknowledging that capturing and recovering explicit and valuable AK will lead to the improvement of architecting process, and of architecture documentation itself [3].

In knowledge management, a distinction is often made between two types of knowledge: implicit and explicit knowledge [7]. Implicit (or tacit) knowledge is the knowledge residing in people's heads, whereas explicit knowledge is the knowledge which has been codified in certain form (e.g., a document or a model). Two forms of explicit knowledge can be discerned: documented and formal knowledge. Documented knowledge is explicit knowledge which is expressed in natural language or images in documents. Typical examples of documented AK are Word documents that contain architecture descriptions. Formal knowledge is explicit knowledge codified using a formal language or conceptual model of which the exact semantics are defined. Typical examples of formal AK models include AK ontologies [5] or AK conceptual models [8] that formally define the AK concepts and their relationships, and aim at providing a common language for unambiguous interpretation by stakeholders. In comparison with documented AK, formal AK provides a clearer description of AK entities and their relationships in SA documents supported by conceptual model. This paper focuses on the recovery of formal AK that is codified in an AK conceptual model.

Formal AK recovery from SA documentation is critical to various architecting activities, e.g., architecture evaluation and maintenance. Jansen *et al.* stated that SA is often documented after most of AK have been produced and used [6]. Therefore, a large amount of AK may get lost during architecting process and SA documents get quickly outdated, also the trust to SA documents is decreased. To alleviate these problems, an architect needs to recover formal AK from SA documents and identify the gap in AK (e.g., an *architectural design decision* without supporting *design rationale*) with the support of formal AK management tool (e.g., Knowledge Architect [13]). In a human perspective, AK recovery activity might take place in the following situations: (1) AK recovery for understanding existing design and making a new design by architects. (2) AK recovery for communicating architecture design through AK sharing [11]. In both cases, architects need to first recover AK from SA documentation.

An annotation is generally a comment, note, explanation, or other types of external remarks that can be attached to an object, i.e., a sentence in a SA document. Different from general text annotation, formal AK annotation uses controlled terms (i.e., the AK concepts from a specific AK model) to annotate the SA documents. We name this approach AKRCM

* Corresponding author

This work is partially sponsored by the NSFC under Grant No. 61170025 and AFR-Luxembourg under the contract No. 895528.

(AK Recovery using Conceptual Model). A sample of AK annotation using AKRCM approach in a SA document is shown in Figure 1. Note that practical AK annotation is generally made by tools, e.g., Word plug-in [13]. The snapshot in Figure 1 just shows how the subjects in this experiment annotate the SA document (See Section III). AK recovery requires a lot of design experiences and domain expertise of the architects that normally junior architects don't have. The other characteristic of junior architects is that they need more time to get familiar with a new system and may overlook some key architecture information when they jump into a new project. To the best of our knowledge, there is currently no any evidence on how software architects, especially junior architects, annotate the text in SA documents and recover formal AK using an AK conceptual model.

To this end, we conducted an experiment: asking the participants (a group of junior architects, in our setting, the master students following a SA course) to recover formal AK from a SA document using AKRCM approach based on their understanding of the SA document with the support of a specific AK model. We employ LOFAR AK conceptual model (the concepts and their relationships are shown in Figure 2 and detailed in Section III) to annotate and recover AK from an LOFAR SA document. The reason for choosing LOFAR AK model is that the SA document used for this experiment is from LOFAR (Low Frequency Array) project undertaken by Astron, the Dutch Astronomy Institute, which is involved in the development of large software-intensive systems used for astronomy research. It would be easier for participants to recover most AK using this model. In this work, we first investigate how junior architects understand AK concepts in LOFAR AK model, and how they annotate the SA document and recover AK using this conceptual model. Second, the effectiveness of an AK conceptual model in AK recovery is analyzed and evaluated.

The reminder of this paper is organized as a follows: Section II presents the challenge of AK recovery and research questions we try to answer by the experiment. Section III describes the experimental setup on AK recovery, with the threats to the validity of the experimental results. To answer the research questions, the experiment results are reported and analyzed in Section IV. Section V discusses the threats to the validity of this study. Section VI presents the lessons learned according to the experiment results. Section VII discusses related work on recovering AK. Section VIII concludes this paper with future work directions.

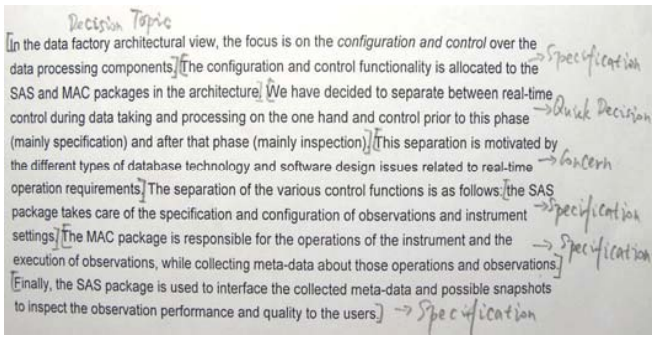


Figure 1. Snapshot of AK annotations in a SA document

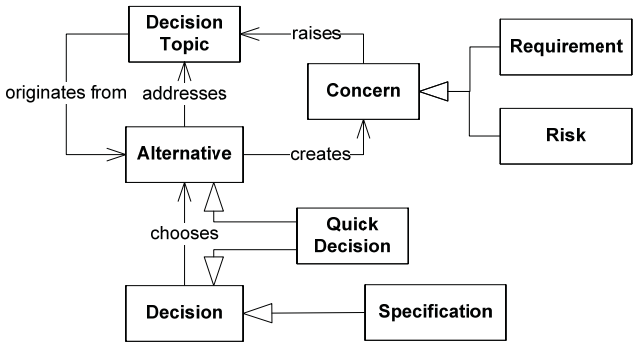


Figure 2. LOFAR AK conceptual model

II. GOAL AND CHALLENGE OF AK RECOVERY

The goal of this experiment is to present the findings on formal AK recovery using an AK model by junior architects. The next step, i.e., how to use recovered AK in SA activities, such as architecture evolution, is scheduled as the future work.

The challenge this paper tries to address is how an AK conceptual model can help junior architects to recover AK from SA documents. This challenge can be further detailed in three research questions:

RQ1: How junior architects understand AK concepts in an AK conceptual model? A domain-specific AK conceptual model is normally co-constructed by experienced architects and domain experts, but junior architects with few background and experiences may have different understanding to the same AK concept.

RQ2: How junior architects annotate SA documents and recover AK? Experienced architects may recover AK from SA documents by employing their tacit knowledge (e.g., experience on architecture patterns, domain knowledge, architecture reasoning knowledge, etc.) in the recovery process [6]. In other words, AK recovery results may heavily depend on the skills and experiences of software architects, which junior architects normally lack of.

RQ3: Is the AK conceptual model helpful for junior architects to recover better-quality AK from SA documents? To answer this question, we try to compare the results of AK recovery by junior architects and experienced architects to investigate the effectiveness of an AK conceptual model in AK recovery.

III. EXPERIMENT SETUP FOR AK RECOVERY

We conducted a descriptive study using experiment to answer the research questions in Section II. Descriptive study is an observational study that tries to determine the distribution of certain characteristics or attributes. The focus of a descriptive study is not about why the observed distribution exists, rather what it is [12].

At Wuhan University, SA course is introduced as an optional course in the curriculum for the first-year master students major in software engineering. 36 students participated in this course in the academic year of 2010-2011. During this course, the students learned fundamental knowledge about architecture design, process, and evaluation by conducting an

architecture course project. We regard these students as junior architects in this experimental context since Host *et al.* found that students are suitable replacements for industry professionals if performing small tasks of judgment [14]. The task of this experiment is to ask students to annotate one page (about 400 words) of corpus selected from the LOFAR SA document[1] using LOFAR AK conceptual model, which are both produced in an industrial project. A snapshot of the AK annotations by annotators (i.e., the subjects in this experiment) is shown in Figure 1. The experiment steps are following:

- **Step1**: hand out the printed text in one page selected from a SA document to the students (i.e., the subjects);
- **Step2**: present a short tutorial in 10 minutes on the AK conceptual model (i.e., LOFAR AK model) that are used for the AK annotation with AK annotation examples;
- **Step3**: ask the subjects to annotate the printed text (using pen) with the AK concepts of the introduced AK conceptual model within limited time (15 minutes).

A short description about the LOFAR AK model shown in Figure 2: A *Concern* is an interest to the systems development. A *Requirement* represents something demanded from the system, which is a specific type of *Concern*. A *Risk* is also a special type of *Concern*, which expresses a potential hazard. A *Decision Topic* is a certain problem that a *Decision* should be made to solve it. *Alternative* concept represents one or more potential candidate solutions to address the described problem. A *Decision* is chosen from the *Alternatives* to address the described *Decision Topic*. If only one *Alternative* is proposed and no (sufficient) motivation is given why this *Alternative* is chosen, we then have a *Quick Decision*. A *Specification* is the lowest level architectural *Design Decision* that is being made. The refinement process of architecture documentation is finished when it comes to *Specifications*. The concepts *Concern*, *Decision*, *Alternative* and the like are not specific concepts to the LOFAR AK model. These concepts exist in most AK conceptual models although different AK models might employ these concepts with different terms and with different relationships [9].

IV. EXPERIMENT RESULTS AND ANALYSIS

There were 36 subjects (students), and 35 AK annotation results were returned, in which 33 results were valid with 2 invalid results: one was no any AK annotations, and the other one provided annotated text without annotation concepts. The SA document (in one page) is composed of 16 sentences and 1 diagram, which are potential annotatable AK entities. Some subjects also annotated part of a sentence as an AK entity or annotated more than one sentences as an AK entity, in which the unit of AK annotation can be a phrase, a paragraph, or a sentence. In this section, we try to answer the research questions presented in Section II.

RQ1: How junior architects understand AK concepts in an AK conceptual model?

As shown in Table 1, different software architects may annotate same unit of text with different AK concepts, because they have different understanding about specific AK concept, or some AK concepts are actually overlapped with each other, e.g., *Specification* is a `subClassOf` *Decision*, *Decision Topic* is `raisedFrom` *Requirement*, etc. One of best practices to resolve this issue is to combine two AK concepts into one AK concept when most of architects annotate the same unit using the two AK concepts alternatively. For example, in LOFRA AK model, the concept *Requirement* is a `subClassOf` *Concern* and most of junior architects annotate the same unit using these two AK concepts alternatively. Table 1 shows that most of junior architects (78.8%) annotate the Sentence 1 and 2 with *Requirement* and *Concern* alternatively. Therefore, it is suggested that these two concepts are combined as one concept, and the merging of AK concepts is considered as a post-processing to annotated AK. Note that, the focus of this work tries to investigate how junior architects recover AK with the support of an AK model, but not to evaluate the effectiveness of an AK model for AK annotation. As a side-effect of AK recovery activity, the AK model can be improved according to the AK annotations, which is not discussed in details in this paper.

RQ2: How junior architects annotate SA document and recover AK?

Junior architects annotate the text in different granularity, e.g., one junior architect annotates two sentences as a *Requirement*, and the other one annotates these two sentences as two *Requirements*. For example, Table 1 shows that Subject 26 annotated Sentence 1 and 2 as one AK entity of *Requirement*, while Subject 16 annotates the two sentences as two different AK entities of *Requirement* and *Specification* respectively. Most of junior architects did not consider the figure (and related caption) in the SA document as an AK entity. Figure 3 shows what kind of text units (phrase, sentence, and paragraph) are mostly annotated by junior architects. The result shows that *sentence* is mostly used as the unit of AK annotation. We assume that this is partially because sentence is a unit that has a logical-sound meaning.

RQ3: Is the AK model helpful for junior architects to recover better-quality AK from SA documents?

A consistent annotation in this experiment refers to the AK concept used by the most of subjects to annotate AK entities. Figure 4 and Table 1 show how junior architects annotate the 16 sentences. For example, 9, 16, 1, and 7 out of 33 subjects annotated Sentence 1 as *Requirement*, *Concern*, *Decision Topic*, and *No-AK entity* (No-AK entity means that subjects did not consider the sentence as any AK concept) respectively. Therefore, we regard that junior architects get a consistent AK annotation of Sentence 1 as *Concern*, which is listed in the column 4 of Table 2.

To evaluate the quality of AK annotation and recovery results by junior architects, the results by junior architects are compared to the AK recovery results by experienced architects. We asked two LOFAR architecture experts to annotate the same SA corpus. Table 2 (columns 2 and 3) shows that the annotation results by LOFAR experts 1 and 2 are largely overlapped with each others, i.e., **62%** of AK annotations by experts 1 and 2 are similar (in yellow rows). The comparison of the annotation results between junior architects (column 4 of Table 2) and two experts (columns 2 and 3 of Table 2) shows that **57%** of AK annotations by them are identical (in

[1] The corpus of the LOFAR SA document is available at http://www.cs.vu.nl/~liangp/project/AKRv/LOFARdoc.pdf

yellow rows). The comparison result supports our hypothesis that a pre-defined AK model (e.g., the LOFAR AK model in this experiment) has a very positive impact to help junior architects to get a fair understanding of SA documents and recover better-quality AK.

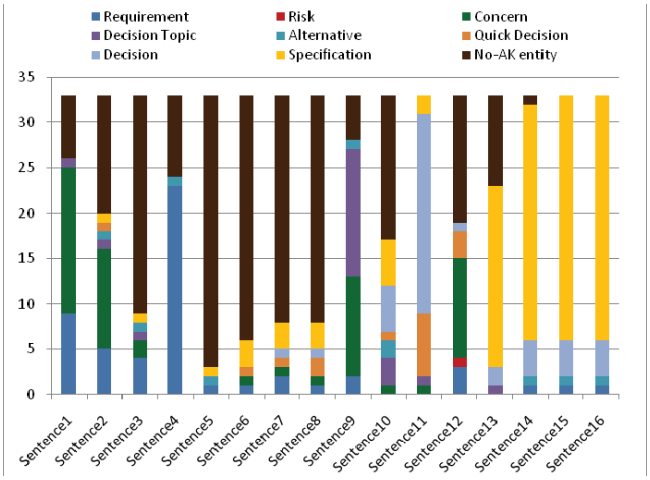


Figure 4. Consistent annotations with the support of an AK conceptual model

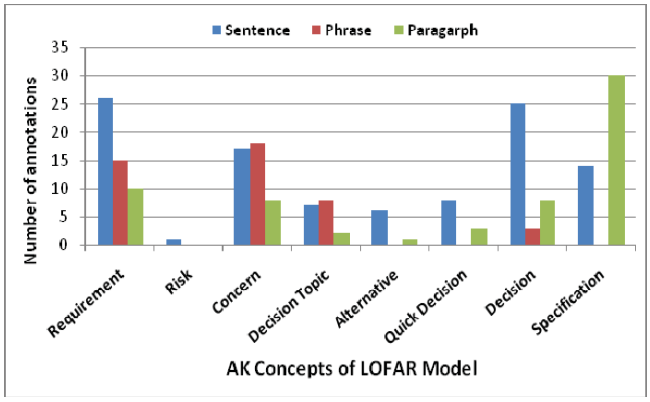


Figure 3. Units of AK entities annotated in LOFAR AK concepts by junior architects

As shown in Table 2, the LOFAR architecture experts annotated more AK entities (e.g., *Requirement*, *Quick Decision* in Sentence 5~8) than junior architects did. The implication of this difference is that the AK conceptual model can do help junior architects to understand/annotate/recover AK in SA documents, but domain knowledge about projects is also necessary/helpful and sometimes critical to recover AK.

Table 1. AK annotation results by junior architects

	Sentence															
Subject	**S1**	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**	**S8**	**S9**	**S10**	**S11**	**S12**	**S13**	**S14**	**S15**	**S16**
Subject 01				R			R	QD	DT	QD	D		S			
Subject 02	R			R					C		D		S			
Subject 03	R								C	DT	S		D			
Subject 04	C			R					DT		D	QD	S			
Subject 05	R	A				QD			DT	D				A	A	A
Subject 06		C		R			D				QD				S	
Subject 07	DT			R					C		D	D	S			
Subject 08	R								C	DT	QD			S		
Subject 09	C		R			S			A		QD		S			
Subject 10	C			R					DT	A	D	C	S			
Subject 11	C			R					DT		D			S		
Subject 12	C			R					DT	A	D		S			
Subject 13	C	C		R					C	C	QD			S		
Subject 14				R					C		C	Rs	S			
Subject 15	R			R					DT		QD	R	S			
Subject 16	R	S		R					DT	S	QD	R	S			
Subject 17	C									S	S		DT	D		
Subject 18	C			R							D			S		
Subject 19									R	S	D	C	S			
Subject 20	C								C		D	C	S			
Subject 21									C		D		S			
Subject 22	C			R					DT		D	C		S	S	S
Subject 23	C					S			DT	D	D	R		R	R	R
Subject 24	C		R						C		D		S			
Subject 25	R	C	DT	R	S				R	S	D	C	S			
Subject 26	R					S					D	C	S			
Subject 27	C		R						DT		D		S			
Subject 28	C		R						DT	D	QD	C	S			
Subject 29				R					DT		D		S			
Subject 30	C	QD		R					C	S	DT	C		S		

Subject 31				R		C		DT		D	C	S	
Subject 32	R	R		R		R	R		D	D	C		D
Subject 33	C		S	R				C	D	D	C	D	

Requirement (R), Concern (C), Decision Topic (DT), Quick Decision (QD), Decision (D), Alternative (A), Risk (Rs), Specification (S)

Table 2. Comparison of AK annotations between LOFAR experts and junior architects

	LOFAR Expert 1	LOFAR Expert 2	Consistent Annotations by Junior architects
Sentence 1	Concern	Concern	Concern
Sentence 2	Concern	Requirement	Concern
Sentence 3	No-AK entity	No-AK entity	No-AK entity
Sentence 4	Requirement	Requirement	Requirement
Sentence 5	No-AK entity	Quick Decision	No-AK entity
Sentence 6	No-AK entity	Quick Decision	No-AK entity
Sentence 7	Requirement	No-AK entity	No-AK entity
Sentence 8	Requirement	No-AK entity	No-AK entity
Sentence 9	Decision Topic	Decision Topic	Decision Topic
Sentence 10	Specification	Quick Decision	No-AK entity
Sentence 11	Quick Decision	Quick Decision/Decision	Decision
Sentence 12	Concern	Concern	Concern
Sentence 13	No-AK entity	No-AK entity	Specification
Sentence 14	Specification	Specification	Specification
Sentence 15	Specification	Specification	Specification
Sentence 16	Specification	Specification	Specification

V. THREATS TO VALIDITY

Due to the limitations of this experiment, there are several threats to the validity of the experiment results:

The size of the SA document used in this experiment is relatively small due to the time limitation of the experiment since this experiment was conducted during a two-hour course session. We plan to extend the size of the experimental SA document to cover all the LOFAR AK concepts.

In this experiment, we use one AK model (i.e., LOFAR AK model). The LOFAR model is a specific AK model for the SA documentation of our industrial partner, and may not cover the AK concepts in other SA documents. We plan to repeat this experiment with more AK models, and the general AK core model proposed in [8], in order to investigate how junior architects annotate AK using various AK models, and what kind of model is more cost-effective for AK recovery.

The third threat of validity is related to the selected SA document, which is from the domain of astronomy research, and the subjects of this experiment are junior architects who are not the experts in that domain. Therefore, the subjects (students) may have some difficulties to understand correctly the SA document of the LOFAR system (e.g., this SA document may include specific and domain information which is only understandable after special training).

VI. LESSONS LEARNED

The experiences and issues during applying AK recovery method using AK conceptual model (AKRCM) are discussed:

(1) The AK conceptual model used for AK recovery is not fixed. An AK conceptual model can evolve (e.g., merging, splitting, modifying, and removing AK concepts) according to users annotations, in order to get a wider consensus on annotation results and get more AK being recovered.

(2) The AK recovery method using conceptual model focuses on explicit AK (i.e., from documented AK to formal AK) in SA documents without considering tacit AK (e.g., implicit design rationale and decisions). The method for recovering architectural design decisions (an important type of AK) proposed in [6] can be complementary to our method since the two methods cover both explicit and tacit knowledge in architecture design, documentation, and maintenance.

(3) In current software development practices, most of software artifacts are recorded and managed in the form of file-based documents besides SA documents, including requirements documents, test specifications, etc. The preliminary results of AK recovery from SA documents suggest that knowledge recovery using specific conceptual models can be extended and employed in the formal knowledge recovery of other file-based software artifacts. Furthermore, we propose that software knowledge recovery using conceptual models, as part of knowledge reengineering activity in software development, can be synthesized as a best practice in reverse engineering of software-intensive systems.

VII. RELATED WORK

Capturing and recovering AK are two complementary ways to record AK during architecting process. Capturing AK is different from recovering AK in that AK capture normally

takes places during architecture design activity, while AK recovery is an activity after the fact [6]. ADDRA (Architectural Design Decision Recovery Approach) is an approach to recover AK especially architectural design decisions and document them [6]. Archium conceptual model is used as input in ADDRA to document the recovered AK. The concern of this approach is that it is heavily based on tacit knowledge of original architect to recover AK. Roeller *et al.* proposed an approach to recover architectural assumptions (the architectural design decisions that are implicit and undocumented) from existing software artifacts of software products [10]. The approach, RAAM (Recovering Architectural assumptions Method), uses *documentation*, *financial reports*, *free interviews*, *version control*, and *source code* as inputs, and then produces a list of assumptions.

ADDRA and RAAM are both heavyweight approaches that try to recover tacit knowledge from existing SA documents and transform them into explicit knowledge. The AK recovery method in our work is based on annotations of existing SA documents to transform the documented AK to formal AK, which is more meaningful than documented AK for understanding and communicating architecture design, and facilitates AK reasoning for e.g., design maturity assessment.

There are also some work on recovering implicit AK (e.g., architecture layers and externally visible features) at source code level using knowledge annotations, e.g., in [15]. The advantage of this method is that it does not mandate a fixed conceptual model for knowledge annotation in advance, but employs an iterative process to refine flexible annotation types (i.e., concepts in AK models), which introduces a viable solution for iteratively refining AK models using the results of AK annotations (see the point 2 of the future work).

VIII. Conclusions and Future Work

In this paper, we conducted a descriptive study using experiment to investigate how an AK conceptual model can help junior architects to recover AK from existing SA documents in order to further support other architecting activities. Experiment results indicate that an AK model can do help junior architects in AK recovery in two aspects: (1) it helps junior architects get a fair understanding of SA documents and recover better-quality AK; (2) it has a very positive impact to junior architects to make AK annotation.

Based on the experiment results and analysis, we outline the promising future work in several points: (1) Is an AK conceptual model useful for AK recovery by experienced/expert architects even if little impact is identified to them? (2) Can recovered AK and unrecovered AK (i.e., No-AK entities) helps to refine the domain-specific AK conceptual model (e.g., add, remove, or modify AK concepts according to the AK annotations)? (3) How architects are going to annotate SA documents, if an AK conceptual model is not presented to them? AK annotation without following an AK conceptual model is called AK tagging, which produces various AK tags we can not expect since architects can use whatever tags they like, including domain-independent AK tags (e.g., *Design Decision*, *Pros*, and *Cons*) and domain-dependent AK tags (e.g., pattern tags, function-related tags, performance, security, etc.). It is useful to investigate how the two methods (i.e., AK annotation and tagging) can be combined to achieve better AK recovery results. (4) Conduct this experiment in an industrial context with experienced architects and industrial-size SA documents to understand the challenges of AK recovery that architects face in architecting process. (5) Can this AK recovery method using conceptual model be employed and beneficial to other knowledge recovery activities in software development, and further facilitate reverse engineering of software-intensive systems? For example, requirements rationale knowledge recovery from requirements specifications to support evolution of requirements [16].

References

[1] Bass, L., Clements, P., and Kazman, R., 2003. Software Architecture in Practice, 2nd edition. Addison-Wesley Professional.

[2] Jansen, A. and Bosch, J., 2005. Software architecture as a set of architectural design decisions. In: Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA), pages 109-120.

[3] Lago, P. and Avgeriou, P., 2006. First workshop on sharing and reusing architectural knowledge. *ACM SIGSOFT Software Engineering Notes*, 31(5):32-36.

[4] Kruchten, P., Lago, P., van Vliet, H., and Wolf, T., 2005. Building up and exploiting architectural knowledge. In: Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA), pages 291-292.

[5] Kruchten, P., 2004. An ontology of architectural design decisions in software intensive systems. In Proceedings of the 2nd Groningen Workshop on Software Variability Management (SVM), pages 54-61.

[6] Jansen, A., Bosch, J., and Avgeriou, P., 2008. Documenting after the fact: recovering architectural design decisions. *Journal of Systems and Software*, 81(4):536-557.

[7] Nonaka, I. and Takeuchi, H., 1995. The Knowledge-Creating Company, How Japanese Companies Create the Dynamics of Innovation. Oxford University Press.

[8] de Boer, R., Farenhorst, R., Lago, P., van Vliet, H., Clerc, V., and Jansen, A., 2007. Architectural knowledge: Getting to the core. In: Proceedings of the 3rd International Conference on the Quality of Software-Architectures (QoSA), pages 197-214.

[9] Shahin, M., Liang, P., and Mohammad, R. K., 2009. Architectural design decision: existing models and tools. In: Proceedings of the 8th Working IEEE/IFIP Conference on Software Architecture (WICSA), pages 293-296.

[10] Roeller, R., Lago, P., and van Vliet, H., 2006. Recovering architectural assumptions. *Journal of Systems and Software*, 79(4):552-573.

[11] Liang, P., Jansen, A., and Avgeriou, P., 2009. Sharing architecture knowledge through models: quality and cost. *The Knowledge Engineering Review*, 24(3):225-244.

[12] Wohlin, C., Host, M., and Henningsson, K., 2003. Empirical research methods in software engineering. Empirical Methods and Studies in Software Engineering, Springer, pages 145-165.

[13] Liang, P., Jansen, A., and Avgeriou, P., 2009. Knowledge Architect: A Tool Suite for Managing Software Architecture Knowledge. Technical report RUG-SEARCH-09-L01, SEARCH, University of Groningen, February 2009.

[14] Host, M., Regnell, B., and Wohlin, C., 2000. Using students as subjects - a comparative study of students and professionals in lead-time impact assessment. *Empirical Software Engineering*, 5(3):201-214.

[15] Brühlmann, A., Gîrba, T., Greevy, O., and Nierstrasz, O., 2008. Enriching reverse engineering with annotations. In: Proceedings of the 11th International Conference on Model Driven Engineering Languages and Systems (MoDELS), pages 660-674.

[16] Liang, P., Avgeriou, P., and He, K., 2010. Rationale management challenges in requirements engineering. In: Proceedings of the 3rd International Workshop on Managing Requirements Knowledge (MaRK), pages 16-21.

Knowledge Management Applied to Software Testing: A Systematic Mapping

E. F. Souza
Computação Aplicada
Instituto Nacional de Pesquisas Espaciais, INPE
São José dos Campos/SP, Brasil
erica.souza@lac.inpe.br

R. A. Falbo
Departamento de Informática
Universidade Federal do Espírito Santo, UFES
Vitória/ES, Brasil
falbo@inf.ufes.br

N. L. Vijaykumar
Lab. de Comp. e Matem. Aplicada
Instituto Nacional de Pesquisas Espaciais, INPE
São José dos Campos/SP, Brasil
vijay@lac.inpe.br

***Abstract* - With the growth of data from several different sources of knowledge within an organization, it becomes necessary to provide computerized support for tasks of acquiring, processing, analyzing and disseminating knowledge. In the software process, testing is a critical factor for product quality, and thus there is an increasing concern in how to improve the accomplishment of this task. In software testing, finding relevant knowledge to reuse can be a difficult and complex task, due to the lack of a strategy to represent or to associate semantics to a large volume of test data, including test cases, testing techniques to be applied and so on. This paper aims to investigate, through a Systematic Mapping of the Literature, some aspects associated with applying Knowledge Management to Software Testing.**

Keywords: Software Testing, Knowledge Management, Systematic Mapping

I. INTRODUCTION

Software development is an error prone process. To achieve quality software products, it is essential to perform Verification & Validation (V&V) activities throughout the software development process. Verification determines whether the development products of a given activity conform to the requirements of that activity. Validation refers to whether the software satisfies its intended use and the user needs [1]. V&V activities can be static and dynamic. Dynamic V&V activities require the execution of a program, while static V&V activities do not. Static V&V are typically done by means of technical reviews and inspections. Dynamic V&V are done by means of testing [2]. Thus, Software Testing consists of the dynamic V&V of the behavior of a program on a finite set of test cases, against the expected behavior [3].

Due to advances in technology and the emergence of increasingly critical applications, tests have become more and more complex. Currently, software testing is considered a process consisting of activities, techniques, resources and tools. During software testing, a large number of information is generated. In fact, software testing is a knowledge intensive process, and it becomes necessary to provide computerized support for tasks of acquiring, processing, analyzing and disseminating knowledge for reuse [4].

Finding relevant knowledge in software testing is not an easy task. There is a need to represent and process knowledge in an affordable and manageable manner. In this context, principles of Knowledge Management (KM) are pointed out as an important means to manage software testing knowledge [5].

The main goal of KM is to promote knowledge storage and sharing, as well as the emergence of new knowledge [6]. This paper presents a systematic mapping of the literature in order to identify the primary studies that applied principles of KM to software testing. A systematic mapping provides a broad overview of an area of research, to determine whether there is research evidence on a particular topic. Results of such mapping may identify suitable areas for performing systematic reviews and also areas where a preliminary study is more appropriate. A systematic mapping also helps identifying gaps in order to suggest areas for future research and provides a map that allows appropriately to position new research activities [7].

The systematic mapping presented in this paper investigates the following issues: (i) problems related to knowledge in software testing; (ii) organizations' purposes of applying KM in software testing; (iii) types of knowledge items typically managed in the context of software testing; (iv) benefits and problems reported on the implementation of KM initiatives in software testing; and (v) mechanisms or technologies used to provide KM in software testing. This mapping was structured in five research questions, and 336 studies were selected and analyzed according to a systematic mapping method.

This paper is organized as follows. Section 2 presents the main concepts used in this paper. Section 3 describes the systematic mapping method applied, and discusses the main parts of the mapping protocol used, including research questions, inclusion and exclusion criteria, searched sources and search string. Section 4 presents the main results of the mapping, discussing the selection process, the classification schemas, and presenting data synthesis. Section 5 discusses the findings and the mapping limitations. Finally, Section 6 presents conclusions and future directions for this research.

II. BACKGROUND

In this section, we discuss briefly some of the most important concepts in the research areas studied (namely Software Testing and KM), in order to characterize the scope of our investigation and to support the definition of the research questions that are the subject of the systematic mapping.

Software Testing activities are supported by a well-defined and controlled test process [3]. Testing process concerns to how tests can be conducted and managed. It involves phases, activities, artifacts, techniques, procedures, resources and tools that seek to control and organize tests, in order to achieve high-quality software [2, 3, 8, 9].

As the software development process becomes more complex, testing process also becomes increasingly complex and prone to generate a lot of information. Such information may turn into useful knowledge to potentially benefit future projects from experiences gained from previous projects [4]. However, converting this information into applicable knowledge is not an easy task. There is a need to properly represent and process the knowledge so that it can be accessible and manageable. In this context, Knowledge Management (KM) principles can be applied.

Different KM approaches have been applied in the context of software testing to promote reuse of knowledge generated in the testing process. Given this context, we conducted a systematic mapping of the literature aiming at synthesizing the evidences related to KM in software testing.

III. RESEARCH PROTOCOL

The research method for this systematic mapping was defined based on the guidelines for systematic literature reviews given in [7]. A systematic mapping helps providing a wide overview of a research area and identifying areas suitable for conducting Systematic Literature Reviews and areas where a primary study is more appropriate. It involves three main phases [7]: (i) **Planning**: refers to the pre-review activities, and aims at establishing a review protocol defining the research questions, inclusion and exclusion criteria, sources of studies, search string, and mapping procedures; (ii) **Conducting:** regards searching and selecting the studies, in order to extract and synthesize data from them; (iii) **Reporting:** is the final phase and aims at writing up the results and circulating them to potentially interested parties. Following, the main parts of the mapping protocol used in this work are presented.

A. Research Questions

This mapping aims at answering the following research questions:

RQ1. What are the problems reported by software organizations related to knowledge about software testing?

RQ2. What are the purposes of employing KM in software testing?

RQ3. What are the types of knowledge items typically managed in the context of software testing?

RQ4. What are the main conclusions (benefits and problems) reported on the implementation of KM initiatives in software testing?

RQ5. What are the mechanisms or technologies used to provide KM in software testing?

B. Inclusion and Exclusion Criteria

The selection criteria are organized in one inclusion criterion (IC) and five exclusion criteria (EC). The inclusion criterion is: (IC1) The study discusses KM applied to software testing. The exclusion criteria are: (EC1) The study does not have an abstract; (EC2) The study is just published as an abstract; (EC3) The study is not written in English; (EC4) The study is an older version (less updated) of another study already considered; and (EC5) The study is not a primary study, such as editorials, summaries of keynotes, workshops, and tutorials.

C. Sources

The search was applied in seven electronic databases that were considered the most relevant according to [10]. They are:

IEEE Xplore (*http://ieeexplore.ieee.org*)
ACM Digital Library (*http://dl.acm.org*)
SpringerLink (*http://www.springerlink.com*)
Scopus (*http://www.scopus.com*)
Science Direct (*http://www.sciencedirect.com*)
Compendex (http://www.engineeringvillage2.org)
ISI of Knowledge *(http://www.isiknowledge.com)*

D. Keywords and Search String

The search string considered two areas, Software Testing and KM (Table I), and it was applied in three metadata fields (title, abstract and keywords). The search went through syntactic adaptations according to particularities of each source.

TABLE I. KEYWORDS SEARCH

Areas	***Keywords***
Software Testing	"Software Testing", "Software Test"
KM	"Knowledge Management", "Knowledge Reuse"
Search string: (*"Software Testing" OR "Software Test") AND ("Knowledge Management" OR "Knowledge Reuse")*	

E. Data storage

The publications returned in the searching phase were cataloged and stored appropriately. This catalog helped us in the classification and analysis procedures.

F. Assessments

Before conducting the mapping, we tested the mapping protocol. This test was conducted in order to verify its feasibility and adequacy, based on a pre-selected set of studies considered relevant to our investigation. The review process was conducted by one of the authors and the other two carried out its validation. They analyzed 36% of the studies using two different samples.

IV. CONDUCTING THE MAPPING

In this section, the main steps that we performed in this mapping are discussed, namely: search and selection, data extraction and classification, and synthesis and data analysis.

A. Search and Selection

In the search process, we considered the studies published until January 2013. As a result, a total of 336 publications were returned, out of which 53 from **IEEE Xplore**, 67 from **Compendex**, 70 from **Scopus**, 2 from **Science Direct**, 4 from **ACM Digital Library**, 134 from **SpringerLink** and 6 from **Thomson Reuters Web of Knowledge**.

Then, a selection process, divided into 3 stages, was applied on the returned publications. In the first stage duplicates were eliminated based on examining title and abstract. In this step, the number of publications was reduced to 253 (approximately 25% reduction), since many publications were available in more than one source.

In the second step, inclusion and exclusion criteria were applied considering title and abstract. 219 publications (86.5%) were eliminated. Although the publications cite, in the abstract, the terms contained in the search string, they did not have the principles of KM applied in the area of software testing and thus were eliminated by the inclusion criterion (IC1). Finally, in the third phase, the exclusion criteria were applied considering the entire text, resulting in a reduction of 70.5%. It is worth pointing out that, in the third stage, one publication was eliminated because we did not have access to the full text.

From the three stages of the selection process, 10 studies were considered relevant, from which data were extracted. Table II summarizes the stages and their results. It shows the progressive reduction of the number of studies throughout the selection process. 10 out of 336 was the final number, with a reduction rate of about 97%. Table III lists the 10 studies considered relevant.

TABLE II. RESULTS OF THE SELECTION PROCESS STAGES

Stage	Criteria	Analyzed Content	Initial N. of Studies	Final N. of Studies	Reduction (%)
1st	Eliminating duplication	Title and abstract	336	253	25%
2nd	IC1, EC1, EC2, EC3, EC4 e EC5	Title and abstract	253	34	86.5%
3rd	IC1, EC4, EC5 e EC6	Entire Text	34	10	70.5%

B. Data Extraction and Classification

To answer the research questions from the 10 selected studies, we used a form containing some parameters, including the following: id, bibliographic reference, problems, purpose, types of knowledge, and benefits and problems, related with the implementation of KM in software testing. This form was used to extract the answers. Therefore, before the extraction, categories for classifying the studies were defined according to the research questions. So, depending on the focus of each category item, the study was classified as one or any combination of this. Categories were defined as follows.

Classification schema for problems: this is based on the main problems related to knowledge about software testing. We have identified five main categories of problems, namely: (i) Barriers in transferring testing knowledge, (ii) Loss of testing knowledge, (iii) Low reuse rate of testing knowledge, (iv) Testing knowledge is not properly shared, and (v) Testing knowledge is not properly considered for planning the testing process (including human resource allocation to testing activities).

TABLE III. SELECTED STUDIES

ID	Bibliographic reference
#1	Y. Liu, J. Wu, X. Liu, G. Gu "Investigation of Knowledge Management Methods in Software Testing Process," International Conference on Information Technology and Computer Science, v.2, pp. 90 – 94, 2009.
#2	O. K. Wei, T. M. Ying, "Knowledge Management Approach in Mobile Software System Testing," Industrial Engineering and Engineering Management, pp. 2120 - 2123, 2007.
#3	L. Xu-Xiang, Z. Wen-Ning, "The PDCA-based software testing improvement framework," Apperceiving Computing and Intelligence Analysis (ICACIA), pp. 490 - 494, 2010.
#4	R. Abdullah, Z. D. Eri, A. M. Talib, "A Model of Knowledge Management System in Managing Knowledge of Software Testing Environment," Malaysian Conference in Software Engineering (MySEC), pp. 229 – 233, 2011.
#5	X. Li, W. Zhang, "Ontology-based Testing Platform for Reusing," Sixth International Conference on Internet Computing for Science and Engineering, pp. 86 – 89, 2012.
#6	A. Desai, S. Shah, "Knowledge Management and Software Testing," International Conference and Workshop on Emerging Trends in Technology (ICWET), pp. 767-770, 2011.
#7	E. Collins, A. Dias-Neto, V. F. Lucena, "Strategies for Agile Software Testing Automation: An Industrial Experience," 36th Annual Computer Software and Applications Conference Workshops, pp. 440-445, 2012.
#8	J. Andrade, J. Ares, M. Martínez, J. Pazos, S. Rodríguez, J. Romera, S. Suárez, "An architectural model for software testing lesson learned systems," Information and Software Technology, pp. 18-34, 2013.
#9	K. Karhu, O. Taipale, K. Smolander, "Investigating the relationship between schedules and knowledge transfer in software testing," Information and Software Technology, Vol. 51, pp. 663-677, 2009.
#10	K. Nogeste and D. H.T. Walker, "Using knowledge management to revise software-testing processes," Journal of Workplace Learning, v.18, n.1, pp. 6-27, 2006

Classification schema for purposes: we wanted to know what are the organizations' purposes, when employing KM in software testing. We have identified five main categories of purposes: (i) Reuse of knowledge related to software testing (including lessons learned), (ii) Support for decision making, (iii) Cost reduction, (iv) Competitive advantages, and (v) Organizational learning (including also lessons learned).

Classification schema for types of knowledge: This schema shows what types of knowledge are dealt with by organizations and how they are handled. In this case, we analyzed the explicit and tacit knowledge generated by an organization in the context of software testing. Tacit knowledge comes from individual experiences. It is highly personal, hard to formalize and, therefore, difficult to communicate to others. On the other hand, explicit knowledge is formal and systematic. For this reason, it can be easily communicated and shared. It can be expressed as tables, figures, drawings, sketches, diagrams and requirements [11].

Other classification schemes: In these schemes we collected unstructured data without a predefined classification. We looked at the main findings found as benefits and problems related with the implementation of KM in software testing, and the main mechanisms and technologies reported by the selected studies.

C. Synthesis and Data Analysis

Publications over the years: In order to offer a general view of efforts in the area of KM in Software Testing, a distribution of the 10 selected papers over the years is shown in Figure 1. As this figure suggests, KM in Software Testing is very recent, occurring basically from 2006 to nowadays.

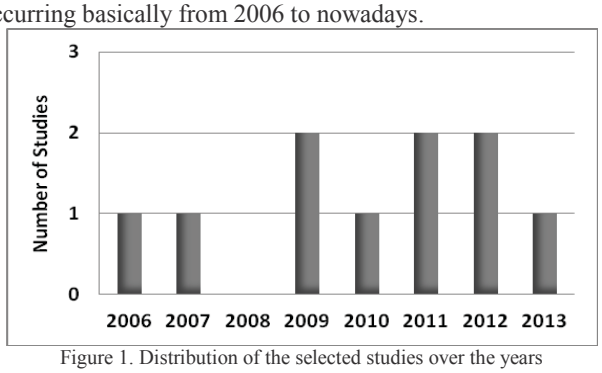


Figure 1. Distribution of the selected studies over the years

Main problems related to knowledge about software testing (RQ1): Figure 2 shows the percentage of studies per category, considering the problems reported by software organizations related to knowledge about software testing. We can notice that "Barriers in Knowledge Transfer" has the largest representativeness (9 studies in 10, corresponding to 90%). It stands out because transfer of organizational knowledge can be quite difficult to achieve. This occurs because most of the knowledge in organizations is tacit, that is, derived from experience, and it becomes difficult to articulate. Another category with a high percentage is "Low Reuse Rate Knowledge" with 60% (6 studies). Software Testing, in general, can involve reusing modules, test cases, components, and experiences. However, testing teams, generally, do not reuse or take advantage on the knowledge acquired or the experience gained. Therefore, the same mistakes are repeated, even though there are individuals in the organization with the knowledge and experience required to stop this [4].

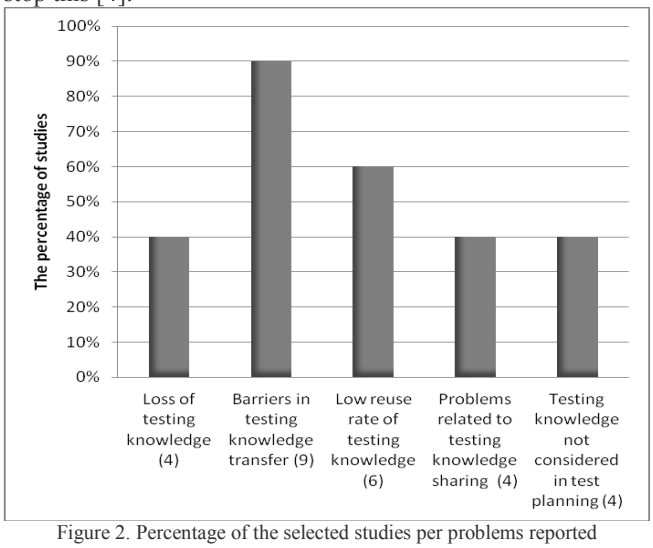


Figure 2. Percentage of the selected studies per problems reported

Main purposes to employ KM in software testing (RQ2): Figure 3 shows the percentage of studies per category, considering the organizations' purposes in managing software testing knowledge. We can notice that "Knowledge Reuse" (10 studies – 100%), "Organizational Learning" (7 studies – 70%) and "Competitive Advantages" (6 studies – 60%) have the largest representativeness. We should highlight that some purposes identified are strongly related. For instance, lessons learned are both a way to promote knowledge reuse and organizational learning. Thus, studies reporting that one of the purposes of applying KM in software testing is registering and disseminating lessons learned (5 studies – 50%) were considered in both categories. Knowledge reuse, in turn, helps increasing test effectiveness and thus leads to competitive advantages and cost reduction.

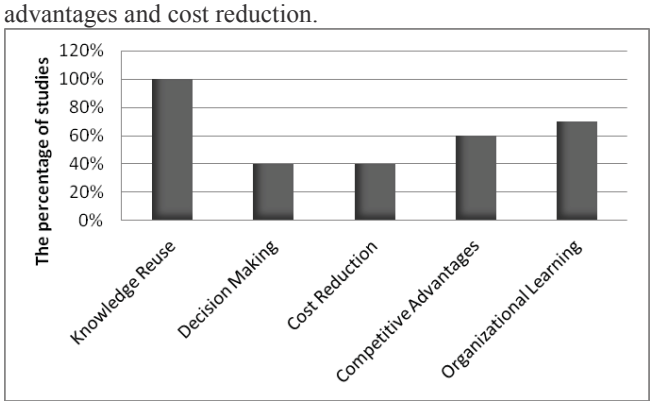


Figure 3. Percentage of the selected studies per purposes reported

Types of knowledge typically managed (RQ3): Knowledge can be of two main types: tacit and explicit knowledge. In the 10 selected studies, both of them are considered. Tacit knowledge is taken into account in all studies (100%), whereas 7 studies (70%) consider also explicit knowledge. Explicit knowledge appears mainly as test artifacts. Some examples of explicit knowledge are: Test Plan, Test Cases, Test Results, Requirements Specification, and Conceptual Models. Out of all these examples, Test Cases are the most common cited artifact in most of the literature evaluated.

Most of the studies identify that tacit knowledge is more difficult to acquire, as part of personal experiences by the members of the test team. They also mention that tacit knowledge can be acquired from discussions, experiences from project members, questionnaires and communications.

Main conclusions (benefits and problems) reported on the implementation of KM initiatives in software testing (RQ4): Regarding this issue, we have to highlight that 4 studies, although discussing some aspects related to KM in software testing, they do not report KM initiatives in software testing. In fact, only six of them (studies #1, #2, #3, #5, #8, and #10 in Table III) discuss KM initiatives in software testing. From these studies, we identified the following conclusions from employing KM in software testing:

- **Major problems found:** (i) Employees are normally reluctant to share their knowledge; (ii) if, on top of this,

knowledge sharing increases the employee workload, KM strategies fail; and (iii) the existing communication systems are not appropriate.

- **Primary benefits found:** (i) Selection and application of better suited techniques, methods and test cases; (ii) Cost reduction; (iii) Increasing test effectiveness; and (iv) Competitive advantages.

Mechanisms or technologies used (RQ5): From the selected studies, 6 of them use a KM system (one using a general purpose one), and 4 present KM models or architectures devoted to KM in software testing. From the mechanisms and technologies applied by them, two highlight: yellow pages (or knowledge maps) are used in 4 studies (40%); and ontologies are used in 3 studies (30%).

V. RESULTS DISCUSSION

In this section we discuss some important findings and limitations of this mapping.

Several studies have reported the problem of knowledge reuse within organizations. The main issue is that knowledge is retained with a single individual and therefore becomes more difficult to raise this knowledge to the organizational level. Even when some knowledge management strategy is applied, it is not always feasible to achieve organizational learning because the employees are reluctant to share their knowledge as they feel that retaining this knowledge is an advantage over their colleagues [4].

Several studies report on the use of a KM system (#1, #2, #3, #5, #8, #10). Others propose knowledge management models or architectures (such as #1, #4, #5, #8). A KM system should support the integration of information from disparate sources, wherein a decision maker manipulates information that someone else conceptualized and represented. So, the KM system must minimize ambiguity and imprecision in interpreting shared information. This can be achieved by representing the shared information using ontologies [12]. Although ontologies have been widely recognized as an important technology for KM, only three studies (#1, #5, and #8) use ontologies. More specifically, only one (#5) uses an ontology of the software testing domain. This seems to be a problem, since, as pointed by Staab et al. [13], ontologies are the glue that binds KM activities together, allowing a content-oriented view of KM. Ontologies define shared vocabulary to be used in the KM systems to facilitate communication, integration, search, storage and knowledge representation [14].

With respect to the types of knowledge, both tacit and explicit have been investigated in the literature. According to [11], as expected, tacit knowledge is more valuable, it is hard to be acquired, and it requires good strategies to acquire and process this knowledge. However, results obtained from it are rich. Therefore, tacit knowledge is important to the test team as it refers to previous experiences and thorough analysis of past projects [15]. In this context, yellow pages or knowledge maps are considered important tools for managing testing knowledge, and the some studies cite their importance, such as #1, #5, and #8.

During the mapping it was possible to infer that much of the explicit knowledge was related to reuse of test cases derived from documents considered complete and correct. According to [16], more detailed information on test cases can provide a greater learning. As test cases evolve in applications, they may be changed for a variety of reasons. Thus an efficient and effective KM process can help in evaluating the impact and in conducting changes of the test cases.

As we can see by means of this mapping, there are many benefits of implementing KM in organizations for managing software testing knowledge:

- **Selection and application of better suited techniques, methods and test cases.** Experience plays a key role in testing, and managing past experience helps to effectively tailor the techniques and methods to the ongoing project. Some of these techniques, such as White-box Testing Techniques, Black-box Testing Techniques or Defect-based Testing Techniques [2], depend on the knowledge, experience and intuition of test analyst.
- **Cost reduction.** In the testing context, cost has a very strong relationship with time. Tester experience is crucial for designing test cases and regression test selection. A good selection of test cases minimizes not only costs but also reduces time [17].
- **Test effectiveness increase.** Knowledge and experience about the domain and the system under test is essential for increasing test effectiveness. This helps testers improve decision making on which techniques to use, selection of test cases or approaches for test input generation [4].
- **Competitive advantages.** In organizations, KM is now seen as a strategic factor and knowledge is also recognized as one of the main sources of cost savings and competitive advantage [18]. The ability to transfer best practices in the organization is a means to build competitive advantage through the appropriation from scarce knowledge [19].

Although KM in software testing brings many benefits, there are also problems, such as:

- **Employees are normally reluctant to share their knowledge:** Many experiences are grasped by only a few people and haven't become public knowledge. This causes many difficulties in knowledge transfer about testing [4].
- **Increased workload:** Shortage of time is a potential risk to incorporate the principles of KM in software testing, because knowledge sharing can imply in increasing the employee workload and costs [4, 18].

- **KM systems are not appropriate yet:** There are many difficulties in implementing knowledge acquisition, coding, storage and searching functionalities effectively in KM system, because it involves all the problems mentioned above as time and interest of the employees.

The mapping conducted in this study also had some limitations. In order to reduce subjectivity, the other two authors made a random validation of 36% of the studies. We did several tests with the search string to try our best not to compromise the return of some preliminary studies. We cannot say what is the best technique applied, but the objective was to map how the principles of KM has evolved in the software testing domain over the years. Furthermore, the results may be different if conducted to another area of application different from the testing software.

VI. CONCLUSIONS

This paper presented a systematic mapping in the context of software testing and KM. Five research questions were defined and addressed investigating the following aspects: (i) main problems found related to knowledge in software testing; (ii) purposes to employ KM in software testing; (iii) types of knowledge typically managed in the context of software testing; (iv) main conclusions (benefits and problems) reported on the implementation of KM initiatives; (v) mechanisms or technologies used in KM in software testing.

The contributions of this work are on making evident some aspects associated to the employment of KM in software testing and research efforts that can drive future research. In this context, we highlight the following conclusions: (i) the major problem in organizations are barriers in knowledge transfer with largest representativeness; (ii) reuse of testing knowledge is the main purpose of applying KM in software testing; (iii) there is a great concern with tacit knowledge.

Implementation of KM strategies in the field of software testing has shown very promising research, since KM helps in handling knowledge within the organization in several respects as shown in this systematic mapping. However, a point seems to be a challenge for KM in software testing. Although recognized as an important instrument by the KM community [12, 13, 14], ontologies are not being widely used in KM initiatives in software testing. Thus, as future work, we intend to explore how ontologies can be used for managing knowledge in the software testing domain.

ACKNOWLEDGMENT

The first author would like to acknowledge FAPESP (Process: 2010/20557-1) for the financial grant. The second author acknowledges FAPES (Process Number 52272362/11) for the financial grant.

REFERENCES

[1] IEEE Std 1012-2004: IEEE Standard for Software Verification and Validation. New York, NY, USA. pp. 120, 2004.

[2] A. P. Mathur, Foundations of Software Testing. 5rd ed. Delhi, India: Dorling Kindersley (India), Pearson Education in South Asia, 2012.

[3] IEEE Computer Society, SWEBOK, A Guide to the Software Engineering Body of Knowledge, 2004.

[4] J. Andrade, J. Ares, M. Martínez, J. Pazos, S. Rodríguez, J. Romera, S. Suárez, "An architectural model for software testing lesson learned systems," Information and Software Technology, pp 18-34, 2013.

[5] A. Desa, S. Shah, "Knowledge Management and Software Testing," in International Conference and Workshop on Emerging Trends in Technology (ICWET 2011) – TCET, Mumbai, India, 2011.

[6] D. O'Leary and R. Studer, "Knowledge management: An interdisciplinary approach," IEEE, vol. 16, No. 1, 2001.

[7] B, Kitchenham, S. Charters, "Guidelines for performing Systematic Literature Reviews in Software Engineering," School of Computer Science and Mathematics Keele University and Departament of Computer Science University of Durham, UK, v. 2.3, 2007.

[8] G. J. Myers, The Art of Software Testing. 2rd ed. Canada: John Wiley and Sons, 2004.

[9] R. Black, J. L. Mitchell. Advanced Software Testing: guide to the ISTQB advanced certification as an advanced technical test analyst. USA:Oreilly & Assoc, 2008.

[10] T. Dyba, T. Dingsoyr, G. Hanssen, "Applying systematic reviews to diverse study types: An experience report," First International Symposium on Empirical Software Engineering and Measurement, Madrid, pp. 225-234, 2007.

[11] I. Nonaka, H. Takeuchi, The Knowledge-Creating Company. How Japanese Companies Create the Dynamics of Innovation, Oxford University Press, Oxford, 1999.

[12] H. M. Kim, "Developing Ontologies to Enable Knowledge Management: Integrating Business Process and Data Driven Approaches," Workshop on Bringing Knowledge to Business Processes, 2000.

[13] S. Staab, R. Studer, H. P. Schurr, and Y. Sure, "Knowledge Processes and Ontologies," IEEE Intelligent Systems, vol. 16, No. 1, 2001.

[14] V. R. Benjamins, D. Fensel, and A. G. Pérez, "Knowledge Management through Ontologies," The 2nd International Conference on Practical Aspects of Knowledge Management (PAKM98), Switzerland, 1998.

[15] K. W. Ong, M.Y. Tang, "Knowledge management approach in mobile software system testing," in Proceedings of the IEEE International Conference on Industrial Engineering and Engineering Management, IEEE IEEM 2007, Singapore, pp. 2120–2123, 2007.

[16] X. Li, W. Zhang, "Ontology-based Testing Platform for Reusing," Sixth International Conference on Internet Computing for Science and Engineering, pp. 86- 89, 2012.

[17] A. Beer, R. Ramler, "The role of experience in software testing practice," in Proceedings of the 34th Euromicro Conference Software Engineering and Advanced Applications, Italy, pp. 258–265, 2008.

[18] O. Taipale, K. Karhu, K. Smolander. "Observing Software Testing Practice from the Viewpoint of Organizations and Knowledge Management," In Empirical Software Engineering and Measurement (ESEM), 2007. pp. 21-30, 2007.

[19] K. Karhu, O. Taipale, K. Smolander, "Investigating the relationship between schedules and knowledge transfer in software testing," Information and Software Technology, Vol. 51, pp. 663-677, 2009.

Improving Architectural Knowledge Management in Public Sector Organizations – an Interview Study

Dan Tofan
University of Groningen
Groningen, Netherlands
d.c.tofan@rug.nl

Matthias Galster
University of Canterbury
Christchurch, New Zealand
mgalster@ieee.org

Paris Avgeriou
University of Groningen
Groningen, Netherlands
paris@cs.rug.nl

***Abstract* — Architecting software systems is a knowledge-intensive activity. It requires significant knowledge about architecting in general, but also about domains and technologies. Such knowledge should be managed systematically to make it available throughout the whole software development cycle (e.g. to facilitate maintenance). Architectural knowledge management (AKM) literature covers organizations in the private sector (e.g. software vendors). However, there is a lack of studies on AKM practices in public sector organizations (e.g. municipalities), even though AKM practices in the public sector are immature. Therefore, we propose applying lessons from AKM practices found in the private sector to address AKM challenges in the public sector. Thus, we conducted an interview study with four public and four private sector organizations. We identified challenges for AKM in the public sector. Then, we derived solutions from the private sector to the challenges in the public sector. The main challenges in the public sector are vaporization of architectural knowledge, insufficient knowledge sharing, and organizational cultures that do not encourage AKM. Solutions to these challenges include community building, improved tool support, quality control and management support. The results help improve AKM practices in the public sector.**

Keywords – software architecture; knowledge management; public sector; private sector

I. INTRODUCTION

Within software developing organizations, architectural knowledge is considered an important asset that needs to be managed systematically [1]. Software architects make important decisions. Making architectural decisions is a knowledge-intensive task. For example, deciding on the decomposition of a system requires experience, domain-specific knowledge, knowledge about technologies, and general software architecture knowledge. The software architecture is thus the result of early design decisions with regard to a software system. Typical examples of architectural decisions include choosing a development framework (e.g. J2EE, .NET), selecting architectural patterns (e.g. client-server, layers), or deciding on the middleware for a distributed software system (e.g. an enterprise service bus) [2].

Architectural knowledge management (AKM) increases the quality of software products by creating, capturing, and sharing knowledge among architects, developers and other stakeholders [1]. AKM can also improve the overall software development process [1], mainly by reducing architectural knowledge (AK) vaporization [3]. AK vaporization happens when knowledge about critical design decisions is lost. AK vaporization increases maintenance costs as stakeholders miss the rationale of previous decisions later on in a project (e.g. during maintenance) [3]. Therefore, the field of software architecture knowledge management has seen increased attention in recent years [1, 4].

Most work on AKM has been conducted in the context of private sector organizations (e.g. commercial software vendors or companies that develop products that rely heavily on software). Unfortunately, AKM in public organizations has neither been studied, nor understood well enough to propose solutions to its AKM challenges.

Architects in private sector organizations are software or enterprise architects. Organizations in the private sector are not owned or operated by a government. Typical private sector organizations are corporations, regardless of their size. In contrast to private sector organizations, public sector organizations are owned and operated by some government. Typical public sector organizations are municipalities or government agencies. Architects in public sector organizations are usually enterprise architects. Public sector architects are involved in e-government projects, which offer services to citizens, private sector organizations, and other public sector organizations.

From our previous work on service-oriented architectures in e-government [5], we learnt that AKM in the public sector needs improvement. For example, immature AKM leads to constraints on designing specialized reference architectures for municipalities [5]. Additionally, similar to the private sector, e-government projects in public sector organizations are under pressure to reduce costs. As shown for the public sector, AKM helps reduce costs. However, we could not find literature on AKM in the public sector. Therefore, the goal of this study is to understand AKM in the public sector and to provide solutions for improving AKM in the public sector. Towards this goal, we formulate the following research question**: What are potential solutions to the challenges for AKM in public sector organizations?**

To answer this research question, we conducted an interview study in public and private sector organizations, with the purpose of improving AKM practices in the public sector. The first step towards improving AKM in the public sector is to understand AKM challenges in the public sector. Then, we use AKM solutions from the private sector to address AKM challenges in the public sector. Proposing solutions for improving knowledge management practices in the public sector by using practices from the private sector has already been applied successfully [6, 7].

The main contribution of this paper is a set of AKM challenges in the public sector, mapped to challenges and solutions found in the private sector. Researchers can use the

results to propose further improvements to AKM practices in public sector organizations. Practitioners can apply the solutions to AKM challenges, to improve current AKM practices in public sector organizations.

II. Related Work

This paper is related to three research areas: knowledge management (KM) in software engineering, AKM, and KM in the public sector. We discuss related work from each area.

Dingsøyr and Conradi [8] analyzed eight case studies of KM in software engineering. All cases reported benefits due to KM, such as time savings. However, results from a systematic literature review on KM in software engineering indicate that most existing work consists of informal lessons learnt from applying KM, instead of scientific studies [9]. In contrast, we conducted an interview study to answer our research question in a scientific manner.

Various AKM challenges and solutions have been investigated in private sector organizations. For example, the challenge of architectural knowledge vaporization can be addressed by documenting design decisions [3]. Furthermore, the challenge of sharing architectural knowledge can be addressed by considering communication, planning issues, and quality of captured knowledge [1, 4] when implementing AKM strategies. Finally, a delicate balance must exist between sharing architectural knowledge through documentation and social interactions [6] to ensure that knowledge is made explicit, without causing much burden on architects.

The idea of getting inspiration from the private sector for improvements in the public sector has been used before. Bate and Robert [6] describe how knowledge management concepts and practices from the private sector can improve health care organizations in the UK public sector. Another study compares public and private sector perceptions and the use of knowledge management [7]. In both types of organizations, improved quality and efficiency were the main benefits of knowledge management.

Overall, many reports exist on AKM in the private sector (e.g. [1, 4]), as well as on general knowledge management in the public sector (e.g. [6, 7]). However, we could not find any work on AKM in the public sector.

III. Research Method

To answer the research question in Section I, we conducted an interview study in public and private sector organizations, using semi-structured interviews. Such interviews belong to qualitative research, which aims at investigating and understanding phenomena within their real life context [10]. Challenges and solutions for AKM are linked tightly to their context. Also, we needed flexibility during the interviews, so that we could ask new questions, to further probe for AKM challenges and solutions.

Similar to [11], we decided to conduct extended, semi-structured interviews. Using surveys was less optimal, because of the lack of reports on AKM practices in public sector organizations, which inhibits the development of relevant questionnaires. Additionally, in a survey, participants might have different interpretations of the questions. Therefore, we decided to conduct semi-structured interviews, which enabled us to present our topics of interest, and discuss them directly with the participants. Furthermore, semi-structured interviews are useful as preliminary work for an in-depth case study [10]. However, semi-structured interviews require significant effort to prepare a discussion plan, recruit participants, and conduct the interview sessions. Overall, semi-structured interviews suited best our research goal, given the lack of previous work on AKM in the public sector.

A. Data Collection and Analysis Procedures

To conduct the interviews, we selected organizations from the private and public sectors which had enterprise or software architects. We contacted diverse organizations from our collaboration network. For the interviews, we used recommendations from [12] to ensure that the interviewer has the needed skills, and to facilitate good interaction between interviewer and interviewees. In each organization, we interviewed one or two persons, depending on their availability. In total, we interviewed eleven persons. We conducted face-to-face interviews that typically lasted one hour. The interviews took place between January 2010 and July 2012. We made audio records for the interviews, with the permission of the interviewed persons. We used a discussion plan with open-ended questions structured around three areas: *strategy* (e.g. "what are the objectives of the AKM strategy?"), *processes* (e.g. "what are the processes for sharing AK?"), and *tools* (e.g. "what tools are used for AKM?"). We derived these areas from AKM literature [1, 4].

To analyze the interviews, we transcribed the audio recordings. Next, two researchers performed content analysis, by assigning individually codes to sentences, phrases or paragraphs [10]. Each code corresponded to either a challenge or solution for managing AK. Different codes could be assigned to the same piece of content. Afterwards, researchers discussed their differences, and they agreed on a common interpretation. In case of disagreements, we consulted a third researcher. Data analysis also included a mapping of challenges to solutions by identifying which challenges were addressed by which solutions.

B. Organizations

The organizations that took part in this study are listed in TABLE I. Only the software architect in PS1 had about five years of practical experience. All the other participants had at least ten years of practical experience. The private sector organizations are international corporations. The public sector organizations are part of Dutch government. For confidentiality reasons, we provide limited details on the organizations, and we assign aliases to them.

TABLE I. Summary of participating organizations

ID	Sector	Domain	# Employees	Interview with
Gov1	Public	Municipality	~1.000	Enterprise architect KM consultant
Gov2	Public	Municipality	~100	Enterprise architect
Gov3	Public	Agency	~1.300	Software architect Software architect
Gov4	Public	Ministry	~30.000	Enterprise architect
PS1	Private	Software provider	~600	KM director Software architect
PS2	Private	IT consultancy	~40.000	Enterprise architect
PS3	Private	Engineering	>100.000	Enterprise architect
PS4	Private	IT consultancy	>100.000	Software architect

C. Validity Threats

We discuss validity threats using the recommendations from [13], in line with a report that uses the same methodology conducted by [11]. *Construct validity* refers to the relation between the observations and the theory behind the research. We interviewed many practitioners to avoid mono-operation bias [13]. We avoided evaluation apprehension [13] by using the recommendations from [12] to create a comfortable and nonjudgmental atmosphere for the interviews, and ensuring their confidentiality. *Conclusion validity* refers to the accuracy of the study conclusions. To increase conclusion validity, we involved more researchers in the data analysis, who reached a high agreement when interpreting the data. *External validity* refers to how well the results can be generalized beyond the study. To increase external validity and to reduce validity threats, we conducted interviews at a variety of organizations in the public and private sectors. Besides architects, we also interviewed knowledge management consultants, who could offer insights on how architectural knowledge is managed. *Internal validity* threats are not applicable to this study, because we do not try to establish any causal relationships.

IV. CHALLENGES

We identified three common challenges for the public and the private sector, as well as a challenge only for the private sector. Additionally, we link these challenges to results from knowledge management literature. We summarize these challenges in TABLE II. Afterwards, we present details on all challenges, their consequences, and concrete examples from the public and private sectors.

TABLE II. CHALLENGES IN PUBLIC AND PRIVATE SECTOR ORGANIZATIONS.

Challenge	Public sector	Private sector
AK vaporization	Gov1, Gov2, Gov3, Gov4	PS1, PS2, PS3, PS4
Low AK sharing	Gov1, Gov2	PS1, PS2, PS3, PS4
Organizational culture	Gov1, Gov2, Gov3	PS1, PS2, PS4
Low integration	-	PS1

A. Challenges in the public sector

1) AK Vaporization: This challenge refers to the loss of architectural knowledge in an organization [3]. We learnt that AK vaporization contributes to increased vendor lock-in because the less in-house AK remains in public sector organizations, the more they depend on software vendors for technology decisions (e.g. extending existing software depends on one vendor). Also, AK vaporization makes it more difficult to modify the architecture without involving vendors. For example, migrating existing systems to a service-oriented architecture depends on the willingness of the vendors. Having more in-house AK enables organizations to make better decisions about software solutions that meet their core needs, and to decrease vendor lock-in. Overall, AK vaporization reduces flexibility for public sector organizations and increases maintenance costs.

AK vaporization is a challenge across all public sector organizations that we studied. In Gov3, little architectural knowledge was captured on a regular basis. Architects had no formalized way to capture their knowledge. A wiki was used in the past, but only for a brief period, so the content became quickly outdated. Consequences of AK vaporization were that similar problems were solved in different ways. Thus, new people who joined a team needed to re-discover solutions, instead of reusing a proven solution. Instead of reusing captured knowledge, much informal communication of knowledge needed to take place. Architects often needed to explain the same solution to more developers, instead of documenting a solution and sharing the documentation.

Similar to the other organizations, little architectural knowledge was captured in Gov4. The architects working for Gov4 were employed through external companies, and were not asked to document their knowledge, although they were willing to do it. Moreover, little architectural knowledge existed inside Gov4 to facilitate knowledge sharing through direct interactions. Therefore, when the external architects stopped working for Gov4, their knowledge vaporized from Gov4, because there was no mechanism for preserving it.

2) Low AK Sharing: This challenge refers to insufficient sharing of architectural knowledge, inside and across organizations [1]. We learnt that low AK sharing existed across Gov1 and Gov2. An architect from Gov1 compared his current position with his previous job in the private sector, where co-workers were much more open to knowledge sharing, resulting in higher efficiency, by helping each other.

At Gov3, architects worked in small, isolated groups, without sharing much knowledge across groups. Also, architects could allocate parts of their time to increase their knowledge, but not for sharing it with others. In Gov4, the same tendency for isolation between groups existed, with little knowledge sharing between them. Moreover, in Gov4 most architects were from external companies, and very few knowledgeable people existed in Gov4, so architects could not share their knowledge with them. Overall, low AK sharing caused inefficiencies.

3) Lack of Supportive Organizational Culture: Culture contains norms about who controls what knowledge, and who can share or hoard it [14]. For example, a cultural norm is accepting knowledge hoarding as a source of job security or power [14]. An architect from Gov1 stated: *"Nearby municipalities are very small compared to us, maybe they fear we are going to take over things from them. That's the kind of feeling, which is very old."* Such fears encouraged knowledge hoarding and reduced knowledge sharing.

An architect at Gov3 considered that organizational culture played a role in a previous failed attempt to use a wiki for knowledge sharing between architects and developers. However, there were no accepted norms in Gov3 to capture and share knowledge, so the wiki content became gradually outdated, and was abandoned. Overall, we noticed that the lack of a supportive organizational culture increases knowledge vaporization and leads to reduced knowledge sharing, within and across organizations.

B. Challenges in the private sector

The challenges in the private sector match the ones from the public sector and include one extra challenge, namely low integration of AKM with organizational goals.

1) AK Vaporization: We found this challenge in all the private sector organizations. Architects mentioned several factors that contribute to this challenge. First, due to lack of time, less knowledge can be documented (PS1, PS2, and PS3). Second, documentation becomes irrelevant a few years after writing it, so the return for spending much time documenting is unclear (PS1, PS2, and PS4). The architect at PS2 summarized his view on documenting AK: "*We typically document when either the client asks for it or we discover that we need it. I'm not really interested in this documentation, unless I discover that the speed by which I can address a problem depends on the documentation.*" Third, the differences in educational background between software architects and maintainers increased the documentations costs. The architect at PS2 described this as follows: "*I have a designer, who has knowledge, puts it into a document, and pass it to someone who does maintenance, and who reads that information, generates knowledge from it, and these two do not match. Why not? Well, this one has architectural schooling for eight years and this one is good at programming routers. The points of view are so different, that these simply do not match, even if the documentation is the same.*" Forth, existing research results on capturing architectural design decisions are not fully adopted in industry (PS1, PS2, and PS3). Overall, similar to public organizations, AK vaporization lead to increased maintenance costs.

2) Low AK Sharing: This challenge exists in all the private sector organizations. From the interviews at PS1, we learnt that a factor contributing to this challenge was sharing knowledge by e-mails, because senders determined receivers of its content. This created an obstacle for other persons that might be interested in the knowledge captured by e-mail. For example, let us assume the rationale for an architectural decision is in an e-mail thread among a few architects. If a developer working on the code is interested in the rationale for that decision, then he would need to find out that the e-mail thread exists, and then ask one of the architects to forward it to him. Reducing overhead from these steps may facilitate AK sharing.

3) Lack of Supportive Organizational Culture: We identified this challenge in the interviews at PS1, PS2, and PS4. Several factors contributed to this challenge. First, architects and developers needed to be convinced to deliver not only source code, but also their knowledge. For example, at PS2, architects were not interested in transferring knowledge, because they do not consider it an interesting activity. Second, trust was an important factor in organizational culture, as put by the interview at PS1: "*It's not about software. It's not about wiki content, it's about people getting trust and solving problems*."

4) Low Integration with Organizational Goals: This challenge refers to the integration of knowledge management efforts with the goals of the organization [15]. From the interviews at PS1, we learnt that if such integration is low, then AKM efforts carry the risk of adding too little value to the organization. Specifically, the challenge is to provide value from AKM efforts throughout the lifecycle of projects for customers, i.e., from sales, to architecting, development, and during maintenance. AKM efforts need to show benefits, such as time savings for architects and other stakeholders.

Although the integration challenge did not emerge from the interviews in the public sector organizations, we considers this challenge is also relevant to public sector organizations, because such integration is a critical element of knowledge management, regardless of the type of organization [15]. Due to their different nature, the organizational goals in the public sector differ from the goals in the private sector. However, in both types of organizations, AKM efforts must serve organizational goals.

V. SOLUTIONS

We describe six solutions to the challenges in Section IV, elicited from the interviews in the private sector organizations: *community building*, *tool support*, *training*, *resources allocation*, *quality control*, and *management support*. Next, we present details about each solution.

1) Community Building: This solution was described in all private sector organizations. PS1 built its community, based on three elements: people, tools, and processes. People include architects, developers, testers, partners, and customers, who joined the community voluntarily and gradually. The main tool is a commercial wiki. Processes are managed through PS1's own business process management tool. For example, architects follow predefined processes for capturing knowledge regularly in the company wiki. If an architect leaves, the impact is reduced, because the other people in the organization can still use the architect's previous regular contributions to the wiki.

PS2 supports the creation of various communities of practice, in which architects can share knowledge with people in other positions or fellow architects. Moreover, collocating architects with other project groups improves AK sharing across projects. Architects who work in other groups *"get the feeling on what that really means and how that works."* Overall, getting perspectives from other groups helps architects deliver better documentation as architects became aware of the documentation needs of other groups.

Architects in PS3 share their knowledge through communities of practice, on architectural or other technical topics (such as Java or .Net), or business related topics. For these communities, the company organizes regular events to help networking, and promote knowledge sharing. Recognized experts are invited to share their insights at such events. The architect at PS3 stressed the idea that although tools help, they are less important than networks of people.

2) Tool Support: This solution receives much attention in all private sector organizations. At PS1, tool support shifted from a sender-dominant paradigm (e-mail) to a receiver-dominant paradigm (subscription). This means that notification about content and the actual content are separated. For example, instead of architects emailing content, they put architectural content in the wiki, and then send an e-mail notification with the wiki link. If a person considers that the content is interesting for her work, then the person can subscribe to the topic, and receive future notifications about it, without the constraint of receiving content through email.

Moreover, at PS1 knowledge capturing is based on a wiki, to avoid using different tools (e.g. forums, wikis, or document management systems). Having content in multiple locations creates obstacles for end users in accessing and sharing it. Therefore, all content must be delivered in the wiki. For example, if architects produce artifacts with other tools (e.g. PowerPoint slides), then the artifacts need to be attached to a wiki page.

At PS2 and PS3, various tools (e.g. SharePoint, wikis, internal blogs, and a third party collaborative software system) are used for capturing and sharing architectural knowledge. Additionally, social networking tools (e.g. Skype, Twitter, and Yammer) are widely used in PS2, PS3, and PS4, enabling knowledge exchanges across offices around the world.

3) Training: PS2 develops training materials for maintenance persons, to facilitate the transfer of architectural knowledge. In PS3, to increase peoples' AK, architectural training take place as part-time assignments, which may take from six to nine months. Although demanding, such trainings are necessary to ensure similar levels of AK throughout PS3. In addition, PS4 has central training facilities in which architects from various offices can meet in person during trainings, which leads to stronger connections through the social networking tools.

At PS3, in addition to trainings, there are company-wide events with software architecture experts. Architects can attend such events to expand their knowledge, or share their knowledge with each other.

4) Resources allocation: This solution refers to planning and allocating resources for AKM activities. At PS1 and PS3, 10% of architects' time is allocated for KM activities. At PS2, transferring architectural knowledge to maintenance people is considered a project by itself. As part of the project, architects need to consider what knowledge is needed for maintenance, and plan for its transfer. Architects may join temporarily the maintenance team to facilitate the transfer.

5) Quality Control: This solution refers to measures for increasing the quality of captured knowledge. At PS1, various metrics are collected for the wiki pages, such as number of visitors, profile of visitors, time spent on a page, or next visited pages. Such metrics indicate issues with content. If the content in the wiki is useful and up to date, then visitors perceive value in accessing the wiki.

At PS3, peer-review is used to evaluate the quality of captured AK. For example, a group of architects involved in a healthcare project sent some design documents to another group of experienced architects for review. The experienced architects provided constructive feedback to increase documentation quality. On the other hand, the reviewers (experienced architects) improved their knowledge on the healthcare domain.

At PS4, a solution to increase quality is to separate domain-specific knowledge from department-specific knowledge in the wiki system used for capturing knowledge. The rationale was that domains and departments evolve at different speeds. For example, a department might disappear during a re-organization, but knowledge from that department about the architecture of a specific system might be needed across other departments. If no separation exists, then the captured knowledge about that specific system becomes difficult to update, because it is mixed with irrelevant knowledge about the disappeared department.

6) Management Support: Support from top management was essential for the knowledge management efforts at PS1, because AKM is a long term effort. A person from PS1 summarized this in a metaphor: "*Grass doesn't grow by pulling it.*" PS1 needed two to three years to implement its new knowledge management practices. To sustain momentum for long-term knowledge management efforts, knowledge workers (including architects) needed to experience benefits from the new practices. This was mainly achieved by saving time through AK reuse.

Top management influences organizational culture by encouraging initiatives, and having tolerance for mistakes. This was described as a success factor at PS1: "*You'll only get fired if you didn't take initiative, not because you made a mistake. Otherwise I wouldn't be doing this. I wouldn't even be close to this kind of ideas [for knowledge management]*."

At PS4, management supported knowledge management efforts by providing positive reinforcements to the top wiki contributors who shared their knowledge. The positive reinforcements were in the form of emails from the top management thanking contributors, and internal news articles praising their efforts. By receiving recognition for their efforts, the organizational culture became more supportive for knowledge management activities. In turn, people became comfortable to share their knowledge and help colleagues.

VI. DISCUSSION

A similar study in the UK public sector (i.e. national healthcare) [6] describes knowledge management as a core activity for organizational improvements. Unfortunately, knowledge management in UK public sector is much more immature, compared to private sector organizations [6]. Therefore, the public sector can benefit from the lessons and experiences in the private sector [6].

In our study, we noticed a similar situation for the Dutch public sector. Although architectural knowledge management provides significant benefits, AKM in the public sector is much less mature than AKM in the private sector. For example, interviewees from the public sector mentioned previous failed attempts to use wikis for capturing and sharing knowledge. Therefore, we think that the experiences derived from the private sector will help improve AKM practices in the Dutch public sector and elsewhere. Similar to [6, 7], we consider that solutions from the private sector help improve the situation in the public sector. Also, the improved quality and efficiency that the private sector derives from its AKM efforts can motivate public sector organizations to pay more attention to AKM.

We summarize the solutions from the private sector (detailed in Section V) and map them to the challenges in the public sector (detailed in Section IV.IV.A) in TABLE III. Each solution exists in two or more private sector organizations, and addresses one or more challenges. For example, community building addresses the AK vaporization and sharing challenges. Also, tool support addresses AK vaporization, sharing and organizational culture challenges.

TABLE III. SUMMARY OF SOLUTIONS AND CHALLENGES.

Organizations	Solution	Challenges
PS1,PS2,PS3,PS4	Community building	vaporization, sharing
PS1,PS2,PS3,PS4	Tool support	vaporization, sharing, culture
PS2,PS3,PS4	Training	vaporization, sharing, integration
PS1,PS2,PS3	Resources allocation	vaporization, integration
PS1,PS3,PS4	Quality control	vaporization, sharing, integration
PS1,PS4	Management support	culture, integration, sharing

Dependencies among challenges have received little attention in AKM literature on the private sector. We noticed dependencies between AK sharing and AK vaporization: sharing reduces the risk of vaporization. On the other hand, addressing vaporization by creating architecture documentation makes it possible to share AK. Also, to address the lack of AK sharing and vaporization we can use a common set of solutions: trainings, processes, tools and building communities. Another dependency is that organizational culture influences the willingness of architects to share and capture their knowledge. For example, architects might not share their knowledge because there is no positive reinforcement in their organization for sharing. On the other hand, management support influences organizational culture, by providing the positive reinforcement and long-term focus. Both are needed to foster an organizational culture, which encourages knowledge-related activities.

This study also contributes to existing literature on AKM in the private sector. For example, various solutions have been proposed to address AK vaporization and sharing [1, 3]. However, little work exists on the role of organizational culture and the integration of AKM efforts with organizational goals. Results from KM literature [14, 15] and from this study encourage more research on these challenges that focuses on architectural knowledge.

Researchers can use these results to develop a taxonomy of AKM challenges and their solutions, for the public and private sectors. Such taxonomy would make explicit the relationships among challenges, among solutions, and between challenges and solutions. Additionally, practitioners can use the results of this study to improve AKM practices in the public sector. For example, the challenges indicate potential pitfalls when implementing AKM strategies in public sector organizations, so practitioners can define actionable AKM activities from the solutions, such as improving tool support, and securing management support.

VII. CONCLUSION AND FUTURE WORK

In this paper, we present the results of an interview study consisting of eleven interviews conducted over two years. We conducted the interviews in four public and four private sector organizations. The paper contributes to the existing body of work on AKM (e.g. [1, 2, 4]) with lessons learnt from implementing AKM in the private sector and proposes these as solutions to the challenges in the public sector. Also, our study confirms that AK vaporization and sharing as major challenges. Furthermore, we identified less studied challenges in the AKM literature: the role of organizational culture, and the integration of AKM with organizational goals. These challenges require further research to understand their role in AKM efforts in the public and private sectors. Finally, we think that dependencies among AKM challenges need further attention. Overall, researchers can use the results of this study to propose taxonomies of challenges and solutions for AKM in both public and private sectors. Practitioners can use this study to improve AKM practices in the public sector.

As future work, we will conduct workshops in the Dutch public sector with architects and their managers to improve AKM practices. Also, using insights from this study, we are developing tool support (https://github.com/danrg/RGT-tool/wiki) to help architects capture their knowledge.

ACKNOWLEDGMENT

We thank interview participants for their help. This research has been partially sponsored by the 'Software as a Service for the Varying Needs of Local e-Government' project, via contract no. 638.000.000.07N07.

REFERENCES

[1] M. A. Babar, T. Dingsøyr, P. Lago, and H. van Vliet, Software Architecture Knowledge Management: Theory and Practice. Springer Berlin, 2009.

[2] T. Dingsøyr and H. van Vliet, "Introduction to software architecture and knowledge management," in Software Architecture Knowledge Management, M. A. Babar, T. Dingsøyr, P. Lago, and H. van Vliet, Eds. Springer Berlin, 2009, pp. 1-17.

[3] A. Jansen and J. Bosch, "Software architecture as a set of architectural design decisions," in Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture, 2005, pp. 109-120.

[4] P. Avgeriou, P. Kruchten, P. Lago, P. Grisham, and D. Perry, "Architectural knowledge and rationale: issues, trends, challenges," ACM SIGSOFT Software Engineering Notes, vol. 32, no. 4, pp. 41-46, 2007.

[5] M. Galster, P. Avgeriou, and D. Tofan, "Constraints for the design of variability-intensive service-oriented reference architectures - An industrial case study," Information and Software Technology, vol. 55, no. 2, pp. 428-441, 2013.

[6] S. P. Bate and G. Robert, "Knowledge management and communities of practice in the private sector: lessons for modernizing the National Health Service in England and Wales," Public Administration, vol. 80, no. 4, pp. 643-663, 2002.

[7] R. McAdam and R. Reid, "A comparison of public and private sector perceptions and use of knowledge management," Journal of European Industrial Training, vol. 24, no. 6, pp. 317-329, 2000.

[8] T. Dingsøyr and R. Conradi, "A survey of case studies of the use of knowledge management in software engineering," Journal of Software Engineering and Knowledge Engineering, vol. 12, no. 4, pp. 391-414, 2002.

[9] F. O. Bjørnson and T. Dingsøyr, "Knowledge management in software engineering: A systematic review of studied concepts, findings and research methods used," Information and Software Technology, vol. 50, no. 11, pp. 1055-1068, 2008.

[10] C. B. Seaman, "Qualitative methods," in Guide to Advanced Empirical Software Engineering, F. Shull, J. Singer, and D. I. K. Sjøberg, Eds. Springer London, 2008, pp. 35-62.

[11] R. Berntsson Svensson, T. Gorschek, B. Regnell, R. Torkar, A. Shahrokni, and R. Feldt, "Quality Requirements in Industrial Practice - An Extended Interview Study at Eleven Companies," IEEE Transactions on Software Engineering, vol. 38, no. 4, pp. 923-935, 2012.

[12] S. E. Hove and B. Anda, "Experiences from conducting semi-structured interviews in empirical software engineering research," in Proceedings of the 11th IEEE International Software Metrics Symposium (METRICS'05), 2005, pp. 23-23.

[13] C. Wohlin, P. Runeson, and M. Höst, Experimentation in Software Engineering: an Introduction. Springer, 2000.

[14] D. W. D. Long and L. Fahey, "Diagnosing cultural barriers to knowledge management," The Academy of Management Executive (1993-2005), vol. 14, no. 4, pp. 113-127, 2000.

[15] B. Rubenstein-Montano, J. Liebowitz, J. Buchwalter, D. McCaw, B. Newman, and K. Rebeck, "A systems thinking framework for knowledge management," Decision Support Systems, vol. 31, no. 1, pp. 5-16, 2001.

Enhancing Deployment Requirements' Traceability via Knowledge Management Audit

Naomi Unkelos-Shpigel
Department of Information System, University of Haifa, Haifa, Israel
naomiu@haifa.ac.il

Irit Hadar
Department of Information System, University of Haifa, Haifa, Israel
hadari@is.haifa.ac.il

Meira Levy
Department of Industrial Engineering and Management,
Shenkar College of Engineering and Design, Ramat-Gan, Israel
lmeira@shenkar.ac.il

Abstract— This paper presents a case study of an IT firm, in which the deployment architecture process was analyzed from a knowledge management (KM) perspective, using the KM audit methodology, SEKAM. The analysis identified several KM gaps, which can cause serious deployment requirements' traceability (RT) problems. Based on these findings, we propose a preliminary reengineered deployment architecture process, including enhanced knowledge repositories and processes that facilitate RT throughout the deployment process. The enhanced process aims at assisting architects in better receiving a precise and wide perspective of the overall requirements of the product, and improves RT throughout the deployment process.

Keywords-component; deployment architecture; knowledge management; knowledge audit; requirements' traceability Introduction

I. INTRODUCTION

Deployment architecture is an important part of the software development lifecycle. Its purpose is to bridge the gap between the requirements of the capabilities, to the point where the solution is delivered and installed at the customer`s site [1]. Deployment architecture is defined as "allocation of the system's software components (and connectors) to its hardware hosts" [2,p.1], and highly influences quality of service (QoS). Since the deployment solution involves adjusting the product to the customer`s needs and environment, it has a major effect on customer satisfaction of the product [3].

Several empirical studies have examined the deployment process, most of which focus on defining the requirements met in the deployment architecture [4]. Deployment concerns that are not widely inquired include the challenges along different phases of the deployment process and the relations the service architect (who creates the architecture, hereafter referred to as the architect) maintains with other stakeholders. These concerns may influence the decisions made by the architect while constructing the deployment architecture solution, namely, specifying the components composing the solution and the connection between them. This stage, though sometimes short, is very important and affects the product`s overall architecture.

An important aspect of the software development process is requirement traceability (RT). Gotel and Finkelstein [5] define RT as "the ability to describe and follow the life of a requirement, in both a forwards and backwards direction." They address several problems and solutions related to traceability, emphasizing the importance of managing and organizing information before, during and after requirements specification. Spanoudakis and Zisman [6] discuss the use of automated tools as a way to enhance RT in software projects. However, we did not find studies addressing the deployment RT within the deployment architecture process.

The deployment architecture process is a knowledge-intensive business process (KIBP), thus it requires knowledge-related practices. Knowledge management (KM) has been recognized as an essential component of knowledge-intensive industries, which are characterized by technological uncertainty and a competitive environment [7]. Over the past years, we have witnessed an increased focus on KM as a major part of organizational strategy in knowledge-intensive organizations and as a significant driver for business process design and reengineering in such organizations [8]. KM enhances the ability of knowledge-intensive organizations to continuously learn and adapt, and to rapidly respond to changes in technology, market, and customer preferences [7], mainly by improving their KIBP.

In this study, we analyze the deployment architecture process using the KM audit methodology SEKAM [9], which aims at identifying KM requirements within a KIBP. We posit that handling knowledge throughout the deployment architecture process enhances handling deployment RT. Accordingly, our research question is: "How can we enhance deployment RT within the deployment architecture process, using a KM audit methodology?" In this ongoing study, we have examined thus far this question from the viewpoint of deployment architects, based on a case study in a large IT firm. Based on that, we present a preliminary proposal for managing and enhancing requirements and architecture interdependencies in the domain of deployment, as a means to

assist firms in improving the deployment architecture process, and as a means to assist architects in creating better deployment solutions.

II. Theoretical Background

A. Deployment architecture

Creating deployment architecture is a process that involves matching all the customer requirements to the physical components, and creating an overall solution [1]. We found in the literature two main approaches of defining deployment architecture, viewing deployment architecture either as a process, namely, describing the sequence of activities of creating the deployment architecture, or as an artifact, namely describing the deployment documents and blueprints.

The definitions referring to the process elaborate on the different steps of creating the deployment architecture solution including, for example, the interrelated activities of release, install, activate, deactivate, update, and adapt (e.g., [1]). The definitions referring to the artifact typically state what information the documents and blueprints comprising this artifact should provide (e.g., [4]).

These two approaches can be viewed as complementary. However, since our intent here was to uncover factors related to the decisions made by the architects while creating a deployment architecture solution, we focus on the deployment architecture process.

B. Requirement Tracability (RT)

As stated in the introduction, RT is recognized as a concern in guidelines and standards in requirement engineering [5]. It tackles the changes a requirement undergoes, prior and post its inclusion in the requirement specification, and its overall effect on the system.

Spanoudakis and Zisman [6] suggest the use of repositories as a tool for conducting successful RT. They use the term "traceability relations" to describe several challenges in a system`s lifecycle; for example, the way a change in requirements will be expressed in the system`s structure. However, they address tractability with respect to the full software development cycle, without referring to its various stages and aspects. Specifically, the need to follow the system`s compatibility with a particular set of requirements, which are related to its deployment, has not yet been studied.

C. Social Engineering Knowledge Audit Methodology (SEKAM)

SEKAM [9] is a methodology for eliciting and analyzing KM infrastructure requirements in the context of KIBP. SEKAM is highly connected to the business context through integrative audit of knowledge and business processes. It provides integration of both aspects – social and engineering – for analyzing KM infrastructure solution requirements. Its comprehensive analysis identifies knowledge bottlenecks, thus contributing to KIBP reengineering. This study utilizes f SEKAM for identifying and studying KM gaps within the deployment architecture process and presents a reengineered process that enhances the KM of the software deployment process, focusing on its RT.

III. Research method

The main objective of this study was to explore the work process of deployment architects, and their perceptions on that process, focusing specifically on the process of RT. For conducting this research, we chose the qualitative research approach, where the investigator does not have a pre-defined theory about the environment inspected, but rather uses techniques and tools to explore and discover phenomena [10]. We used the grounded theory methodology (ibid), in which data are collected from the field, analyzed and used to build a theory, as was done in this research in the context of the deployment process.

We initiated our research by distributing an open-questions questionnaire to 25 architects at a global, large-scale IT firm. This firm is one of the largest IT management software providers worldwide, with headquarters in the US and 150 offices in more than 45 countries. The questionnaires were intended to achieve a preliminary identification of the major concerns of the deployment process and its main challenges and opportunities for improvement, as perceived by architects. Next, we conducted 5 in-depth interviews with senior service architects in the firm, focusing on the challenges emerging from the questionnaires. Data collected were inductively analyzed and categorized. Categories were detected and named by using open coding [6]. Data was then further analyzed using the SEKAM model [9], in order to describe the process and its challenges, and suggest a preliminary solution for KM within the process. While using grounded theory, it is permitted to consult the literature for further analysis of the data [11]. Due to space limitations, in this paper we present a partial set of the SEKAM tools (*SEK 1, SEK 6 and SEK 8*) that best describe the knowledge gaps within KIBP and the reengineered KIBP, and in our case elicit RT concerns and solutions.

IV. Main findings

In this section we present our findings of RT concerns. Some emerged concerns were not directly described by the architects as RT concerns, but – based on literature review – were found to imply to RT problems in the deployment process.

A. SEK 1 - General description of the business process environment

The first stage of SEKAM aims at identifying knowledge oriented problems and opportunities within organizational KIBPs (*SEK 1*). Data collected in questionnaires and interviews, point at the deployment architecture process as the KIBP to focus on, since it is a major factor influencing customer satisfaction.

The deployment architecture process starts when an architect receives the product requirements from the marketing unit. In some cases, marketing promises customers product functionalities without checking their feasibility in advance. This lack of knowledge in this case results is conflicts, misunderstandings and critical errors in product design and deployment. In addition, since architects are involved with product requirements at a late stage of the project, they usually do not have the ability to express their opinions about the requirements.

"Sales people sell the software and not the services with it; there is always a gap between business requirements and capabilities… the architect should always be a part of pre-sales."

Next, the architect creates the high-level design document (HLD) with no designated tool that could have also facilitated diagram drawing. In cases where diagrams are required during the design, they are created with stand-alone CASE tools which make the design validation and relevant knowledge extraction harder. Therefore, architects often use peer support (product managers), Share Point documents from previous projects, and online help from the firm`s guide books or the internet. Moreover, since there is no FAQ repository, additional required information about product`s components is hard to find.

"One of my biggest concerns is lack of availability of validated certified references".

After completing the HLD, a review session usually takes place. In this review, each stakeholder (i.e., market analyst, product manager) usually addresses only requirements which are relevant to their point of view. There is no tool which enables the different stakeholders to view and communicate about the requirements which are relevant to several stakeholders, or can be critical in the future and affect the overall product behavior. In addition, architects do not obtain the ability to get several views of the solutions. As a result, the architect is not aware of changes in product configuration, which might prevent the product from being installed at the customer site because errors or deviation of the product from the requirements. In addition, since these tools contain a vast amount of complicated, unneeded features, architects usually prefer to work with simpler tools, which do not provide the wider needed perspective.

"I want to be able to receive information from automated tools about failing tests of another stakeholder I depend on…People prefer to use excel sheets rather than requirement managing tool… the current tool has a lot of un-used features".

B. SEK8 - Requirement traceability problem example

For combining hard elements such as process flowchart and soft elements such as culture, the *SEK 8* modeling tool [9], named knowledge interactions diagram, is used, as demonstrated in Figure 1. We were frequently told that in recent years the firm has been increasingly concerned with customer complaints, and in some cases even the CEO got involved in their resolution. We found that in some cases, well managed RT would have been able to prevent these complains.

Figure 1 illustrates an example of a lack of RT during the deployment process, where a customer was promised to have an XML interface of the product, while the product itself did not have this feature. This gap was discovered only at the product installation. The flow presented in the diagram is based on a real case we encountered in the interview, and reflects the conversations between various stakeholders in the deployment architecture process. The scenario starts during installation when the customer complains about having no XML interface to his company portal, although they specified it and the marketing division approved it. While the product manager knew that this requirement could not be fulfilled, the deployment architect was not updated on that. Moreover, this requirement was not written in the SRS document, resulting in no XML interface in the product. Support team checked the complaint and found it just, as did the QA team. This did not please the project manager. Checking with the product manager, it appeared that this requirement could not be implemented, which decreased customer satisfaction. Had this problem been identified in advance, and informed to all the stakeholders, this undesired event could have been prevented.

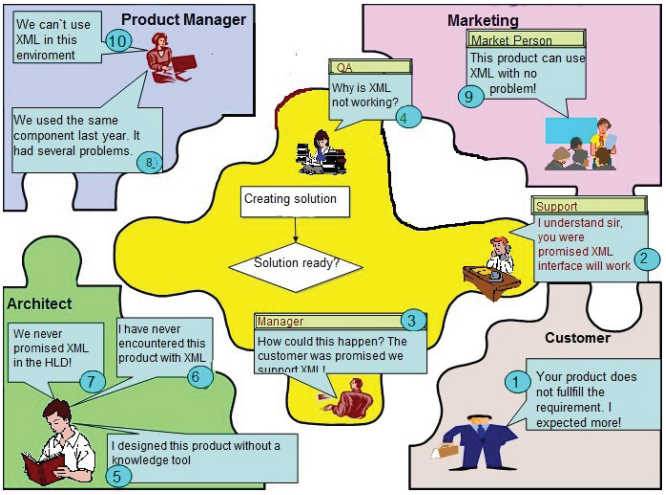


Figure 1. *SEK 10* - Conceptual Scheme of Informal Knowledge Interaction Diagram[9]

C. Reengineering the deployment architecture process

SEK 6 modeling technique [9] aims at generating a knowledge inventory diagram and examining knowledge bottlenecks of the process, and is originally presented before *SEK 8*. However, due to space limitations we used *SEK 6* also for suggesting solutions that will be incorporated into the deployment architecture process. Hence, *SEK 6* describes both, the current and the proposed processes. The diagram indicates knowledge input bottlenecks within the deployment architecture tasks (i.e., marketing information during requirements' specification; similar solutions during looking for existing solutions). Realizing that the firm does not use a requirement tool or a knowledge repository, we decided to add them to the presented current deployment architecture process flow. However, the use of knowledge tools is not enough in order to improve RT [6]. Additional changes to the process were made, mainly adding a phase of brainstorming review sessions, and connecting the requirement tool usage to various steps in the process (i.e., designing a solution). In addition, we proposed updating the knowledge repository at each step.

In what follows, we present a preliminary model of the deployment architecture process, based on the problems we identified. It addresses the RT problem directly. Since this is an ongoing research, we plan to further refine this model. The reengineered process is presented in Figure 2, where we elaborate on additional tasks that focus on KM and enable RT:

- Brainstorming review session – includes the marketing person, product manager, and architect. In this session the main project requirements are presented, the requirement tool is used for handling the project's requirements from the

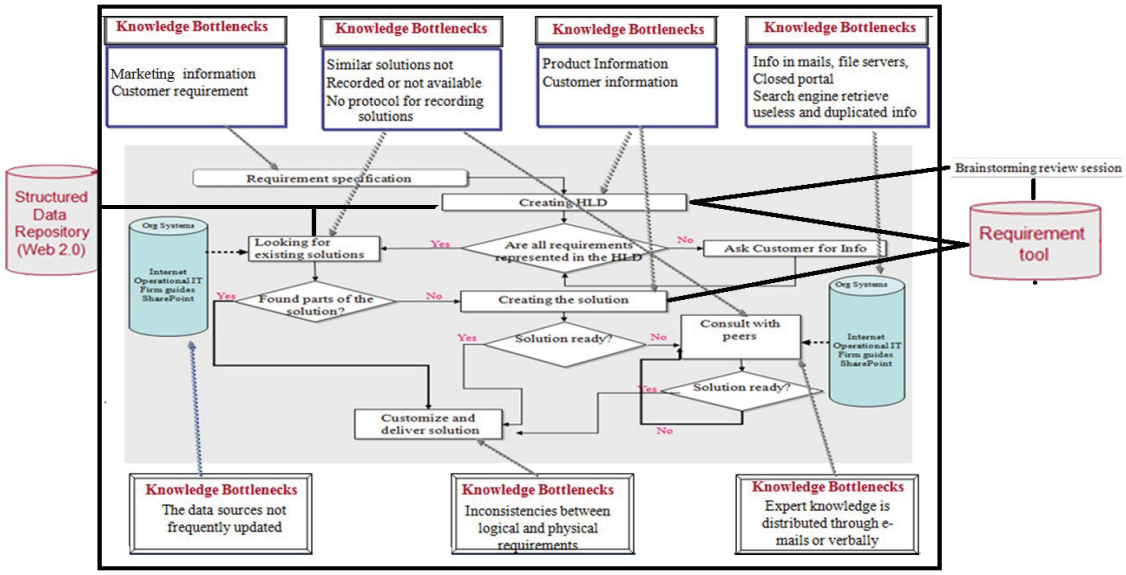


Figure 2. *SEK 8* – Deployment architecture flow chart –problems and suggestion for improvement

start, and common issues are raise concerning possible challenges in the project.

- Creating the high-level design – utilizing the tool and the repository, which include all the information, gathered from similar former projects, thereby, enabling architecture deployment reuse. The architect can easily access all relevant information via designated requirement tool – documents, forums, blogs, peers' comments, wikis – without spending time searching for the information. If there is a lack of information, the architects can summon a peer review, where they can get guidance from other stakeholders.
- Customize and deliver – at this stage, the architect has finished building the solution and delivers it to the customer. Since many of the conflicts in requirements can occur at this stage, the architect can use the data repository and the KM tool to immediately seek for help and consult with peers.

V. Conclusion

In this paper we presented a work in progress, focusing on a case study of an IT firm, in which we analyzed the deployment architecture process from a KM point of view, aiming to identify knowledge gaps in the process and their sources. We found the KM perspective appropriate for our research since the interviewees referred time and again to knowledge they require and not always have access to, and the ability to trace requirements' changes.

Using the SEKAM model to analyze the existing situation and developing the structure of the solution, helped us to point out the RT problem as a basic concern of the process. Based on these findings, we proposed a reengineered deployment architecture process, facilitated with enhanced knowledge repositories and processes that will assist architects in better receiving a precise and wide perspective of the overall requirements of the product, and improve RT throughout the deployment process. Since the proposed solution is embedded in the process, little additional effort is required. Future research will study the proposed model implementation and further required development and evaluate the solution.

VI. References

[1] A. Carzaniga, A. Fuggetta, R. S. Hall, D. Heimbigner, A. Van Der Hoek, and A.L. Wolf, "A characterization framework for software deployment technologies," Technical Report. Colorado State Univ. Fort Collins Dept of Computer Science, 1998.

[2] N. Medvidovic, and S. Malek "Software deployment architecture and quality-of-service in pervasive environments," Int. Workshop on Engineering of Software Services for Pervasive Environments, in conjunction with the 6th ESEC/FSE joint meeting, ACM, 2007, pp. 47-51.

[3] A. Mockus, P. Zhang, and P. L. Li, "Predictors of customer perceived software quality," Int. Conference on Software Engineering (ICSE 2005), ACM, May 2005, pp. 225–233.

[4] S. Adam and J. Doerr "On the notion of determining system adequacy by analyzing the traceability of quality," Workshops and Doctoral Consortium, vol. 1, 2007.

[5] O. C. Gotel, and C. W. Finkelstein, "An analysis of the requirements traceability problem," Proc. IEEE Int. Conf. on Requirements Engineering, April 1994, pp. 94-101.

[6] G. Spanoudakis,and A. Zisman, "Software traceability: a roadmap," Handbook of Software Engineering and Knowledge Engineering, 3, 2005, pp. 395-428.

[7] A. Lonnqvist, "Business Performance Measurement for Knowledge-intensive Organizations, Business Performance Measurement: Towards Organizational Excellence," Le Magnus University Press, 2005, pp. 17 - 35.

[8] N. Gronau, and E. Weber, "Management of Knowledge Intensive Business Processes,"In Business Process Management, Desel, J., Pernici, B., Weske, M. (eds.), Springer-Verla, 2004.

[9] I. Aviv, M. Levy, and I. Hadar, "Knowledge-Intensive Business Process Audit: The Practical Aspect," Proceeding of the 9th International Conference on Knowledge Management and Knowledge Technologies, Graz, Austria, 2009.

[10] A. Strauss, A. and J. Corbin, "Basics of Qualitative Research: Grounded Theory Procedures and Techniques," Sage Publications, Inc., 1990.

[11] R. Suddaby "From the editors: What grounded theory is not," Academy of Management Journal 49(4), 2006, pp 633-642.

Generating Partial Covering Array for Locating Faulty Interactions in Combinatorial Testing

Ziyuan Wang[1, 2] Ting Guo[1] Wujie Zhou[3] Weifeng Zhang[1] Baowen Xu[2]
[1]School of Computer Sci. & Tech., Nanjing University of Posts and Telecommunications, Nanjing, 210003, China
[2]State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, 210093, China
[3]Department of Mathematics, Southeast University, Nanjing, 210096, China
Corresponding: wangziyuan@njupt.edu.cn

***Abstract*— Combinatorial testing is widely used to detect failures caused by interactions among parameters. After detecting faults, the subsequent problem in combinatorial testing is characterizing faulty interactions that cause these detected failures. In order to characterize faulty interactions accurately and efficiently, a test suite that usually differ from the original combinatorial test suite is required in fault diagnosis. By assuming the number of faulty interactions is under control, the *partial covering array*, whose size grows logarithmically with the number of parameters, can serve as a test suite for locating faulty interactions in the non-adaptive faulty interaction locating process. In this paper, we extend the one-test-at-a-time framework, which is widely used in combinatorial test generation, for generating partial covering arrays. We combine this framework with greedy methods and meta-heuristic search methods respectively. An experiment shows that partial covering array needs smaller number of test cases than existing non-adaptive methods.**

Keywords — combinatorial testing; software debugging; fault diagnosis; faulty interaction; minimal faulty schema

I. INTRODUCTION

Assume that we have a software system whose behavior is affected by k parameters (or factors). In the testing of this software system, besides the failure that caused by a specific parametric value of a specific parameter, there may be failure that caused by a combination of τ specific parametric values of τ specific parameters ($1<\tau\leq k$). We call such a failure-causing combination of τ parametric values as a faulty interaction with size τ or a τ-way faulty interaction.

In order to testing these faulty interactions, an ideal method is the exhaustive testing that covers all the possible k-tuple combinations of parametric values. But it is unfortunate that, the exhaustive testing is unacceptable since the number of required test cases grows exponentially with k. Combinatorial testing, which uses covering array or mixed covering array as test suite, could provide a tradeoff between the cost of testing and the degree of interaction coverage. E.g., for a given integer $\tau(<k)$, the τ-way combinatorial testing requires covering all the τ-tuple combinations of parametric values rather than the k-tuples, and the number of test cases in the τ-way combinatorial test suite grows logarithmically with k [1]. The reduction of test cases in combinatorial test suite will not reduce the failure detection ability of combinatorial testing in most applications, since the investigation on real software failures states that most failures are caused by faulty interaction with a small size [2].

Combinatorial testing is used widely for its efficiency and effectiveness. Up to date, most works in combinatorial testing focus on test generation and fault detection. These works could provide multiply results on detecting software failures. After detecting failures by combinatorial test suite, we must answer a question: why did the software fail in testing? Or what is the faulty interaction that causes failure? The fault diagnosis will answer this question and provide an assistant to the software debugging. In the fault diagnosis, the faulty interaction that causes the detected failure will be characterized and located, by using the information that obtained from the execution results of test cases in combinatorial test suite or some other types of test cases.

All the previous faulty interaction locating techniques in combinatorial testing could be categorized into two classes: *adaptive strategy* and *non-adaptive strategy* [3]. Adaptive strategy generates and runs additional test cases by using the executions results of prior test cases in combinatorial test suite. And sometimes it need diagnose iteratively. In non-adaptive strategy, test cases that characterizing faulty interactions are independent from the execution results of prior test cases. So there is an advantage of non-adaptive strategy that, test cases for diagnosis could be generated before the execution of those test cases for detecting failures. Such an advantage could help us to increase the efficiency of locating faulty interactions, since the detection and locating of faulty interactions could be parallel if some essential conditions are satisfied.

In this paper, we mainly focus on the non-adaptive faulty interaction locating strategy because of its advantage on high efficiency. Besides the existing *locating and detecting arrays* [3] and *error locating array* [4][5], we found another algebraic structure we called *partial covering array* [6] could be used as test suite for locating faulty interactions, and may be better than previous two types of arrays since it requires less test cases. **Contributions of our paper mainly include**: (1) Introduce non-adaptive faulty locating strategy in detail. (2) Extend the one-test-at-a-time strategy that widely used in combinatorial test generation for generating partial covering array, and combine this framework with greedy strategy and meta-heuristic search strategy. (3) Give simulation experiment to compare proposed algorithms, and show that partial covering array needs smaller

number of test cases than other existing non-adaptive faulty interaction methods.

The rest of this paper will be organized as follows. Section 2 presents backgrounds. Section 3 describes the non-adaptive faulty interaction locating strategy. The definition of partial covering array is given is section 4. We extend the one-test-at-a-time strategy to generate partial covering array in section 5. Experimental results are shown in sections 6. Related works are discussed in section 8. And finally a conclusion is given.

II. PRELIMINARIES

A. Combinatorial Testing

For a software system with k parameters (or factors), we suppose each factor f_i has a_i ($1 \le i \le k$) discrete valid values. Let $F=\{f_1, f_2, \ldots, f_k\}$ denote the set of factors, and $V_i=\{0, 2, \ldots, a_i-1\}$ ($1 \le i \le k$) the value set for f_i without loss of generality.

Definition 1 (Test case). A k-tuple $test=(v_1, v_2, \ldots, v_k)$ is a test case, where $v_1 \in V_1, v_2 \in V_2, \ldots, v_k \in V_k$.

Definition 2 (Covering array). Given a 2-dimension array $A=(a_{i,j})_{m \times k}$, where the j-th column denotes the parameter f_j and all elements of this column come from the set V_j ($j=1, 2, \ldots, k$), that is $a_{i,j} \in V_j$. If each $m \times \tau$ ($1 \le \tau \le k$) sub-array contains all value combinations of corresponding τ parameters, then A is a τ-way covering array or a covering array with strength τ. It could be denoted as $CA(m; \tau, F)$. [7]

A τ-way combinatorial test suite could be obtained easily from τ-way covering array, by mapping each row of covering array to a test case of combinatorial test suite. So a τ-way combinatorial test suite must cover all tuples in set:

$$CombSet = \bigcup_{r \in R} CombSet_r$$

Where

$$R = \{\{f_{i,1}, f_{i,2}, \ldots, f_{i,\tau}\} \mid f_{i,1}, f_{i,2}, \ldots, f_{i,\tau} \in F\}$$

$$CombSet_r = \{(v_{i,1}, v_{i,2}, \ldots, v_{i,\tau}) \mid r = \{f_{i,1}, f_{i,2}, \ldots, f_{i,\tau}\}, v_{i,1} \in V_{i,1}, v_{i,2} \in V_{i,2}, \ldots, v_{i,\tau} \in V_{i,\tau}\}$$

TABLE I. PARAMETERS AND THEIR PARAMETRIC VALUES

Browser	OS	Connection	DB
Firefox	Linux	LAN	DB/2
IE	Windows	ISDN	Oracle
Opera	Macintosh	Modem	Access

TABLE II. A 2-WAY COMBINATORIAL TEST SUITE

No.	Browser	OS	Connection	DB
1	Firefox	Linux	LAN	DB/2
2	Firefox	Windows	ISDN	Oracle
3	Firefox	Macintosh	Modem	Access
4	IE	Linux	ISDN	Access
5	IE	Windows	LAN	Oracle
6	IE	Macintosh	Modem	DB/2
7	Opera	Linux	Modem	Oracle
8	Opera	Windows	LAN	Access
9	Opera	Macintosh	ISDN	DB/2

B. Faulty Interaction

In combinatorial testing, we mainly focus on the failure caused by a combination of τ specific parametric values of τ specific parameters ($1 \le \tau \le k$). We call this type of failure as the interaction failure, and call such a failure-causing combination as a faulty interaction with size τ or a τ-way faulty interaction.

In the field of faulty interaction locating, there is usually an assumption that, a test cases matching a faulty interaction must fail [3][4][8][9]. So any higher-size interaction matching a τ-way faulty interaction must be a faulty interaction too. In order to focus our views on the small-size faulty interactions, we use the model of minimal faulty schema [8].

Definition 3 (Interaction schema). A k-tuple $s=(-, \ldots, -, v_{i,1}, -, \ldots, -, v_{i,2}, -, \ldots, -, v_{i,\tau}, -, \ldots, -)$ is a τ-way interaction schema, or interaction schema with size τ, or τ-schema for short ($1 \le \tau \le k$). Where τ values are fixed as $v_{i,1} \in V_{i,1}, v_{i,2} \in V_{i,2}, \ldots, v_{i,\tau} \in V_{i,\tau}$, and other k-τ values are not fixed and represented as "-".

Definition 4 (Sub-schema and parent-schema). Schemas $s_1=(v_1, v_2, \ldots, v_k)$ and $s_2=(v_1', v_2', \ldots, v_k')$ are τ_1-schema and τ_2-schema respectively ($\tau_1 \le \tau_2$). If $\forall 1 \le i \le k$, $(v_i="-") \vee (v_i=v_i')$ is true, then s_1 is sub-schema of s_2, and s_2 is parent-schema of s_1. It is denoted as $s_1 \prec s_2$. Especially, if $s_1 \neq s_2$, then s_1 is real sub-schema of s_2, and s_2 is real parent-schema of s_1.

For a test case t, there are totally 2^k-1 sub-schemas. They form a set $ScheSet(t)=\{s \mid s \prec t\}$. And for a test suite T, there is $ScheSet(T)=\{s \mid s \in ScheSet(test), test \in T\}$.

Definition 5 (Minimal schema). A schema $s \in ScheSet$ is a minimal schema of a set of schemas $ScheSet$, if $\forall s' \in ScheSet$ $(s' \prec s) \Rightarrow (s'=s)$.

E.g, for the test case t=(IE, Linux, ISDN, Access) in Table 2, $ScheSet(t)$={(IE, -, -, -), (-, Linux, -, -), (-, -, ISDN, -), (-, -, -, Access), (IE, Linux, -, -), (IE, -, ISDN, -), (IE, -, -, Access), (-, Linux, ISDN, -), (-, Linux, -, Access), (-, -, ISDN, Access), (IE, Linux, ISDN, -), (IE, Linux, -, Access), (IE, -, ISDN, Access), (-,Linux, ISDN, Access), (IE, Linux, ISDN, Access)}. There are totally 4 minimal schemas (IE, -, -, -), (-, Linux, -, -), (-, -, ISDN, -), (-, -, -, Access) in $ScheSet(t)$.

Definition 6 (Faulty schema). A schema $s \in ScheSet(T_{all})$ is a faulty schema, if test cases $t \in T_{all}=V_1 \times V_2 \times \ldots \times V_k$ containing s as a sub-schema $\Rightarrow$ t is failed in testing.

Definition 7 (Minimal faulty schema). A faulty schema s is a minimal faulty schema (or MFS), if any real sub-schema of s is not faulty schema.

Based on such a minimal faulty schema model, each faulty interaction can be mapped to a faulty schema. When locating faulty interactions, only the interactions that described as a minimal faulty schema need to be characterized.

III. NON-ADAPTIVE FAULTY INTERACTION LOCATING

In order to characterize faulty interactions or minimal faulty schemas, both adaptive strategy and non-adaptive strategy are available. Here "adaptive" means that, test cases that used in the phase of fault diagnosis (we call them *diagnosing test cases* in this paper) should be generated and executed according to

the execution results of prior test cases that used in the phase of failure detection (we call them *detecting test cases*).

Differing from adaptive strategy, in non-adaptive strategy, diagnosing test cases could be generated before the execution of detecting test cases, since all diagnosing test cases should be independent from the execution results of detecting test cases. So if there is high risk of failures in testing, we could design and execute both detecting test suite and diagnosing test suite in parallel. Such a parallel work can increase the efficiency of locating faulty interactions, and even the efficiency of whole process of software development and maintenance. Figure 1 shows the difference between the processes of adaptive strategy and non-adaptive strategy.

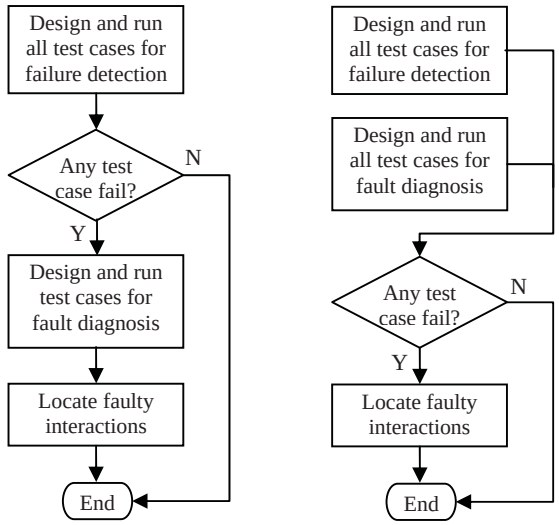


Figure 1. Processes of adaptive fault locating strategy (left) and non-adaptive fault locating strategy (right)

One of possible scenarios, where parallel detecting failures and locating faults are required, is the well-known *daily build* or *nightly build*. A daily build is the practice of each day doing a building and smoke testing of the latest version of a program [18]. Typically, developers submit program after work, and testers test program on night. Now we assume that there is a high risk of detecting failures. If the adaptive fault diagnosis were started in the next day, the resource of fixing software bugs, which is a manual action, would be occupied by the fault locating. On the contrary, if we started the non-adaptive fault diagnosis in parallel with the nightly testing, developers could fix bugs at the beginning of the next working day. It means that, in the daily build process, non-adaptive faulty interaction locating may increase the efficiency of software development.

IV. Partial Covering Array in Non-Adaptive Faulty Interaction Locating

For non-adaptive faulty interaction locating strategy, people have proposed to use *locating and detecting arrays* [3] and *error locating array* [4][5] as the diagnosing test suites. Here we focus on the error locating array in this paper.

For a system with safe values, if the maximum number of minimal faulty schemas is not larger than integer d, and the maximum sizes of minimal faulty schema is not larger than integer τ, a ***error locating array*** $ELA(m; \tau, F)$, which contains m test cases, can locating all minimal faulty schemas. Here the system has safe values, if $\forall f_i \in F$, $\exists v \in V_i$ is a safe value that involved in none minimal faulty schema. Algorithm 1 gives the process of locating faulty interactions.

Algorithm 1. Locating minimal faulty schemas with error locating array

Input: $ELA(m; \tau, F)$: a error locating array
Output: *MFSs*: minimal faulty schemas

1. $MFSs = \varnothing$;
2. $\Gamma = ScheSet(ELA(m; \tau, F))$;
3. **For Each** test case $t \in ELA(m; \tau, F)$
4. **If** (test case t passes) **Then**
5. $\Gamma = \Gamma - ScheSet(t)$;
6. **End If**
7. **End For**
8. Return all minimal schemas in Γ;

People usually use a high-way covering array $CA(m; \tau+d, F)$ as the error locating array $ELA(m; \tau, F)$ for the situation that $\tau+d \le k$ [4][5]. However, the numbers of test cases in these high-way covering arrays are usually too large. Hence it is necessary to find other method to construct error locating array. Zhou *et al* proposed to use ***partial covering array*** to replace traditional error locating array [10].

Definition 8 (Partial covering array). Assume Γ is a set of interaction schemas. Given a array $A=(a_{i,j})_{m\times k}$, where the j-th column denotes the factor f_j and all elements of this column come from the set V_j (j=1, 2,…, k), that is $a_{i,j} \in V_j$. If all schemas in Γ could be covered by at least one row of A, then A is a τ-way partial covering array or a partial covering array with strength τ. It could be denoted as $PCA(m; \tau, F, \Gamma)$.

Theorem 1. For a software system with safe values, assume that $0 \in V_i$ is one of safe values of parameter $f_i \in F$. If the maximum number of minimal faulty schemas is not larger than integer d, the maximum sizes of minimal faulty schema is not larger than integer τ, and $\tau+d \le k$, then a partial covering array $PCA(m; \tau+d, F, \Gamma_{\tau, d})$ is a $ELA(m; \tau, F)$, where the tuple set $\Gamma_{\tau, d}$ ={($\tau+d$)-way schema | d values are fixed as 0, τ values are fixed as any element except 0}.

The proof of this theorem could be found in ref [10].

The partial covering array has an advantage over traditional error locating array that, it needs smaller number of test cases. If the software system has a property that $a=a_1=a_2=\ldots=a_k$, the number of test cases in corresponding partial covering array follows $m=O(d^{\tau+1}\times \log k)$, while the size of ($\tau+d$)-way covering array follows $m=O(d\times a^{d+\tau}\times \log k)$ [10].

V. Generating Partial Covering Array with One-test-at-a-time Strategy

Though partial covering array could use smaller number of test cases to serve as an error locating array in non-adaptive

faulty interaction locating, the problem of generating partial covering array has not been studied previously. We study this problem in this section.

For traditional combinatorial test case generation, one-test-at-a-time strategy have been well studied [7][17]. We extend this strategy for the partial covering array generation. The framework of one-test-at-a-time strategy for partial covering array can be summarized and described as Algorithm 2. It starts with an empty initial test suite. Then build one test case at a time to cover some uncovered tuples in $\Gamma_{\tau, d}$ until all the tuples in $\Gamma_{\tau, d}$ have been covered.

Algorithm 2. One-test-at-a-time Framework

Input: F: set of parameters, $\Gamma_{\tau, d}$: set of uncovered tuples
Output: T: partial covering array

1. $T = \varnothing$;
2. $Uncover = \Gamma_{\tau, d}$;
3. **While** ($Uncover \neq \varnothing$)
4. Build one single test case t;
5. $T = T + \{t\}$;
6. $Uncover = Uncover - ScheSet(t)$;
7. **End While**

To generate a partial covering array as small as possible in one-test-at-a-time framework, an ideal method, which selects a "best" single test case each time to cover the greatest number of uncovered tuples in $\Gamma_{\tau, d}$, is fascinating to us. But it is unfortunate, selecting such a "best" test case is very difficult. Colbourn *et al* proved that, in combinatorial test generation, the problem that determining whether there is a test case covering a given number of uncovered tuples is a NP-C problem [17]. It is similar in the problem of partial covering array generation.

For above limitations, to replace the method that selecting the "best" test case in one-test-at-a-time framework, a feasible method is to generate approximate "best" test case with some more efficient strategies, such as greedy strategy and meta-heuristic search strategy.

A. Greedy Methods

A feasible method to generate a single test case in partial covering array is determining an order of factors and fixing parametric values in a determined turn. This kind of algorithms in combinatorial test generation includes AETG [1], DDA [17], and etc. We attempt to extend these two algorithms for the partial covering array generation.

1) AETG-Liked Algorithm:

When generating a single test case, the first parameter and its value are selected as the one that appears in the greatest number of uncovered tuples in $\Gamma_{\tau, d}$. Then the order of other parameters is random, and the value of each parameter is fixed as a value that appears most frequently in the uncovered tuples in $\Gamma_{\tau, d}$.

2) Density-Based Algorithm:

When generating a single test case, the order of parameters is determined deterministically according to the "local density" of each parameter. To fix parametric value, the value that takes the greatest "global density" is selected. Because the partial covering array needs to covers all the (τ+d)-tuple combinations in $\Gamma_{\tau, d}$, the local density and global density are same as that in the generation of (τ+d)-way combinatorial test suite [17].

B. Meta-Heuristic Search

Besides the greedy strategy, another feasible method to generate a single test case in partial covering array is meta-heuristic search optimization, including simulated annealing, generic algorithm, ant colony algorithm, taboo search, particle swarm, and etc. We attempt to apply three of these algorithms in the one-test-at-a-time framework for partial covering array.

1) Simulated Annealing:

Simulated annealing applies transformations on a single test case. The modified parameters and its values are selected random. To determine whether to accept a modification, the cost is evaluated and accepted according to a cooling schedule. Based on the information about cost and cooling schedule, the probability of acceptance is computed. In our application, the cost could be defined according to the number of new-covered (τ+d)-tuple combinations in $\Gamma_{\tau, d}$.

2) Genetic Algorithm:

Genetic algorithm mimics the process of natural evolution. It is based on the concept that the candidate solution created by swapping two good candidates is also good. Genetic algorithm applies transformations on a population of test cases. Test cases will be modified in a process including selection, crossover, mutation, and be evaluated by a fitness function. The fitness function depends on the number of new-covered (τ+d)-tuple combinations in $\Gamma_{\tau, d}$.

3) Particle Swarm Optimization:

Particle swarm optimization could optimize a test case by moving a population of candidate test cases. Particles (or test cases) move according to simple mathematical formulae over the particle's position and velocity. Each particle's movement is influenced by its local best known position and is also guided toward the best known positions in the search-space, which are updated as better positions are found by other particles. Here the position is determined by test cases' tuple coverage ability.

VI. Computational Results

Experiments in this section are designed to compare the performance of partial covering array generation algorithms, including the AETG-liked algorithm, density-based algorithm, simulated annealing, genetic algorithm, and particle swarm. All these algorithms work in the framework of one-test-at-a-time to generate a single test case each time.

Table 3 illustrates the sizes of partial covering arrays that generated by different 5 algorithms. For simulated annealing (SA), genetic algorithm (GA), and particle swarm (PSO), we list the worst, average, and best results respectively for each input, since all these meta-heuristic search algorithms are non-deterministic. We can conclude from the data that, for many inputs, meta-heuristic search algorithms generate the smaller

TABLE III. SIZES OF PARTIAL COVERING ARRAYS AND HIGH-WAY COVERING ARRAYS

Input			AETG-like	Density-based	SA(*Wst/Avg/Best*)	GA(*Wst/Avg/Best*)	PSO(*Wst/Avg/Best*)	(τ+*d*)-way covering array
F	τ	*d*						
2^6	2	2	16	17	23/17/15	29/17/15	22/19/16	21
3^{10}	2	2	122	119	133/120/119	141/123/119	136/129/119	159
4^{13}	2	2	366	363	377/368/360	401/366/361	370/364/360	508
5^{15}	2	2	743	744	765/749/739	798/773/740	769/744/740	1245
2^{20}	2	3	108	108	119/109/107	129/109/107	121/114/109	119
3^{20}	2	3	465	462	461/460/460	499/467/459	476/465/460	1266
4^{20}	2	3	1058	1056	1102/1056/1043	1152/1067/1043	1077/1056/1043	5516
5^{15}	2	3	1417	1420	1439/1420/1411	1499/1478/1419	1441/1419/1412	12704
3^{20}	2	4	763	763	770/764/760	793/769/763	767/764/761	4486
4^{15}	2	4	1240	1239	1299/1247/1235	1370/1264/1237	1253/1239/1235	16384
5^{15}	2	4	2207	2207	2251/2236/2206	2282/2247/2206	2246/2227/2206	82139
2^{10}	3	2	47	47	52/50/47	56/54/47	59/56/47	56
3^{14}	3	2	696	696	711/707/696	736/715/696	709/703/696	922
4^{15}	3	2	2592	2597	2607/2600/2590	2693/2631/2590	2600/2597/2590	3364
2^{10}	3	3	109	108	110/109/107	137/121/108	112/110/107	116
3^{15}	3	3	1686	1677	1679/1671/1665	1722/1696/1672	1689/1669/1663	3234
4^{12}	3	3	4093	4088	4098/4082/4079	4099/4090/4081	4100/4087/4078	14281

partial covering arrays than greedy algorithms. And especially, the particle swarm usually has a better performance than other meta-heuristic search algorithms.

By the way, the sizes of traditional error locating array (or (τ+*d*)-way covering array) for the same inputs are also list in the table. It is evident that, the non-adaptive faulty interaction locating method that uses *partial covering array needs smaller number of test cases* than the method that uses high-way covering array.

VII. RELATED WORKS

As we mentioned previously, there are adaptive and non-adaptive faulty interaction locating strategies.

Adaptive faulty interaction locating methods include the well known Delta Debugging [16], AIFL and Iterative AIFL methods [15], FIC and FIC_BS methods [9], constraint solving method [12], Xie's methods [13][14]. All above adaptive faulty interaction locating methods have an advantage that, they don't need any predication about faulty interactions. Besides, there are also Martinez's adaptive methods [4][5], and an improved method based on Martinez's method [11]. These three methods need an assumption that software system has safe values.

Previous non-adaptive faulty interaction locating methods include the methods that use locating and detecting arrays [3] and error locating array [4][5] as the test suite for locating faulty interactions. They *needs more test cases* than partial covering arrays to locate the same faulty interactions.

VIII. CONCLUSION

Characterizing faulty interaction is an important problem in combinatorial testing. In this paper, we mainly focus on the non-adaptive faulty interaction locating strategy, which has an advantage that diagnostic test suite could be generated and executed in parallel with the detecting test suite efficiently. The problem of generating partial covering array, which could be used as diagnostics test suite in the non-adaptive strategy, is studied in this paper. Experiment results show us that partial covering array needs small number of test cases than existing non-adaptive faulty interaction locating methods. So it is clear that partial covering array has equivalent ability and stronger efficiency in the locating of faulty interactions

In the field of non-adaptive faulty interaction locating, the most important future works may include: (1) more efficient algorithm to generate diagnostic test suite; and (2) avoid the limitation that both the maximum strength and the maximum number of faulty interactions must be given before the fault diagnosis.

ACKNOWLEDGMENT

The work described in this paper was partially supported by the National Natural Science Foundation of China (91018005, 61003020, and 61202006); Natural Science Foundation of Jiangsu Province (BK2011190), Foundation of Nanjing University of Posts and Telecommunications (NY212023); Open Foundation of Guangxi Key Laboratory of Trustworthy Software; and Six Peak Talent of Jiangsu Province.

REFERENCES

[1] D. M. Cohen, S. R. Dalal, M. L. Fredman, *et al.* The AETG System: An Approach to Testing Based on Combinatorial Design. IEEE Transaction on Software Engineering, 1997, 23(7): 437-444.

[2] D. Richard Kuhn, Dolores R. Wallace. Software Fault Interaction and Implication for Software Testing. IEEE Transaction on Software Engineering, 2004, 30(6): 418-421.

[3] C. J. Colbourn, D. W. McClary. Locating and Detecting Arrays for Interaction Faults. Journal of Combinatorial Optimization, 2008, 15(1): 17-48.

[4] C. Martinez, L. Moura, D. Panario, and B. Stevens. Algorithms to locate errors using covering arrays. In LATIN2008: Theoretical Informatics, Lecture Notes in Computer Science, 2008(4957): 504-519.

[5] C. Martinez, L. Moura, D. Panario, and B. Stevens. Locating errors using ELAs, covering arrays, and adaptive testing algorithms. SIAM Journal on Discrete Mathematics, 2009, 23: 1776-1799.

[6] P. Carey, A. Godbole. Partial Covering Arrays and A Generalized Erdos-Ko-Rado property. Journal of Combinatorial Designs, 2010, 18: 155-166.

[7] Changhai Nie, Hareton Leung. A Survey of Combinatorial Testing. ACM Computing Surveys, 2011, 43(2).

[8] Changhai Nie, Hareton Leung. The Minimal Failure-Causing Schema of Combinatorial Testing. ACM Transactions on Software Engineering and Methodology, 2011, 20(4).

[9] Zhiqiang Zhang, Jian Zhang. Characterizing Failure-Causing Parameter Interactions by Adaptive Testing. In Proveedings of the International Symposium in Software Testing and Analysis (ISSTA2011), Toronto, ON, Canada, July 17-21, 2011: 331-341.

[10] Zhou Wu-Jie, Zhang De-Ping, Xu Baowen. Locating Error Interactions Based on Partial Covering Array. Chinese Journal of Computers, 2011, 34(6): 1126-1137.

[11] Zhou Wu-Jie, Zhang De-Ping, Xu Baowen. An Adaptive algorithm of Locating Fault Interactions Based on Combinatorial Testing. Chinese Journal of Computers, 2011, 34(8): 1509-1518.

[12] Jian Zhang, Feifei Ma, Zhiqiang Zhang. Faulty Interaction Identification via Constraint Solving and Optimization. In Proceedings of the 15th International Conference on Theory and Applications of Satisfiablity Testing (SAT2012), Lecture Notes in Computer Science, 2012(7317), Trento, Italy, june 17-20, 2012: 186-199.

[13] Laleh Shikh Gholamhossein Ghandehari, Yu Lei, Tao Xie, Richard Kuhn, Raghu Kacker. Identifying Failure-Inducing Combinations in a Combinatorial Test Set. In Proceedings of the 5th IEEE International Conference on Software Testing, Verification and Validation (ICST2012) , April 17-21, 2012: 370-379.

[14] Kiran Shakya, Tao Xie, Nuo Li, Yu Lei, Raghu kacker, Richard kuhn. Isolating Failure-Inducing Combinations in Combinatorial Testing Using Test Augmentation and Classification. In Proceedings of the 5th IEEE International Conference on Software Testing, Verification and Validation (ICST2012) , April 17-21, 2012: 620-623.

[15] Ziyuan Wang, Baowen Xu, Lin Chen, Lei Xu. Adaptive Interaction Fault Location Based on Combinatorial Testing. In Proceedings of the 10th International Conference on Quality Software (QSIC2010): 495-502.

[16] A. Zeller, R. Hildebrandt. Simplifying and Isolating Failure-Inducing Input. IEEE Transaction on Software Engineering, 2002, 28: 183-200.

[17] R. C. Bryce, C. J. Colbourn. A Density-Based Greedy Algorithm for Higher Strength Covering Arrays. Software Testing, Verification and Reliability, 2009, 19(1): 37-53.

[18] S. McConnell. Daily Build and Smoke Test. IEEE Software, 1996, 13(4).

Analyzing the Effectiveness of a System Testing Tool for Software Product Line Engineering

Crescencio Rodrigues Lima Neto[1,2,3], Ivan do Carmo Machado[2,3],
Vinicius Cardoso Garcia[5], Eduardo Santana de Almeida[2,3,4]

[1]Federal Institute of Bahia
[2]Computer Science Department - Federal University of Bahia (DCC/UFBA)
[3]Reuse in Software Engineering Labs (RiSE)
[4]Fraunhofer Project Center (FPC) for Software and Systems Engineering
[5]Center for Informatics - Federal University of Pernambuco (CIn/UFPE)
{crescencio, ivanmachado, esa}@dcc.ufba.br, vcg@cin.ufpe.br

***Abstract*—This paper reports on a serie of empirical evaluations of a proposed testing tool for SPL, in order to understand how useful it might be consider. The study is a serie of controlled experiment involving over fourty subjects, which analyzed the use of a proposed tool in an SPL project. The results show that the effort spent during the test case design decrease significantly when the tool was used. The subjects preferred to work with the tool, especially users without experience in testing. The collected insights are reported for further studies and the gathered data will serve as baseline values for future experiments, since there is a lack of work in this direction.**

***Keywords*—Software Testing; Software Product Lines; Software Reuse; Testing Tools; Empirical Evaluation**

I. Introduction

Automated tools are available to support testing in every stage of the software development life cycle [1]. Some organizations have used testing tools to manage, store and handle the tests, including their execution results. Nakagawa et al. [2] encourage the use of supporting tools to make testing a more systematic activity, which may lead to reductions in cost and time consumed, and also minimizing errors caused by human intervention.

However, when narrowing the observation scope to only consider testing tools for Software Product Line Engineering, the number of existing testing tools decrease drastically, leading to the need of tools to support testing in such a development paradigm, so as to enable practitioners to experience the same benefits, as the ones mentioned earlier.

Within the set of existing tools for SPL testing [3], either the reports lack details of empirical assessments that endorse their usefulness, or there is really few evidence that support in the large-scale use.

In earlier investigation, we developed SPLMT-TE, a tool to support the design of system test cases from SPL use cases [4], [5]. In this paper, we report on a serie of experimental studies performed in order to evaluate its usage effectiveness, namely an in-vitro experiment, and its replication, in a distinct environment.

The experimental study consisted of three phases. We initially performed a pilot study [6], aiming at analyzing the experiment feasibility. Then, we performed the actual experiment. Next, we carried out the replications.

The experimental study was conceived and structured based on the concepts of experimental software engineering and the evaluation of methods and tools provided by Wohlin et al. [7]. The replications were motivated by the desire of achieving more significance and confidence on the results [8].

The remainder of this paper is structured as follows. Section II introduces the proposed testing tool. Section III presents the experimental study. Section IV discusses the related work. Section V concludes the paper and presents future research.

II. Tool Support for System Testing in SPLE

In previous research [3], we carried out a mapping study of tools for testing software product lines, involving the investigation of thirty-three research papers, dated from 1999 to 2011. Among other goals, such a mapping aimed at identifying the functionalities supported by existing tools, and also to leverage others an SPL testing tool is expected to support. The findings of such an investigation served as input for the analysis and design of our proposal, called SPLMT-TE [4].

The SPLMT-TE was developed using Django, a Python Web Framework[1]. The tool supports the design of system test cases from use case, and manages assets such as test cases, suites, and plans. It intends to reduce the effort and the cost associated to the SPL testing process. We performed an initial evaluation in [5] to assess the tool's effectiveness.

III. Experimental Study

This Section details the experimental study setup and measurements used to investigate the effects of using our proposed tool. We reported the study by following the guidelines for conducting experiments in software engineering defined by Wohlin et al. [7], and reported as suggested by the guidelines described in [9].

[1]http://www.djangoproject.com/

A. Experimental Planning

We applied the GQM method [10] in order to define quantitative measures that could provide objective assessment of the tool's effectiveness. It was structured as follows:

1) Goal.: The objective of this experimental study is to analyze *the SPLMT-TE tool* for the purpose of *evaluation* with respect to its *efficiency* and *effectiveness* from the point of view of the *potential users (testers)* in the context of *an SPL testing project in an academic environment*.

2) Questions.: To achieve this goal, we used the following questions defined in [5] and we added the question Q6:

- **Q1.** *Is the amount of test cases increased when the tool is used?* [5]
- **Q2.** *Is the time required to design system test cases reduced when the tool is used?* [5]
- **Q3.** *Is the time required to execute the designed test cases reduced when the tool is used?* [5]
- **Q4.** *Is the amount of errors detected increased when the tool is used?* [5]
- **Q5.** *Is the effectiveness of test cases improved when the tool is used?* [5]
- **Q6.** *Is the time required to find errors improved when the tool is used?*

3) Metrics.: After defining the questions, they need to be mapped to a measurement value, in order to characterize and manipulate the attributes in a formal way [11]. Hence, the metrics used in this analysis are next described next:

- **M1.** *Designed Test Cases (DTC)*[12], [5];
- **M2.** *Efficiency in Test Case Design (ETCD)*[12], [5];
- **M3.** *Efficiency of Test Cases Execution (ETCE)*[12], [5];
- **M4.** *Number of Errors Found (NEF)* [12], [5];
- **M5.** *Test Cases Effectiveness (TCE)* [13], [5];
- **M6.** *Efficiency in Finding Errors (EFE):* It refers to number of errors found (**M4**) reported over the amount of time spent to execute test cases *(TSE)*.

$$EFE = \frac{NEF}{TSE} \quad (1)$$

4) Design.: In this experimental study, we applied *One factor with two treatments*, as the design type. We compared the two treatments against each other [7]. In this sense, the factor was the use of the proposed tool, and the treatments were: (1) *Creating test cases with the tool support* and (2) *Create test cases manually.* The subjects were divided in two groups, each one addressing a treatment.

5) Experiment materials.: The experiment used Consent Form, Background and Feedback Questionnaires, a set of Test Assets, Use Cases, Test Case Creation Form, Test Case Execution Form, Defect Reporting Form, application to be tested and the proposed tool.

6) Subjects.: We applied *convenience sampling* [7] in this experimental study. Initially, we performed a pilot with fourteen students from the Software Engineering course at Federal University of Bahia, Brazil. Next, the subjects were graduate students (seven M.Sc. Students and five Ph.D. Students) from Federal University of Bahia and Federal University of Pernambuco, Brazil. Then, additional nineteen students from the Software Engineering course at Federal University of Bahia participated in the experiment replications.

7) Hypotheses.: We set up six hypotheses for this experimental study. The Null Hypothesis (H_{0n}) considered that there was no benefit of using the proposed tool.

Conversely, the Alternative Hypothesis (H_{1n}) stated the opposite values. The alternative hypothesis determined that the proposed tool produced benefits that justify its use.

We used the same hypotheses defined in [5]. Moreover, we added the Null Hypothesis $H_{06} : \mu_{EFE_{manual}} = \mu_{EFE_{Tool}}$ and the Alternative $H_{16} : \mu_{EFE_{manual}} \neq \mu_{EFE_{Tool}}$

8) The Experimental Study Project.: We selected two projects for the experimental study. The first one consisted of a product line in the mobile devices games domain called Arcade Game Maker (AGM) Pedagogical Product Line[2] produced by the SEI[3]. The second project consisted of the NotepadSPL, a product line built upon the well-known word processor Notepad. This product line was extracted from the FeatureVisu[4].

B. Execution

The subjects carried out the activities involved in the experiment. When the subjects filled out the background questionnaire, we could organize them into two groups. The balancing strategy was applied to harmonize the groups in terms of expertise. Both groups performed the experiment using the tool in one phase and without using the tool in the other phase.

1) Procedure.: The experiment took place at Federal University of Bahia, Brazil in July of 2011. The replications were also carried out at the same place, namely in November 2011, and in June 2012.

Initially the subjects became aware of the experiment purpose and associated tasks. Next, training sessions were performed with practical exercises.

The first training session consisted of theoretical and practical classes on fundamentals of software testing. Both groups learned how to create a good test case, how to execute test cases, and how to report errors. It was not necessary to offer training classes on software product line engineering, given that all subjects had been involved in research in such a field.

Next training session involved the use of the SPLMT-TE tool. At the end, they had to fill out the feedback questionnaire. This process was also performed at the pilot study and replications execution.

2) Operation.: All subjects were asked to fill out a background questionnaire, as the initial task in the experiment. It served for us to investigate their experience. It was comprised of a set of items, described next:

- Participation in industrial development/testing

[2]http://www.sei.cmu.edu/productlines/ppl/index.html
[3]http://www.sei.cmu.edu
[4]http://fosd.de/FeatureVisu

- Experience in programming
- Experience with testing tools

The execution was performed in two phases. One group used the tool while the other worked without tool support. The pilot study execution was limited to 2 hours of test cases creation and execution because of the discipline schedule. However, at the experiment and replications, we did not restrict the time of creation and execution.

The subjects were instructed to analyze the available components, create, and execute the test cases. Their assigned tasks were: (1) to analyze the available use case document; (2) to build test, i.e., writing the test cases manually or using the proposed tool; (3) to execute them manually step by step; and (4) to report the findings in the proper form. The replications focused only at the last two steps: test execution and reports.

At the end of the experiment, each participant completed the feedback questionnaires. The complete characterization of the subjects, as well as supplemental material of this experimental study such as complete descriptive statistics, questionnaires, reports, and forms can be reached in the paper's resource website[5].

C. Analysis and Interpretation

In [5], we used the wrong hypothesis testing in some descriptive statistic analysis. For this reason, we analyzed all the artifacts produced by the subjects again, including the error report forms and the feedback questionnaires. The new analysis was performed based on descriptive statistics and hypothesis testing using t-test [7] (for parametric statistical hypothesis test) and Mann-Whitney-Wilcoxon [7] (for non-parametric statistical hypothesis test).

Since the replications' subjects had the same profile, undergraduate students from a Software Engineering course inexperienced in testing, the two replications data were combined and analyzed in the same dataset.

Descriptive Statistic

1) Designed Test Cases.: According to Figure 1(a), the amount of designed test cases created with the tool is higher than without tool support. Moreover, since the subjects were inexperienced in testing, we decided not to consider this measure in the replications.

2) Efficiency in Test Case Design.: Figure 1(b) presents that the efficiency in test case design was higher with groups that used the tool, which enabled the creation of more test cases faster than without tool support. Hence, as this metric is related with the previous metric, we preferred not to consider it for the same reason.

3) Efficiency in Test Cases Execution.: Figure 1(c) presents that during the first experiment execution, the effort of test case execution were almost the same. On the other hand, during the replications, the tests created by the tool were more efficient than the tests created manually (see Figure2(a)).

[5]http://www.crescenciolima.com/seke2013/

4) Number of Errors Found.: Figure 1(d) presentes that the subjects were capable to find more errors during the use of the tool. Figure 2(b) confirms that the number of errors found were higher when the subjects used the tests created by the tool.

5) Test Cases Effectiveness.: Figure 1(e) presents that during the experiment, the test case effectiveness was practically the same using the tool or not. The number of errors found per number of test cases was almost the same using and without using the tool. Which can be confirmed by the replications and it can be seen in Figure 2(c).

6) Efficiency in Finding Errors.: Figure 1(f) presents the amount of errors that were found in the experiment execution. The efficiency was practically the same between the amount of errors found using the tests generated by the tool and manually.

On the other hand, during the replications more errors were found faster when the tests created by the tool were used (see Figure 2(d)).

Hypotheses Testing

Since the experiment has one factor with two treatments, completely randomized design, the data collected during the experiment were submitted to descriptive statistic examination. Firstly, in order to reduce the dataset errors, we eliminated the outliers. Secondly, we verified if the sample came from a normally distributed population through Shapiro-Wilk test.

Thirdly, when the sample was not "normal", we used the non-parametric test (Mann-Whitney-Wilcoxon). On the other hand, when the sample was "normal", we verified if it had equal variance through Levene test. Finally, when the sample had different variance, we used non-parametric test (Mann-Whitney-Wilcoxon), and, when the sample had equal variance we used parametric test (t-test).

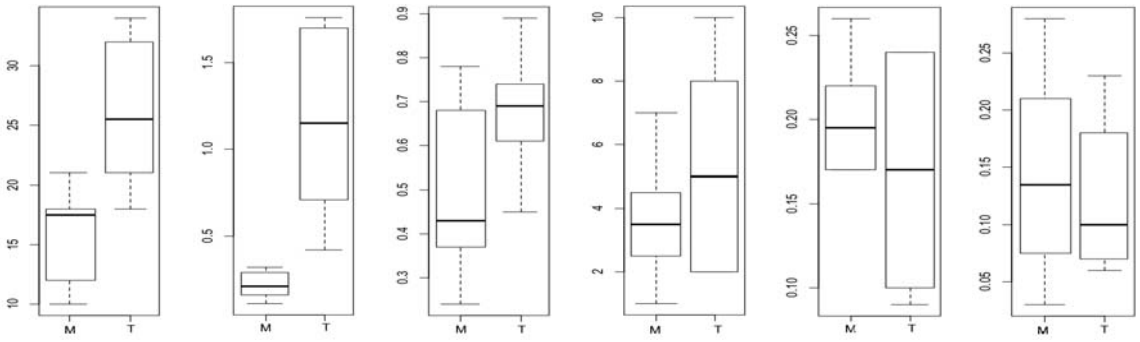
The tests were primarily presented for a significance level of 5%. Table I presents the hypothesis testing results from the experiment and replications. The results are detailed next.

TABLE I
HYPOTHESES TESTING RESULTS

RQ*	Metric	Study	Test	p-value	StD?**
Q1	DTC	Experiment	Wilcoxon	0,0007	Yes
Q2	ETCD	Experiment	Wilcoxon	0,0001	Yes
Q3	ETCE	Experiment	t-test	0,03	Yes
Q4	NEF	Experiment	Wilcoxon	0,35	No
Q5	TCE	Experiment	Wilcoxon	0,34	No
Q6	EFE	Experiment	t-test	0,7	No
Q3	ETCE	Replications	t-test	0,04	Yes
Q4	NEF	Replications	t-test	0,058	Near 0,05
Q5	TCE	Replications	t-test	0,5	No
Q6	EFE	Replications	t-test	0,03	Yes

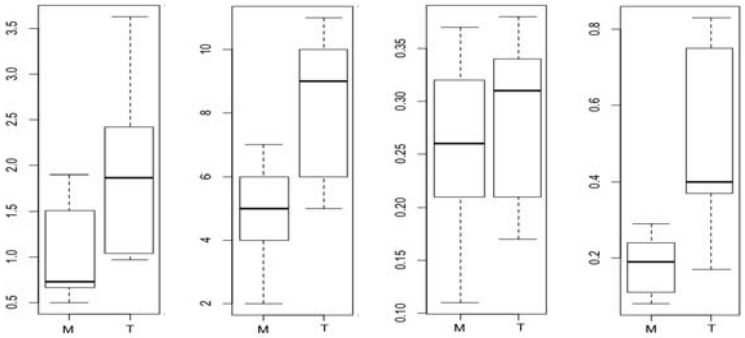
*Research Question **Statistical Diference?

Regarding Q1, the amount of test cases created using the tool is higher than without using it, *p-value* = 0.0007. The *p-value* is smaller than the significance level, rejecting the null hypothesis. **Null Hypothesis** H_{01} is rejected, since H_{01} : $\mu_{DTC_{manual}} \neq \mu_{DTC_{Tool}}$.



(a) DTC [5] (b) ETCD [5] (c) ETCE (d) NEF [5] (e) TCE (f) EFE

Fig. 1. Experiments Bloxplots (T - means tool support, M - means manual)



(a) ETCE (b) NEF (c) TCE (d) EFE

Fig. 2. Replications Bloxplots (T - means tool support, M - means manual)

Time spent creating test cases without the tool is higher than using it. For this reason, in order to answer the Q2, the **Null Hypothesis** H_{02} is rejected, since $H_{02} : \mu_{ETCD_{manual}} \neq \mu_{ETCD_{Tool}}$, *p-value* = 0.0001. This *p-value* allowed the rejection with high significance level.

Thus, the efficiency in test case execution supports the **Null Hypothesis** H_{03}. The question Q3 is answered, *p-value* = 0.03. As the *p-value* is lower than 0.05 the null hypothesis was rejected. The same result is confimed in the replications, *p-value* = 0.04.

The **Null Hypothesis** H_{04} cannot be rejected, since there is no significant difference $H_{04} : \mu_{NEF_{manual}} = \mu_{NEF_{Tool}}$, although the number of errors found were higher using the tool support. The experiment *p-value* = 0.35 and the replication *p-value* = 0.058, are higher than 5%, which did not reject the null hypothesis, answering the Q4.

As a result, the **Null Hypothesis** H_{05} cannot be rejected, since $H_{05} : \mu_{TCE_{manual}} = \mu_{TCE_{Tool}}$, the experiment *p-value* = 0.34 and replication *p-value* = 0.5. Since the *p-value* are higher than the significance level the hypothesis cannot be rejected and no conclusion can be drawn. Regarding Q5, there is no significant differences between the effectiveness test cases values during the tool usage and without use it.

Finally, regarding the Q6, the efficiency in finding erros supports the **Null Hypothesis** H_{06}. The experiment *p-value* = 0.7, cannot reject the null hypothesis. On the other hand, the replications *p-value* = 0.03 is lower than the significance level, rejecting the Null Hypothesis.

D. Evaluation of results and implications

The results presented insights that enable us to consider that activities of test case creation were more productive when the tool was used as we can seen in the hypotheses H_{01} and H_{02}. For this reason, we focused the replication analysis in the null hypotheses that could not be rejected.

During the replications, we identified that in some aspects, subjects without experience were more productive than the experienced ones. In other words, the tool could have hampered the performance of the subjects with experience in testing. Although, almost all the subjects answered the feedback questionnaire saying they preferred to work with the tool.

During study execution, we observed the importance of splitting the design and execution activities. Ideally, these activities should be performed by different subjects.

Perhaps, as the subjects gain confidence on the SPLMT-TE usage (depending on the time availability, to try different scenarios), the results might become better. But it is solely an assumption that should be tested, maybe applying in a larger context.

E. Threats to Validity

1) Maturation.: This is the effect that subjects react differently as time passes. Some subjects can be affected negatively during the experiment, and their performance may be below normal. In order to mitigate this boredom, two familiar domains were provided.

2) Gained Experience.: There were two scenarios: in the first phase group 1 performed the task using tool support, and in the second phase, without the tool. In the second scenario, group 2 executed the first phase manually, and in the second phase with tool support. To mitigate this risk, two distinct problems were analyzed.

3) Experimenter Expectations.: Surely the experimenter expectations may bias the results, and for that reason, all formal definition and planning of the experiment was carefully designed by the first author of this paper, and revised by the two remaining authors. We also had support of two senior researchers in the field of Software Product Line Engineering.

4) Generalization of subjects.: This is an effect of having a subject population not representative of the population we would like to generalize to. The replications were conducted with undergraduate students without knowledge about software testing. In this case, if these subjects succeed using the tool support, we cannot conclude that an experienced testing analyst would use it successfully too.

IV. Related Work

As we mentioned before, there is few empirical evaluations of testing tools for SPL. According to [14], there is a lack of studies reporting on empirical evaluations in SPL testing, more specific in a testing tools viewpoint.

It is even harder to find empirical studies that evaluate SPL testing tools. From the papers we analyzed in the previous work [3], we found only one study considered as a similar study [15].

Nogueira et al. [15] presented the Test and Requirements Generation Tool (TaRGeT), which is a tool for automatic test case generation from use case scenarios written in Natural Language (NL).

V. Concluding Remarks and Future Work

This paper presented the experimental studies conducted to evaluate the proposed tool. It included the definition, planning, operation, analysis and interpretation of a pilot study, experiment, and replications. The guidelines defined by [7] were used to perform the experiment.

The evaluations analyzed the SPLMT-TE effectiveness and efficiency, in order to gather empirical evidence. As one of the findings we had after performing this study, we observed that, for users with experience in testing, it could be more complicated to work with the tool support.

Probably, for these subjects, there was no gain using the tool to save time during the execution of test cases. In the analysis of test case effectiveness, it was not possible to reject the null hypothesis, concluding that there is no advantage in using the tool.

On the other hand, the tool proved to raise the productivity of subjects without experience in testing. Moreover, the number of test cases created and the number of errors found were higher when the subjects used the proposed tool. Finally, the effort spent during the test case design decrease when the tool was used.

As future work, we plan to compare the proposed tool with some other testing tools, which is a difficult task since is necessary to adapt the tool use to SPL. We also intend to replicate this study in other environments.

Acknowledgment

This work was partially supported by the National Institute of Science and Technology for Software Engineering (INES), funded by CNPq and FACEPE, grants 573964/2008-4 and APQ-1044-1.03/10 and APQ-1037-1.03/08 and CNPq grants 305968/2010-6, 559997/2010-8, 474766/2010-1 and FAPESB.

References

[1] M. Fewster and D. Graham, *Software Test Automation: Effective Use of Test Execution Tools*. John Wiley Sons, Ltd., 1999, vol. 10, no. 2.

[2] E. Y. Nakagawa, A. S. Simão, F. C. Ferrari, and J. C. Maldonado, "Towards a Reference Architecture for Software Testing Tools," in *International Conference on Software Engineering & Knowledge Engineering*, 2007, pp. 157–162.

[3] C. R. L. Neto, P. A. M. S. Neto, E. S. Almeida, and S. R. L. Meira, "A Mapping Study on Software Product Lines Testing Tools," in *International Conference on Software Engineering & Knowledge Engineering*, 2012, pp. 628–634.

[4] C. R. L. Neto, I. C. Machado, P. A. M. S. Neto, E. S. Almeida, and S. R. L. Meira, "Software Product Lines System Test Case Tool: A Proposal," in *International Conference on Software Engineering & Knowledge Engineering*, 2011, pp. 699–704.

[5] C. R. L. Neto, E. S. Almeida, and S. R. L. Meira, "A Software Product Lines System Test Case Tool and its Initial Evaluation," in *13th International Conference on Information Reuse and Integration*, 2012, pp. 25–32.

[6] M. Mendonça, D. Cruzes, J. Dias, and M. C. F. de Oliveira, "Using observational pilot studies to test and improve lab packages," in *International Symposium on Empirical Software Engineering*. New York, NY, USA: ACM, 2006, pp. 48–57.

[7] C. Wohlin, P. Runeson, M. C. O. Martin Höst, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering: An Introduction*, ser. The Kluwer Internation Series in Software Engineering, V. R. Basili, Ed. Norwell, Massachusets, USA: Kluwer Academic Publishers, 2000.

[8] N. J. Juzgado and A. M. Moreno, *Basics of Software Engineering Experimentation.* Kluwer, 2001.

[9] A. Jedlitschka and M. Ciolkowski, "Reporting Experiments in Software Engineering," pp. 201–228, 2008.

[10] V. R. Basili, "Software Modeling and Measurement: the Goal/Question/Metric Paradigm," College Park, MD, USA, Tech. Rep. CS-TR-2956, 1992.

[11] V. Basili, R. Selby, and D. Hutchens, "Experimentation in Software Engineering," *IEEE Transactions on Software Engineering*, vol. 12, no. 7, pp. 733–743, July 1986.

[12] N. J. Juzgado, A. M. Moreno, and S. Vegas, "Reviewing 25 Years of Testing Technique Experiments," *Empirical Software Engineering*, vol. 9, no. 1-2, pp. 7–44, 2004.

[13] Y. Chernak, "Validating and Improving Test-Case Effectiveness," *IEEE Softw.*, vol. 18, pp. 81–86, January 2001.

[14] I. C. Machado, E. S. Almeida, G. S. S. Gomes, P. A. M. S. Neto, R. L. Novais, and M. G. de Mendonça Neto, "A preliminary study on the effects of working with a testing process in software product line projects," in *Ibero-American Conference on Software Engineering*, 2012.

[15] S. Nogueira, E. Cartaxo, D. Torres, E. Aranha, and R. Marques, "Model Based Test Generation: An Industrial Experience," in *1st Brazilian Workshop on Systematic and Automated Software Testing*, Oct. 2007.

Exploiting Weights of Test Cases to Enhance Fault Localization

Yihan Li, Chao Liu, Zi Yuan
School of Computer Science and Engineering
Beihang University
Beijing, China
{liyihannew, liuchao, yuanzi}@sei.buaa.edu.cn

***Abstract*—Spectrum based fault localization techniques such as Tarantula and Ochiai have been proposed to guide developers to the locations of faults. These techniques make a summary on the number of failing and passing test cases to calculate the suspiciousness score of a program statement. Though results are promising, these techniques assume all test cases share equal importance, which ignore individual error diagnosis ability for different test cases. In this paper, we present an approach to improve the effectiveness of some existing spectrum based fault localization techniques by exploiting varying weights for different test cases in the computation of suspiciousness scores. Evaluations on Siemens Test Suits demonstrate that the refined fault localization techniques using our approach are significantly more effective than the original techniques.**

Keywords-fault localization; program spectrum; debugging; passed tests; failed tests

I. INTRODUCTION

It is a common phenomenon that developers more or less introduce bugs during software development processes despite the fact that developers may spend much effort on coding and testing. When faults are revealed during testing process, software debugging is employed to remove bugs so as to improve quality of software. Typically, software debugging involves locating faults, repairing faults and verifying repairs. However, according to a previous study, locating faults is one of the most expensive tasks in debugging [1]. Therefore, in order to reduce the cost of debugging, a variety of fault localization techniques has been proposed to alleviate the work developers devoted to locating faults in recent years [2][6][10][11].

In particular, Spectrum-base fault localization (SFL) techniques are effectiveness [3][5]. The basic idea behind them is that they exploit how program entities are correlated with program failure via statistically analyzing coverage information. Specifically, SFL usually collect coverage information of a set of test cases and their corresponding test results to form program spectrum, and then contrast the coverage statistics of program entities between passed executions and failed executions using different statistical formula to produce a ranking list of all program entities in terms of suspiciousness. Those program entities with higher suspiciousness are more likely to contain bugs and thus are given higher priority during software debugging. Following this idea, many researchers have proposed different statistical formulas to measure program entities to be faulty based on coverage information, such as Tarantula [2], Ochiai [4] and Jaccard [4].

Although SFL approaches have brought encouraging results in locating faults, these approaches only use information of the number of failing and passed test cases and thus treat all test cases as equally important, which ignore individual error diagnosis ability for different test cases and may reduce effectiveness of the techniques. Take Tarantula as an example. If two statements are both exercised by the same number of passed tests and the same number of failed tests, the technique assigns same suspiciousness score to these two statements and therefore loses power to distinguish such statements, which may decrease its accuracy in predicting the locations of bugs. Furthermore, during testing, a statement is often executed by different number of passed tests and failed tests. The number of failed tests is often relatively smaller than that of passed tests and there often exists redundancy in passed tests. Due to such distribution of test cases, the contribution of the statement to failure will also be decreased largely even if the statement contains fault, which may yield misleading results in debugging.

In this paper, we propose an approach to measure weights for different test cases, to mitigating noisy introduced by tests. Our approach treats failed and passed tests separately. For each failed test, different weights are assigned with respect to each statement according to its coverage information and relative size of tests associated for that statement. For each passed test, since there is no definite information about fault, we simply assign same weight to all passed tests. In such way, our approach can be applied as a refinement method to some existing SFL techniques. In this paper, four representative techniques are studied, i.e.,Tarantula [2], Jaccard [4], Ochiai [4], and SBI [8]. Based on these four techniques, four corresponding revised techniques are constructed. We use seven Siemens programs to evaluate our revised techniques, and compare effectiveness with original techniques. The empirical results show that our approach is promising on the studied subject programs. Further analysis shows that for the four SFL techniques, our approach can significantly improves their performance. The main contributions of this paper are as follows:

(1) We propose an approach to quantify weights for different test cases according to coverage information and associated test cases for different statements and incorporated the approach into four existing representative techniques.

(2) We verify the effectiveness of our techniques with our experiment and compare refined techniques with other techniques based on program spectrum.

II. PROPOSED APPROACH

Typically, failed tests present definite information about program behavior that fault is activated and propagated to program failure, while passed tests do not guarantee that fault is activated. This implies that contribution for different types of tests to failure should be considered separately in terms of error diagnosis. The current popular SFL techniques often utilize four spectrum parameters denoted as $<a_{ef}, a_{nf}, a_{ep}, a_{np}>$ for each statement to calculate suspiciousness score using different statistical formulas. The four spectrum parameters record the number of tests with respect to a statement, where the first part of the subscript indicates whether the statement is executed (*e*) or not (*n*) and the second indicates whether the test passed (*p*) or failed (*f*). Similar to this, in this paper, four different types of weights with respect to each statement are defined as follows:

$N_{ef}(s)$: the weights of failed test cases that cover statement s

$N_{nf}(s)$: the weights of failed test cases that do not cover statement s

$N_{ep}(s)$: the weights of successful test cases that cover statement s

$N_{np}(s)$: the weights of successful test cases that do not cover statement s

The computation of the four weights for each statement will be presented in the following sections. For the convenience of following discussion, let us suppose a program P consists of n executable statements, which are denoted as $P=\{s_1,s_2,s_3,\ldots,s_n\}$, and m basic blocks, which are denoted as $P=\{b_1,b_2,b_3,\ldots,b_m\}$. Also consider that the program P is tested against a test suite T, which comprises of w different test cases that are denoted as $T=\{t_1,t_2,t_3,\ldots,t_w\}$. The test suite T can be partitioned into two disjoint subsets T_p and T_f according to the passed/failed status of test cases.

A. Weights of Failed Tests

To compute $N_{ef}(s)$ and $N_{nf}(s)$ for each statement, we first discuss two practical test scenarios that motivate us to distinguish weights for failed test, and two equations that capture characteristic of weight are defined respectively. Then these equations are combined to measure different weights of failed tests with respect to each statement.

1) *Weight of coverage of test with respect to each statement.* Consider two tests both result in failure while one of them executes less basic blocks. When measuring fault locating ability for test case, the test that exercises less basic blocks explicitly demonstrate more power to find fault than the other one, since less basic blocks require less code examination for debugging. We use basic blocks rather than statements to capture this characteristic because statements within a basic block cannot be distinguished by tests.

Let a test case t_i be one element of T_f and cover statement s. Spectra of basic block information for t_i are collected, which is recorded as a binary vector $<b_{1i},b_{2i},b_{3i},\ldots,b_{mi}>$. If block b_j is covered by test t_i, then the value for b_{ji} in the vector is 1 otherwise 0. After collecting basic block information, we computer weight of test case t_i about its execution that contributes to statement s using equation (1):

$$E_{ci}(s) = m/\sum_{r=1}^{m} b_{ri} \quad (1)$$

The greater the E_{ci} is, the more important is the test case for debugging. If a failed test executes only one basic block, the value approach maximum. In such scenario, the test case can provide significant information for debugging since developer can right now discover location of fault with the aid of the failed test. When a failed test executes all basic blocks in the program, it provides relatively less information to aid in locating fault.

Similar to eq. (1), for a statement s that is not covered by a failed test t_i, eq. (2) computes the contribution that the failed test subjects the statement s not to be a faulty statement. A small constant (0.001 in this study) is used in the denominator to avoid division by zero when all of the basic blocks are executed in a failing test.

$$E_{ni}(s) = m/\left(m - \sum_{r=1}^{m} b_{ri} + 0.001\right) \quad (2)$$

2) *Weight of status of tests with respect to each statement.* In testing process, the number of passing tests is often relatively larger than that of failing tests and redundancy may exist in the passing tests. Due to such distributions of tests, the effect of failing tests for error diagnosis is reduced, which may degrade effectiveness of SFL. In this study, we propose an information quantity based strategy to reduce imbalance between sizes of passing and failing tests. Specifically, let an event be "a statement is covered by (h+k) tests, of which the number of failing test is k". The information quantity of the event can be represented as: -log(k/(h+k)) = log((h+k)/k). A smaller probability the event occurs with, a larger information quantity the event has. Thus, the information quantity of the event is used as weight for every k failing tests. Based on this idea, we compute the weight of every failed test t_i that covers statement s using equation (3):

$$R_{ci}(s) = \log((\mathrm{h}+\mathrm{k})/\mathrm{k}+1) \quad (3)$$

A constant (1 in this study) is added in the equation to ensure that weight of a failing test is always greater than 1, which is weight of a passing test that will be discussed later. The formula dynamically determines weight for every failed test according to size relation between passed tests and failed tests. A great value means that the failed information is rare for statement s and weight of failed tests are adjusted for better error diagnosis. Such case often happens when a statement is executed by many passed test cases and a few failed test cases, which is common in test process. In such cases, passed test cases may be already redundant for the statement while failed test cases are lacking for the statement, which implicitly indicate that we should pay relatively more attention to such failed test cases owing to redundancy of passing tests.

For a statement s that is not executed in a failed test t_i, the weight of the test for statement s is also calculated using equation (4):

$$R_{ni}(s) = \log((|T_p| - \mathrm{h} + |T_f| - \mathrm{k})/(|T_f| - \mathrm{k}) + 1) \quad (4)$$

TABLE 1. INVESTIGATED TECHNIQUES

Name	Original Formula	Revised Formula
Tarantula	$\frac{\frac{a_{ef}}{a_{ef}+a_{nf}}}{\frac{a_{ef}}{a_{ef}+a_{nf}}+\frac{a_{ep}}{a_{ep}+a_{np}}}$	$\frac{\frac{N_{ef}}{N_{ef}+N_{nf}}}{\frac{N_{ef}}{N_{ef}+N_{nf}}+\frac{N_{ep}}{N_{ep}+N_{np}}}$
Jaccard	$\frac{a_{ef}}{a_{ef}+a_{nf}+a_{ep}}$	$\frac{N_{ef}}{N_{ef}+N_{nf}+N_{ep}}$
Ochiai	$\frac{a_{ef}}{\sqrt{(a_{ef}+a_{nf})*(a_{ef}+a_{ep})}}$	$\frac{N_{ef}}{\sqrt{(N_{ef}+N_{nf})*(N_{ef}+N_{ep})}}$
SBI	$\frac{a_{ef}}{a_{ef}+a_{ep}}$	$\frac{N_{ef}}{N_{ef}+N_{ep}}$

3) *Combined to measure weights of failed tests.* We use the above four equations to compute $N_{ef}(s)$ and $N_{nf}(s)$ of failed tests with respect to every statement in program as follows.

For a failing test t_i that covers a statement s, the weight of the test to the statement is computed as: $E_{ci}(s)*R_{ci}(s)$. Thus, eq. (5) is used to compute weights of a failed test set V all of which covers the statement s by summing over weight of each test case in V.

$$N_{ef}(s) = \sum_{t_i \in V} E_{ci}(s) * R_{ci}(s) \quad (5)$$

Similarly, for a failing test t_i that does not cover a statement s, the weight of the test to the statement is computed as: $E_{ni}(s)*R_{ni}(s)$. Therefore, eq. (6) is used to compute weights of a failed test set U all of which do not cover the statement s by summing over weight of each test case in U.

$$N_{nf}(s) = \sum_{t_i \in U} E_{ni}(s) * R_{ni}(s) \quad (6)$$

B. Weights of Passed Tests

For passed tests, since developers often cannot make sure about whether statements that are executed in passed tests contain fault or not, we simply assign equal weight to each passed test. In this paper, 1 is used for weight of each passed test such that weights of passed tests can be calculated easily using the number of tests that cover or not cover the statements.

For every statement s and a passed test set *C* all of which covers the statement s, the weights of test set *C* for statement s are recorded in N_{ep}, which is summed over each test case in *C* as equation (7):

$$N_{ep} = |C| \quad (7)$$

Similarly, for every statement s and a passed test set *N* all of which does not covers the statement s, a term N_{np} is recorded, which is summed over each test case in *N* as equation (8):

$$N_{np} = |N| \quad (8)$$

C. Refining the existing techniques

After the above four weights are collected for each statement in program, our proposed method can be applied into some existing spectrum-based fault localization techniques as a refinement to calculate suspiciousness scores for every statement. In this paper, we investigate four representative techniques, i.e., Tarantula [2], Jaccard [4], Ochiai [4], and SBI [8]. Table 1 shows original and revised formulas for four techniques. The spectrum parameters $<a_{ef}, a_{nf}, a_{ep}, a_{np}>$ in original formulas are respectively substituted by $<N_{ef}, N_{nf}, N_{ep}, N_{np}>$ to generate new statistical formulas.

TABLE 2. STATISTICS OF SUBJECTS TABLE TYPE STYLES

Subjects	# of faulty versions	# of test cases	Description
print_tokens	4	4130	Lexical analyzer
Print_tokens2	10	4115	Lexical analyzer
replace	27	5540	Pattern replacement
schedule	4	2650	Priority scheduler
Schedule2	9	2710	Priority scheduler
tcas	35	1608	Altitude separation
Tot_info	23	1052	Information measure

III. EXPERIMENTAL RESULT AND ANALYSIS

In this section we report details on our experiments conducted to evaluate our proposed approaches across Siemens test suite.

A. Experiments Setup

The Siemens test suite has been used widely in previous studies [3][4][5][8][10][11] to measure the effectiveness of fault localization techniques. In this paper, we also used Siemens test suite as our subject programs and obtained them from the website [9]. The test suite contains 7 types of programs. For each program, there are a variety of test cases and faulty versions available. For simplicity, only single fault versions are considered in this study. Buggy versions that do not yield failed information on the provided test suite are also excluded. After removing those versions, 112 faulty programs are used to evaluate our techniques in total. Table 2 lists the statistics of the subject programs used in the experiments.

In our experiments, we select four representative techniques Tarantula [2], Jaccard [4], Ochiai [4] and SBI [8] to be compared with their refinement approach using our proposed method. Additionally, comparisons are also made with the weighting method proposed by Lee Naish et al. [10]. They proposed a weighting function for failed tests and considered weight of a failed test is inversely proportional to the number of statements exercised in the test. We implemented their weighting method on the four representative techniques ourselves and carefully examined the weighting function to assure it is strictly consistent with that in [10].

Table 1 lists the four techniques and their respective revised formula using our method. The original formula and revised formula are respectively evaluated against Siemens test suite in terms of effectiveness of fault localization. We use the metric Expense proposed by Yu [3] to measure the effectiveness of the fault localization technique. The Expense metric is computed by the percentage of the program that must be examined to find the fault following rank list from top to down. In case of tie when two or more statements share same suspiciousness score, we adopt the worst cases. That is, developers have to examine all the tie statements to find the faulty statement.

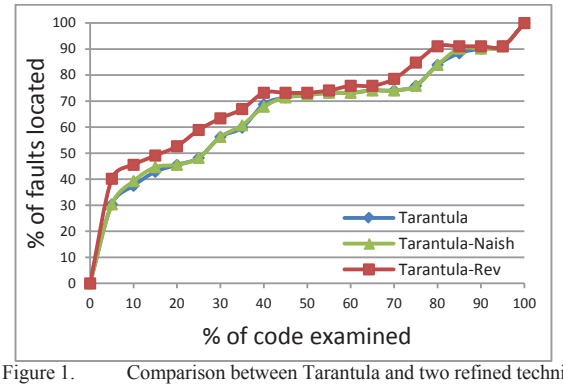


Figure 1. Comparison between Tarantula and two refined techniques

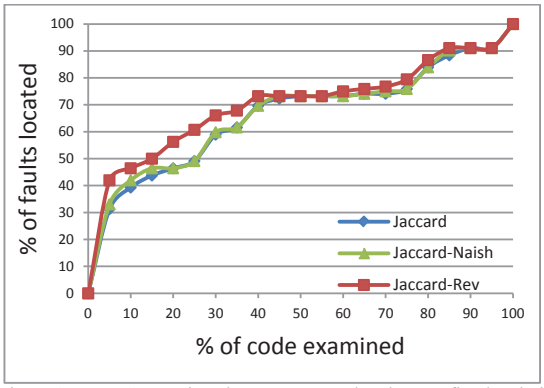


Figure 2. Comparison between Jaccard and two refined techniques

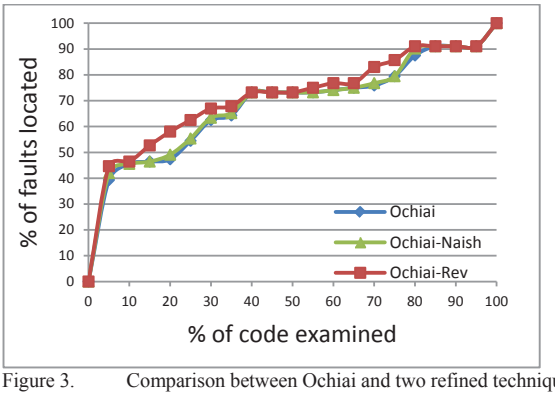


Figure 3. Comparison between Ochiai and two refined techniques

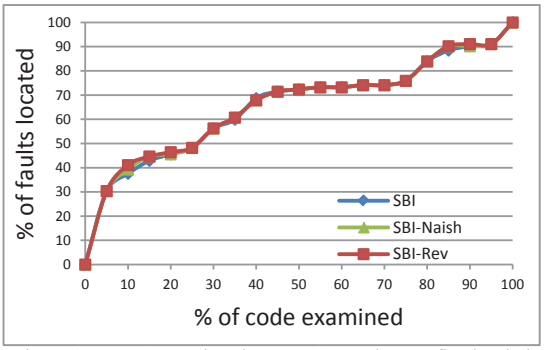


Figure 4. Comparison between SBI and two refined techniques

B. *RESULT AND ANALYSIS*

In our experiments, we apply the refinement method to all the formulas listed in Table 1, investigating its effectiveness in improving the performance of these SFL techniques. We refer to four refinement techniques using our approach as Tarantula-Rev, Jaccard-Rev, Ochiai-Rev and SBI-Rev respectively. For the four techniques that enhanced by methods proposed by Lee Naish et al. [10], their names are suffixed by Naish respectively. Fig. 1 to Fig. 4 illustrates effectiveness between original techniques and corresponding two refinement approaches in all faulty versions. For all four figures, the x-axis represents the percentage of executable statements to be examined. The y-axis denotes the percentage of faulty versions whose faults have been located by examining no more than corresponding percentage of executable statements in x-axis.

From Fig. 1 to Fig. 4, we observe that all the four revised formulas using our approach achieve better performance than, or at least equal with the other two types of SFL techniques in terms of effectiveness of fault localization. The curves for four revised formulas are always higher than the other two. Take Tarantula as an example. By examining up to 5% of all the code in faulty versions, the revised Tarantula technique can locate faults in 40.2% of all faulty versions while the original Tarantula can only locate 30.3% faults and Tarantula refined by Lee Naish et al. [10] can locate 31.2% faults. On average, for the four techniques, the revised technique can locate 20.5% more faults than original technique, and 15.7% more faults than technique refined by Lee Naish et al. [10] when examining no more than 5% of all the code. The gain of performance varies in different techniques. Tarantula, Jaccard and Ochiai achieve similar improvement on performance using our approach while SBI achieve a little improvement on performance. In summary, the figures show that the four revised formulas can be more effectively and precisely in locating faults than original techniques studied and techniques refined by Lee Naish et al.

Table 3 shows the mean effectiveness of these techniques on each program. The lower the value is, the better performance the technique achieves. This table shows that for these programs, our proposed weighting method can be more precise to locate faults than the other two.

IV. RELATED WORK

In this section, we briefly review previous studies related to fault localization.

The Tarantula system [2] colored the statements to highlight the particular statements that contain bugs. It used the number of successful tests and failed tests to locate buggy statements. The intuition is that the more failed tests and the less successful tests cover a statement, the more likelihood for the statement to be faulty. Different colors for different suspicious statements are then assigned to visualize program codes. Therefore a buggy statement is colored as red for highlighting so that developers can focus on it directly. Empirical evaluation [3] showed that Tarantula consistently outperforms four other techniques—Set union [6], Set intersection [6], Nearest Neighbor [6], and Cause Transitions [12].

Abreu et al. [4] proposed several spectra metrics to study

TABLE 3. MEAN EFFECTIVENESS ON INDIVIDUAL PROGRAMS

Subjects	Tarantula	Tarantula-Naish	Tarantula-REV	Jaccard	Jaccard-Naish	Jaccard-REV	Ochiai	Ochiai-Naish	Ochiai-REV	SBI	SBI-Naish	SBI-REV
schedule	7%	7%	**3%**	7%	6%	**3%**	4%	4%	**3%**	**7%**	**7%**	**7%**
print_tokens	9%	8%	**3%**	7%	5%	**3%**	**3%**	**3%**	**3%**	9%	**8%**	**8%**
tot_info	37%	37%	**30%**	35%	35%	**29%**	33%	32%	**28%**	**37%**	**37%**	**37%**
schedule2	59%	57%	**48%**	59%	57%	**54%**	54%	54%	**47%**	59%	**57%**	**57%**
tcas	52%	52%	**50%**	52%	52%	**51%**	50%	50%	**49%**	**52%**	**52%**	**52%**
print_tokens2	30%	30%	**23%**	29%	29%	**20%**	25%	24%	**18%**	30%	30%	**29%**
replace	12%	11%	**10%**	11%	10%	**9%**	**9%**	**9%**	**9%**	12%	11%	**10%**

the accuracy of prediction by fault localization. In their work, they found that Ochiai metric is more effective in bug localization performance than other metrics. Similar to Tarantula, these metrics only use binary information of test execution to rank statements.

Wong et al. [5] proposed some heuristic strategy to assign different weights for passed and failed tests respectively. Weights for passed/failed tests are assigned according to the number of passed/failed tests. As the number increases, weight gets lesser. Our work differs from Wong et al. in that 1)We focus on weight of test cases from test property such test coverage and status of tests associated with a statement. Thus, weights can be dynamically assigned to individual tests. 2) Our approach is a refinement method that can be applied into some existing SFL techniques.

Naish et al. [10] also proposed a weighting strategy for failed tests. The rationale behind the idea is that failed tests that cover few statements provide more information than other failed tests. Thus, they considered weight of a failed test is inversely proportional to the number of statements exercised in the test. Our work extends their studies in two aspects: 1) We use basic blocks rather than statements to calculate weight for each failed test since statements within a basic blocks are often not distinguishable from each others in terms of error diagnosis. 2) We additionally consider the imbalance property of tests with respect to each statement. The weight of a failed test is assigned according to proportion between the number of failed tests and that of passed tests. In this study, we also compare our weighting strategy with method proposed by Lee Naish et al.

Bandyopadhyay et al. [11] extended the idea of nearest neighbor model [6] to incorporating the relative importance of different passing test cases in the calculation of suspiciousness scores. They stated that the importance of a passing test case is proportional to its average proximity to the failing test cases. They used different thresholds for their weighting function to control weights assigned to tests. However, in their study, they do not prescribe how to choose best threshold for weighting function and thus we do not make comparison with their work. Furthermore, our work focuses on relative importance of failing tests while their work focus on relative importance of passing tests.

V. CONCLUSIONS AND FUTURE WORK

In this study, we proposed a weight-based refinement for SFL techniques to enhance errors diagnosis. In our proposal, we take into consideration of variable weights for different tests, which depends on execution information and status of tests. We conducted an experiment to evaluate the refinement techniques using four representative SFL techniques. The experimental results suggested that the revised techniques can always achieved better performance than the techniques' original performance.

In our future study, we plan to conduct more empirical studies by using larger scale programs and multiple-faults versions. We also wish to explore other factors that may impact on weights of test cases to develop more effective strategies for fault localization techniques.

REFERENCES

[1] I. Vessey, "Expertise in Debugging Computer Programs: A Process Analysis", ;presented at International Journal of Man-Machine Studies, 1985, pp.459-494.

[2] J. A. Jones, M. J. Harrold, and J. Stasko, "Visualization of test information to assist fault localization," in ICSE '02: Proceedings of the 24th International Conference on Software Engineering, Orlando, Florida, 2002, pp.467–477.

[3] J.A. Jones and M.J. Harrold, "Empirical evaluation of the tarantula automatic fault-localization technique", in Proc. ASE, 2005, pp.273-282.

[4] R. Abreu, P. Zoeteweij, R. Golsteijn, and A.J.C.V. Gemund, "A practical evaluation of spectrum-based fault localization", presented at Journal of Systems and Software, 2009, pp.1780-1792.

[5] W.E. Wong, Y. Qi, L. Zhao, and K. Cai, "Effective Fault Localization using Code Coverage", in Proc. COMPSAC (1), 2007, pp.449-456.

[6] M. Renieris and S. Reiss, "Fault localization with nearest neighbour queries", in Proceedings of the 20th IEEE/ACM International Conference on Automated software engineering, Montreal, Canada, 2003, pp. 30–39.

[7] H. Cleve and A. Zeller, "Locating causes of program failures", In Proceedings of the International Conference on Software Engineering, pages 342−351, St. Louis, Missouri, May 2005.

[8] Y. Yu, J. A. Jones, and M. J. Harrold, "An empirical study of the effects of test-suite reduction on fault localization," in ICSE '08: Proceedings of the 30th International Conference on Software Engineering, Leipzig, Germany, 2008, pp. 201–210.

[9] [Online]. Available: http://sir.unl.edu/portal/index.html

[10] L. Naish, H.J. Lee, and K. Ramamohanarao, "Spectral Debugging with Weights and Incremental Ranking", in Proc. APSEC, 2009, pp.168-175.

[11] A. Bandyopadhyay and S. Ghosh, "Proximity based weighting of test cases to improve spectrum based fault localization", in Proc. ASE, 2011, pp.420-423.

[12] H. Cleve and A. Zeller. Locating causes of program failures. In Proceedings of the International Conference on Software Engineering, pages 342-351, St. Louis, Missouri, May 2005.

Comparative Evaluation of Programming Paradigms: Separation of Concerns with Object-, Aspect-, and Context-Oriented Programming

Fumiya Kato, Kazunori Sakamoto, Hironori Washizaki, and Yoshiaki Fukazawa
Dept. Computer Science and Engineering
Waseda University
Tokyo, Japan
E-mail: fum_kato@asagi.waseda.jp

Abstract

There are many programming paradigms for the separation of concerns (SoC). Each paradigm modularizes concerns in a different way. Context-oriented programming (COP) has been developed as a supplement to object-oriented programming (OOP), which is one of the most widely used paradigms for SoC. It modularizes concerns that are difficult for OOP. In this paper, we focus on three paradigms - OOP, aspect-oriented programming (proposed as a supplement to OOP that has a different approach from COP), and COP - and study whether COP can modularize concerns better than other two paradigms in given situations. Then we determine the reasons why COP can or cannot better modularize concerns.

1. INTRODUCTION

In software development, the separation of concerns (SoC) is an important matter. To deal with SoC, numerous programming paradigms have been proposed. Object-oriented programming (OOP) is one of the most widely used paradigms. However, some concerns called cross-cutting concerns (CCCs) are difficult to modularize for OOP and are often scattered over modules in a program. As a supplement to OOP, aspect-oriented programming (AOP) has been proposed. AOP modularizes CCCs as *aspects* that weave codes into other modules.

In recent years, context-oriented programming (COP) has been proposed as a supplement to OOP that has a different approach from AOP. COP modularizes behavior that depends on the state of execution as *layers*. Several COP languages have been developed [1, 2, 3] however, there has been little research on the situations COP is most effective for developing and how COP modularizes concerns better than the other paradigms.

In this paper, we focus on three programming paradigms: OOP, AOP, and COP and perform comparative experimentation on these paradigms to research their effectiveness of achieving SoC. To measure the effectiveness of achieving SoC, we perform experimentation in terms of the description amount and the locality of change. The purpose of this study is to answer the following research questions (RQs):

RQ1: Do the description amount and the locality of change differ in implementing programs with the same requirement by OOP, AOP, and COP?
RQ2: Are there any situations in which COP is superior or inferior to OOP and AOP in terms of the description amount and the locality of change?
RQ3: What features of COP result in its superiority to OOP and AOP?
RQ4: What features of COP result in its inferiority to OOP and AOP?

The contributions of this paper are as follows:

- **Suggestion of the comparative evaluation scheme for programming paradigms.**
- **The comparative results of the superiority or inferiority of OOP, AOP, and COP in terms of the description amount and the locality of change.**
- **Specification of the causes of superiority and inferiority in the comparative results.**

The rest of the paper is structured as follows. Section 2 introduces the cross-cutting concerns that are difficult to modularize by OOP, and how AOP and OOP modularize such concerns. Section 3 presents the study format: the purpose of study, the method of comparative study, and the programming languages used. Section 4 shows the results of the study and our analysis. We discuss related works in Section 5 and summarize the paper in Section 6.

2. BACKGROUND

2.1. Cross-Cutting Concerns

CCC is a type of concern that is entangled with other concerns. As the source code level, CCCs are often scattered over modules in a program. For instance, logging to

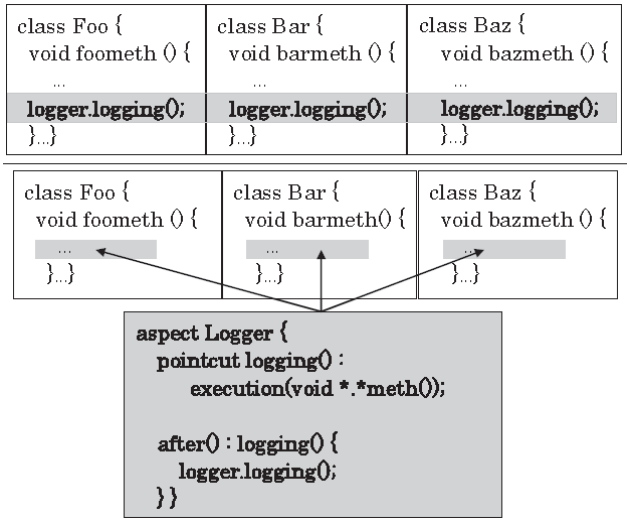


Figure 1. Cross-cutting concern of logging code in OOP (top) and modularization of CCC of logging code in AOP (bottom)

carry out debugging is one CCC. As shown in the top of Figure 1, it is difficult to modularize CCCs by OOP. A logging code can be written in many modules in a program in OOP. Such a situation worsens maintainability, because scattered logging codes make programs unreadable, and causing mistakes in modifying or deleting logging codes.

2.2. Aspect-Oriented Programming

AOP has been proposed as a supplement to OOP. AOP modularizes CCCs as *aspects*. The bottom of Figure 1 shows the modularization of CCCs about logging by AOP. Aspects separate CCCs from the major concerns that each class modularizes by weaving CCC codes into other modules. AOP has a *pointcut-advice* mechanism for achieving weaving. *Advice* defines the operation woven into other modules. *Pointcut* defines the modules into which advice is woven and the points - for instance, particular methods are executed or particular types of objects are accessed - at which woven codes are validated.

2.3. Context-Oriented Programming

COP has been proposed as a supplement to OOP via a different approach from AOP. COP can modularize CCCs about changing behavior depending on the state of execution as *layers*. A layer defines the methods in other classes. Methods defined in a layer are executed in a particular state instead of the original method definitions. Figure 2 shows the modularization of CCCs about logging by COP. A layer defines the methods in other classes with logging codes. Method definitions with logging codes in a layer can be executed instead of the base definitions in other classes when

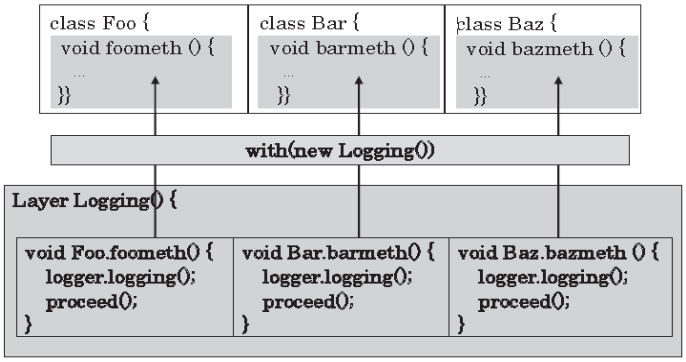


Figure 2. Modularization of CCC with layer

the layer is activated. Layers are activated by *with* statements. In addition, base definitions can be called from the definitions in a layer by *proceed* statements.

3. COMPARATIVE EVALUATION SCHEME

Figure 3 shows the overview of the comparative evaluation. To detect situations in which COP is superior or inferior to OOP and AOP, we performed comparative experimentation. In this paper, we created three sample programs containing CCCs about changing behavior, implemented them in three programming languages, and performed seven modification experimentation. In addition, we implemented some parts of open-source software (OSS) in Java with each programming language. From the results of our implementation, we discuss the superiority and inferiority of COP.

In the analysis of the results, we use two criteria: description amount and locality of change. For comparison of the locality of change, we created seven change tasks as follows: add classes related or unrelated to CCCs, add methods related or unrelated to CCCs, rename methods related or unrelated CCCs, and delete CCCs. We counted the chunks of code that the change tasks forced to modify.

As programming languages for OOP, AOP, and COP, we use Java, AspectJ [4, 5] , and JCop [1] respectively. AspectJ and JCop are implemented by extending Java. Thus, the forms of the fundamental descriptions of Java, AspectJ, and JCop are similar. Therefore, the results should be less affected by differences in the abstraction level and grammar of languages, and more affected by differences among paradigm features.

4. EVALUATION EXPERIMENTATION

4.1. Target

4.1.1. Sample Programs

To compare OOP, AOP, and COP, we created three sample programs that are based on the samples on the JCop project page [6]: address book, bank account system, and

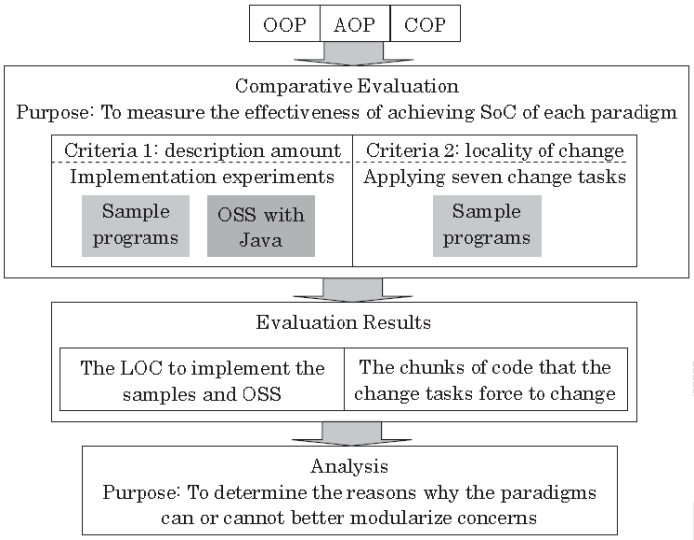


Figure 3. Overview of the comparative evaluation.

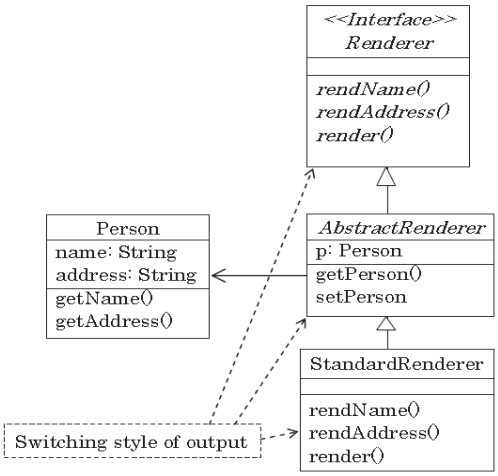


Figure 4. Design of address book and additional design concern

```
Fumiya KATO
Fumiya KATO, Chiba Japan
<div class = "person"><b>*Fumiya KATO*</b></div>
<div class = "person"><b>Fumiya KATO</b></div>, <i>Chiba Japan</i>
```

Figure 5. Output of address book: each option is switched independently

BMI calculator (https://github.com/FumKato/CompParadigm). They contain CCCs about changing behavior of several classes depending on the state of execution.

Figure 4 shows the design of the address book. As the base behavior, the StandardRenderer class implementing a Renderer interface outputs the fields of a Person object. The CCCs about changing behavior enable renderer classes to switch the styles of output: to render a name with an address, to add HTML tags, and to decorate outputs by ‘ * ’ and ‘ _ ’. As shown in Figure 5, each output option is switched independently. As the description amount, the lines of code (LOC) to implement the program are counted. As the change of locality, the seven change tasks defined in Section 3 are applied to the CCCs about output options. For instance, *Add-Related-Class* adds a new renderer class so that output options can be set, and *Add-Unrelated-Class* adds a new class unrelated to output options.

In the same way as the address book, bank account system has CCCs: logging and encryption of data. The BMI calculator, which is a GUI application with a Qt Jambi framework [8], also has CCCs: switching units of input and style of output.

4.1.2 Open-Source Software

We perform a comparative study of sample programs prior to the comparative study of OSS. The results and analysis of the samples, presented in a later section, indicate that a correspondence relation between layers in JCop and the decorator pattern - one of the GoF design patterns - in Java exists. If JCop improves implementation in the decorator pattern, we can propose the part of the programs the decorator pattern is applied as the COP usage guideline. The guideline can be used to recommend to developers the more positive use of COP than of OOP. Then we performed a comparative study of OSS in the Java that decorator pattern are applied. The OSS we used is JHotDraw 5.3 [7], a Java GUI framework for graphics editor. We found four places decorator patterns are applied, and rewrote them in AspectJ and JCop.

4.2. Evaluation Results

The top of Figure 6 shows the LOC to implement sample programs in each programming language. In the three sample programs, implementation in Java needs the most LOC, and implementation in JCop needs the least LOC.

Table 1 shows the change tasks and chunks of code that the tasks forced to change. In *Add-Related-Method*, *Add-Unrelated-Method*, and *Rename-Unrelated-Method*, implementation in AspectJ and JCop need less change than in the case of OOP. In particular, in *Add-Related-Method* of the address book and bank account system, implementation in JCop needs less change than in the other languages. On the other hand, in *Add-Related-Method*, JCop has the largest number of changed chunks of code.

The modularization of CCCs about changing behavior in sample programs is achieved using the decorator pattern in Java, aspects in AspectJ, and layers in JCop.

The bottom of Figure 6 shows the LOC of implementation where the decorator pattern is applied in each programming language. In three of the four cases, implementation in JCop needs the most LOC, and in the other case, it is impossible for implementation in JCop to rewrite the codes

Table 1. Seven change tasks and affected chunks of code. ‘n’ indicates the number of times the tasks, e.g. in *Add-Related (Unrelated)-Class*, ‘n’ indicates the number of added classes (n > 0).

	Address Book			Bank Account System			BMI Calculator		
	OOP	AOP	COP	OOP	AOP	COP	OOP	AOP	COP
Add-Related-Class	n	0	3n	2n	0	6n	13n	4n	2n
Add-Unrelated-Class	0	0	0	0	0	0	0	0	0
Add-Related-Method	3n	4n	2n	2n	2n	2n	3n	3n	2n
Add-Unrelated-Method	n	0	0	2n	0	0	2n	0	0
Rename-Related-Method	3n	2n	2n	2n	2n	2n	2n	2n	2n
Rename-Unrelated-Method	3n	0	0	2n	0	0	2n	0	0
Delete-Concerns	n	0	1	n	0	1	6n	0	0

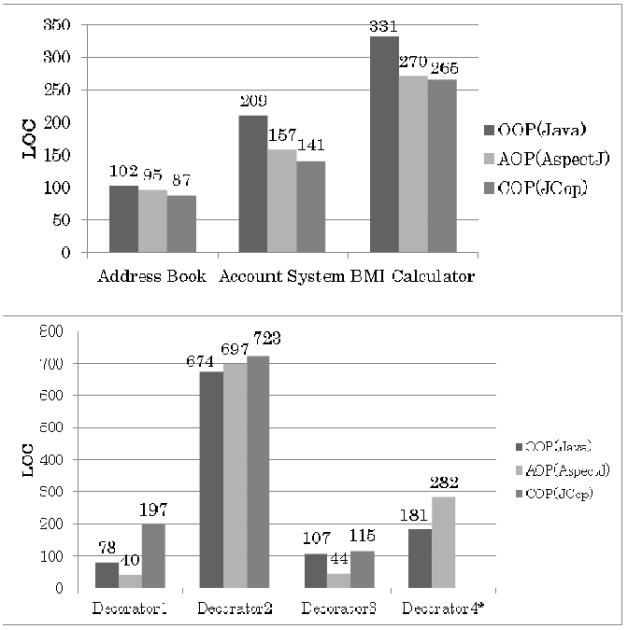


Figure 6. LOC for sample program (top) and OSS with the decorator pattern (bottom) implementation in three programming languages

applied the decorator pattern without changing the fundamental structure of the program. Implementation in AspectJ can reduce LOC in two of the four cases, and in the other cases LOC increases compared with that in OOP.

4.3. Discussion

RQ1: Do the description amount and the locality of change differ in implementing programs with the same requirement by OOP, AOP, and COP?

The comparative result shown in Figure 6 indicates that the description amount differ in implementing programs with the same requirement by OOP, AOP, and COP. The results shown in Table 1 indicate that the localities of change also differ. Therefore, using the programming paradigms properly in appropriate situations would improve the software quality from the viewpoint of the description amount and the locality of change compared with that in the case of using a single paradigm.

RQ2: Are there any situations in which COP has superiority or inferiority to OOP and AOP in terms of the description amount and the locality of change?

The comparative results of the sample programs shown in the top of Figure 6 indicate that situations in which COP is superior to OOP and AOP exist from the viewpoint of the description amount. On the other hand, the results for OSS shown in the bottom of Figure 6 indicate that situations in which COP is inferior to OOP and AOP exist.

The results shown in Table 1 indicate that situations in which COP is both superior and inferior to OOP and AOP exist from the viewpoint of the locality of change. The situations in which COP is superior to OOP are the change tasks of *Add-Unrelated-Method, Rename-Unrelated-Method*, and *Delete-Concerns*. These change tasks affect COP in the same way as AOP. In *Add-Related-Class*, COP is less affected than OOP and AOP.

RQ3: What features of COP result in its superiority to OOP and AOP?

The situations in which COP is superior to other two paradigms can be classified under two types of descriptions: necessary for Java but unnecessary for AspectJ and JCop, and necessary for AspectJ but unnecessary for JCop.

First, we discuss the former. As mentioned above, the decorator pattern is applied in implementation in Java. Figure 7 shows that the concerns about changing behavior are implemented as the Html class; such a class is called *decorator class*. The Html class changes the behavior of StandardRenderer class - such classes are called *component classes* - through the field of a component class object. Therefore, decorator classes need descriptions that set and get component class objects. On the other hand, implementation in AspectJ or JCop does not need such descriptions, because aspects and layers have mechanisms of weaving codes that change behavior into other classes.

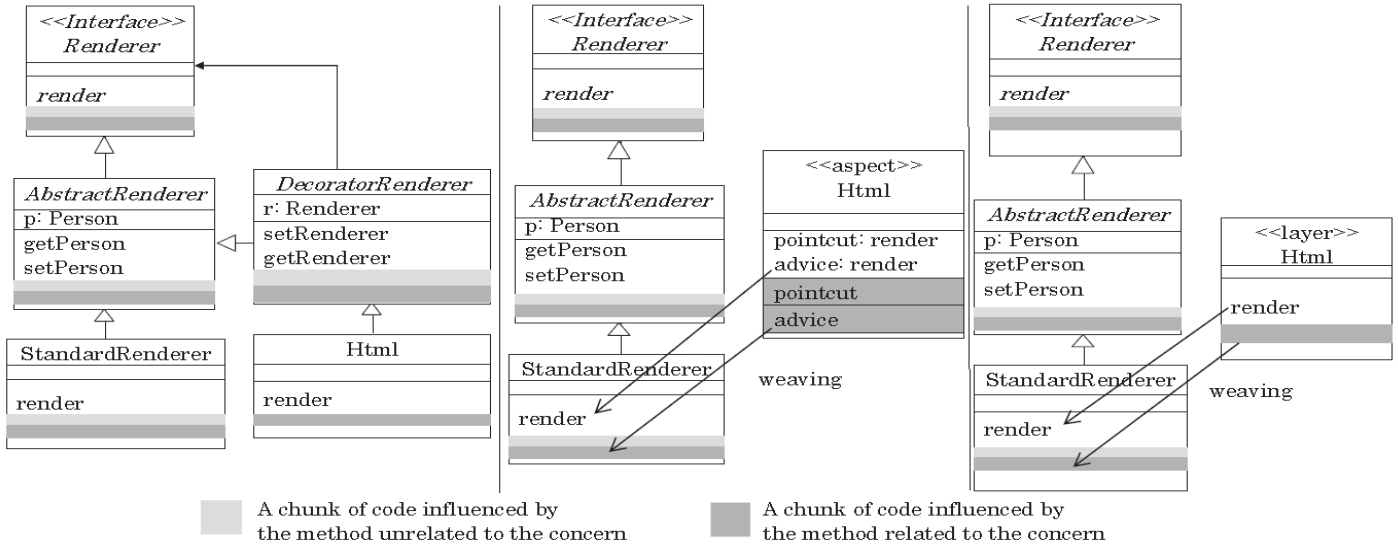


Figure 7. Part of address book program implemented in Java (left), AspectJ (center), and JCop (right).

Furthermore, a decorator class needs to implement the same interface with a component class in implementation in Java. Therefore, a decorator class needs to define not only the methods that change behavior but also other methods unrelated to the concerns about changing behavior. On the other hand, aspects and layers do not need to define the methods unrelated to the concerns about changing behavior, because if aspects or layers do not describe the methods of other classes, the methods defined in each class are just called. For that reason, adding methods unrelated to the concerns about changing behavior does not force aspects or layers to change any codes and only implementation in Java need to change codes as shown in Figure 7.

Secondly, we present the later. Figure 7 shows the example. Adding a method that changes behavior forces decorator classes, aspects, and layers to change codes. To adopt the added method, aspects need to define advice - that is the operation weaved into other classes - and the pointcut that defines the classes and timing advice is weaved for each advice. Therefore, two chunks of code - pointcut and advice - are needed for each new method that changes behavior. On the other hand, the method definitions in layers are bound to only classes weaved codes. The timing at which weaved codes are validated is bound to layers. Therefore, if the timing of validating weaved codes is common to the methods defined in a layer, JCop defines it only once no matter how many methods are defined in a layer. For these reasons, implementation in JCop needs less change in the *Add-Related-Method* than that in AspectJ.

RQ4: What features of COP result in its inferiority to OOP and AOP?

The situations in which COP shows inferiority originate from the change task *Add-Related-Class* in Table 1 and the results of the decorator pattern in OSS shown in Figure 6. Figure 8 shows an example: part of the address book program. In the change task *Add-Related-Class*, the TableRenderer class is added. The TableRenderer class has methods that changes behavior the same as the StandardRenderer class. This change task does not force an aspect to change any codes, because the pointcut rendering in the Html aspect already defines the classes that are weaved codes as subclasses of the Renderer interface. Such a definition is achieved by ‘ Renderer+ ’ .

On the other hand, *Add-Related-Class* forces layers to describe redundant method definitions as shown in Figure 8. An Html layer needs to describe almost the same method definitions that differ only in class path to weave into the StandardRenderer and TableRenderer classes, because each method definition is bound on only one class. Therefore, situations in which COP is inferior to OOP and AOP exist.

The main reasons why implementation in JCop needs more LOC than that in other languages in OSS are the same as those for sample programs. A decorator class is often applied to two or more component classes; therefore, implementation in JCop needs redundant method definitions such as in the example given in Figure 8.

In addition, layers in JCop cannot define members that are not weaved into other classes. Therefore, in Java, if decorator classes define private members, layers cannot define such members. Thus, each class that is weaved code by layers needs to define members that are accessed by only layers. For this reason, implementation in JCop makes codes accessed by only layers scattered.

4.4. Threats to Validity

The results and analysis are based on our implementation in each programming paradigm. Therefore, threats to internal validity exist, because the affect of the difference in the developer on the comparative results is not discussed in this paper. As future work, we will perform implementation experimentation with several programmers to validate

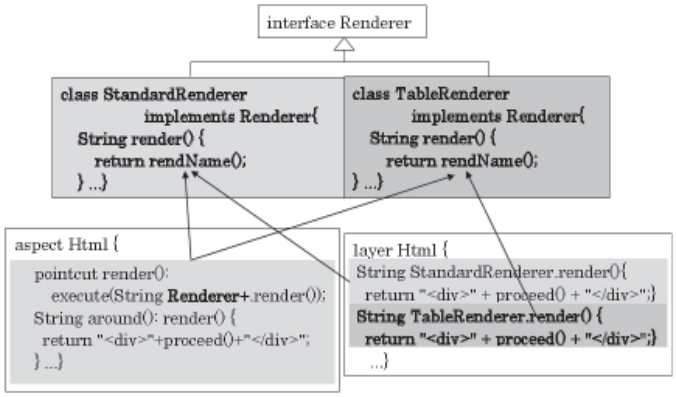


Figure 8. Situation in which COP has inferiority to OOP and AOP. Adding a new related class forces layers to describe redundant method definitions that differ only in class path.

the generality of the results and analysis discussed in this paper.

5. RELATED WORK

In recent years, many COP languages have been developed [9] including those for Java [1, 2, 3, 10]. These studies focused on specification of the languages and their performance of execution. However, these studies do not evaluate how degree and what situations these languages can achieve SoC compared with other programming paradigms.

The prior works that have guided this study is given in [11, 12, 14, 15]. Figueiredo et. al [11] performed a quantitative study of AOP that investigated the efficacy of AOP to prolong design stability of software product lines. This study focused upon a multi-perspective analysis in terms of modularity, change propagation, and feature dependency measured by metrics for concerns [13]. Kiczales and Mezini [14] performed comparative study of programming paradigms, which dealt with procedure calls, pointcut-advice, and annotation by the implementation of a sample program. The comparison was in terms of the locality and the implicitness. Hannemann and Kiczales [15] performed study of improving design patterns by AOP, which showed a comparison of Java and AspectJ by implementation of sample programs including design patterns. The comparison was in terms of the modularity and the reusability. Our study is inspired by these studies and focuses on COP. We perform comparative studies in terms of the description amount and the locality of change measured by basic metrics such as LOC and chunks of code as a first step towards the evaluation of the efficacy of COP to deal with CCCs.

6. CONCLUSIONS AND FUTURE WORK

As the programming paradigm that achieves SoC with a different approach from existing paradigms, COP has been proposed. From the viewpoints of the description amount and the locality of change, we performed comparative studies of OOP, AOP, and COP to detect situations in which COP achieves better SoC and determine the reason why COP can or cannot achieve better SoC. From these results, several avenues for future work exist.

The next step would be a comparative study with larger projects applying metrics used in prior works [11, 13]. A comparative study focusing on multi-perspective analysis would also be interesting, for instance, reusability, implicitness, ease of description, and learning cost. A second line of work would be to discuss and develop a COP language that improves the inferior situations detected in this research.

References

[1] Appeltauer, M. et al. : Event-Specific Software Composition in Context-Oriented Programming, SC 2010, pp. 50-75.

[2] Appeltauer, M. et al. : Context-oriented Programming with Java, 26th JSSST Annual Conference, 2009.

[3] Salvaneschi, G. et al. : JavaCtx: Seamless Toolchain Integration for Context-Oriented Programming, COP'11, 2011.

[4] AspectJ, http://www.eclipse.org/aspectj/

[5] Kiczales, G. et al. : An Overview of AspectJ, ECOOP '01, pp. 327-353, 2001.

[6] JCop-Context-Oriented Programming Projects, https://www.hpi.uni-potsdam.de/hirschfeld/trac/Cop/wiki/JCop

[7] JHotDraw, http://www.jhotdraw.org/

[8] Qt Jambi, http://qt-jabmi.org/

[9] Schippers, H. et al. : An Implementation Substrate for Languages Composing Modularized Crosscutting Concerns, SAC'09, pp. 1944-1951, 2009.

[10] Hirschfeld, R. et al. : Context-oriented Programming, Journal of Object Technology, Vol. 7, No. 3, pp.125-151, 2008.

[11] Figueiredo, E. et al. : Evolving Software Product Lines with Aspects: An Empirical Study on Design Stability, ICSE' 08, pp. 261-270, 2008.

[12] Soares, S. et al. : Implementing Distribution and Persistece Aspects with AspectJ, OOPSLA'02, pp. 174-190, 2002.

[13] Figueiredo, E. et al. : On the Maintainability of Aspect-Oriented Software: A Concern-Oriented Measurement Framework, CSMR 2008, pp. 183-192, 2008

[14] Kiczales, G. and Mezini, M.: Separation of Concerns with Procedures, Annotations, Advice and Pointcuts, ECOOP 2005, pp. 195-213, 2004.

[15] Hannemann, J. and Kiczales, G.: Design Pattern Implementation in Java and AspectJ, OOPSLA'02, 2002.

Extended Design Patterns in New Object-Oriented Programming Languages

Kazunori Sakamoto #1, Hironori Washizaki *2, Yoshiaki Fukazawa *3

National Institute of Informatics
1 exkazuu@nii.ac.jp

* Dept. Computer Science and Engineering, Waseda University
2 washizaki@waseda.jp 3 fukazawa@waseda.jp

Abstract

Most of design patterns are implemented in major object-oriented programming languages such as C++ and Java. However, newer object-oriented programming languages than such languages has new language features which can improve implementations of design patterns.

In this paper, we propose two extended design patterns called customizable state pattern and deeply immutable pattern. We compares implementations of our design patterns in Java, C++ and eight new object-oriented programming languages through our motivating example. As a result, we confirmed new languages, in particular Scala, improved implementations of our design patterns.

1. Introduction

The Gang-of-Four (GoF) design patterns (DPs) make software quality better by providing reusable design solutions [2]. The authors' sample implementations of the GoF DPs are written in C++. Other samples also exist in different programming languages such as Java and C#.

Most of new object-oriented programming (OOP) languages have functional features. For example, Java7 [1] had planned to employ closure features. Although Java8 will employ closure features instead Java7, several new languages on Java Virtual Machine (JVM) such as Scala have already employed functional features including closures.

While DPs do not depend on programming languages, implementations of DPs depend on programming languages. The reason is that each programming language has different language features so that ways to implement instances of DPs are different with respect to each programming language. For example, C++ supports multiple inheritance, while Java lacks it. Thus, it is difficult to design a class to have multiple roles in Java.

The GoF DPs are currently designed mainly for C++ and Java. However, newer OOP languages than C++ and Java can improve implementations of the GoF DPs. For example, the strategy pattern is designed as an alternative way to implement functional values. OOP languages with functional features such as Scala can directly handle functional values without the strategy pattern. As another example, Scala supports an object class which represents a singleton object with the singleton pattern.

Many researchers improved implementations of DPs with other programming paradigms than the OOP [3–5, 9]. Moreover, a number of DPs are also proposed for other paradigms than the OOP [1, 6].

In this paper, we propose two extended DPs through our development experience of ACM JavaChallenge 2012, which is an artificial intelligence programming (AI) contest related to game software [2]. We discuss how we can improve implementations of our DPs in new OOP languages including Scala in comparison with C++ and Java. As a result, we confirmed new OOP languages can improve implementations of our DPs. We believe our result indicates new OOP languages promote improving and extending existing DPs.

The contributions of this paper are as follows.

- Two extended DPs called customizable state and deeply immutable DPs.
- Improvements on implementations of our DPs in new OOP languages such as Scala.
- What language features are required to improve our design patterns.

2. Motivating example

We explain a need of extended DPs through our development experience of JavaChallenge 2012.

2.1. Requirements of sample game software

We developed game software for JavaChallenge 2012 in Java and Scala. The contestants develop AI programs on our game software to compete with each other. As a result

[1] http://www.jcp.org/en/jsr/detail?id=335

[2] We will publish a paper on ICSE GAS 2013 which highlights whole development experience of JavaChallenge 2012 including our DPs. However, this paper highlights only extended DPs in new OOP languages.

of our requirement analysis, we found the game software have to satisfy five requirements as follows. Note that we consider the game software as sample software for our discussion in this paper.

Functional requirement (FR1) The software have to make progress based on a state machine. The software shows and executes various scenes such as a title, main and end scenes corresponding to a current state.

Functional requirement (FR2) The software have to allow users to switch various playing modes such as user manipulation, AI manipulation, graphical user interface (GUI) and console user interface (CUI) modes.

Non-functional requirement, security (NFR1) Game states of the software have to be protected from user-developed AI programs. AI programs is prohibited to modify the game states illegally.

Non-functional requirement, concurrency (NFR2) The software have to be concurrently work to speed up AI programs so that it aids them to find the best action.

Non-functional requirement, maintainability (NFR3) The source code of the software have to have no duplicated code and no redundant code because such code reduces maintainability such as changeability and understandability.

We had previously developed a framework called Game AI Arena (GAIA) which aids to develop game software where user-developed AI programs can be added [7]. GAIA provides a `Scene` interface and a `SceneManager` class as a feature of a state machine designed by the state pattern. The `SceneManager` class changes game scenes by switching an active `Scene` object. The `Scene` interface has an `advance` method which returns a next `Scene` object for the switch.

2.2. Design with the state pattern in Java

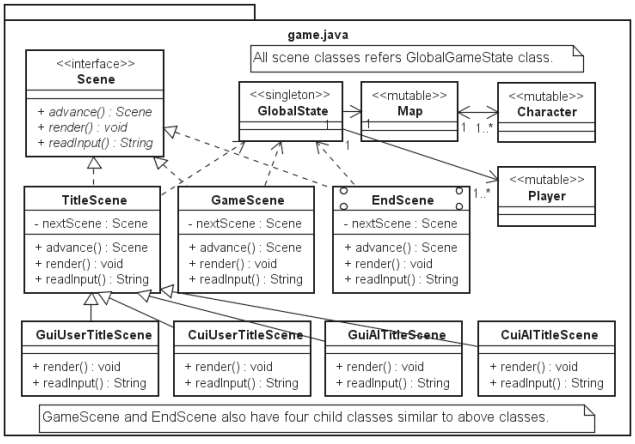


Figure 1. Class diagram of the software with the state pattern in Java

Figure 1 shows a class diagram of the software in Java. This diagram does not contain the `SceneManager` class and several unrelated classes for the simplicity. The `GlobalState`, `Map`, `Character` and `Player` classes are for representing a game state. Theses classes are mutable, and thus, their objects can be modified in playing games.

The game consists of three scenes to satisfy FR1: a title, a main, and an end scenes. The `TitleScene`, `MainScene`, `EndScene` abstract classes represent these scenes. The title scene initializes the game showing a title image, the main scene deals with the game logic showing a game screen and the end scene calculates the result of the game showing the result.

The software has four modes to satisfy FR2: a user manipulation, an AI manipulation, a GUI and a CUI modes. Although the user manipulation mode acquires game inputs from a keyboard, the AI manipulation mode acquires game inputs from AI programs. The user manipulation mode is suitable for debugging the software and AI programs. The GUI mode shows a graphical game screen, while the CUI mode shows a text-based game screen. The CUI mode is also suitable for debugging AI programs by speeding up the game. For example, the `GuiUserTitleScene` class is for the title scene with the GUI and the user manipulation modes.

This design has three problems as follows.

- The classes representing the game state are mutable. The mutable classes cause risks of illegally modifying the game states so that this design violates NFR1.
- The scene classes are strongly combined with the singleton class as a global variable of the game state. This dependence makes the concurrent execution difficult so that this design violates NFR2.
- Duplicated and redundant code exists between scene classes such as the `GuiUserTitleScene` class and the `GuiUserMainScene` class (abbreviated in the diagram) so that this design violates NFR3.

2.3. Design with the state pattern in C++

Although we can apply multiple inheritance which is applied only when defining classes into the scene classes to reduce duplicated code, applying such multiple inheritance increases classes. Figure 2 shows a class diagram of the software with the state pattern in C++, which supports the multiple inheritance. For example, duplicated code between the `GuiUserTitleScene` and `CuiUserTitleScene` classes are extracted into the `UserInputScene` class. The `GuiScene`, `CuiScene` and `AIInputScene` classes are also newly added to extract the duplicated code.

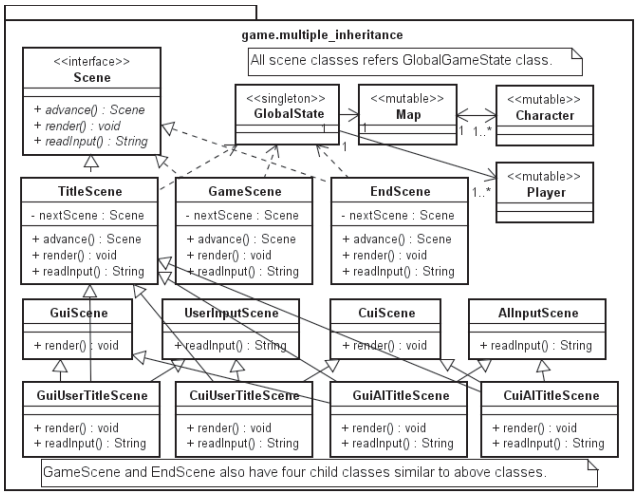


Figure 2. Class diagram of the sample game software with the state pattern in C++

2.4. Design with the state and decorator patterns

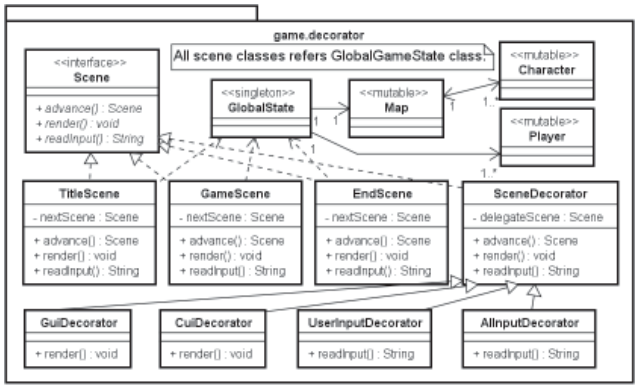


Figure 3. Class diagram of the software with the state and decorator patterns

```
1 Scene scene = new GuiDecorator(
2     new UserInputDecorator(new TitleScene()));
```

Figure 4. Java code constructing a similar object to a `GuiUserTitleScene` object

We can also employ the decorator pattern instead of multiple inheritance. The decorator pattern allows to enhance objects with delegation. Figure 3 shows a class diagram of the software with the state and decorator patterns in Java. The `SceneDecorator` class is a base class of a decorator which has the `delegateScene` field to be enhanced. For example, Figure 4 shows Java code which constructs a similar object to a `GuiUserTitleScene` object. Although this design adds the five decorator classes such as the `SceneDecorator`, `GuiDecorator`, `CuiDecorator`, `UserInputDecorator` and `AIInputDecorator` classes, it removes 16 redundant classes such as the `GuiUserTitleScene` class in Figure 1 to satisfy NFR3. Moreover, this design extracts the duplicated code into the five decorator classes to satisfy NFR3. Thus, this design is better than the designs of Figures 1 and 2.

2.5. Design with the state pattern in Scala

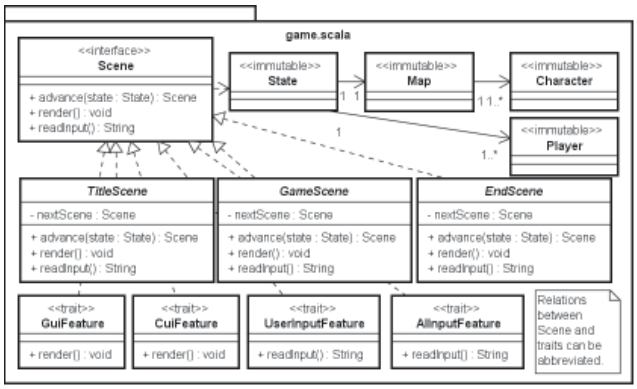


Figure 5. Class diagram of the software with the state pattern in Scala

```
1 val scene = new TitleScene()
2             with UserInputFeature with GuiFeature
```

Figure 6. Scala code constructing a similar object to a `GuiUserTitleScene` object with mixin

Scala supports mixin instead of multiple inheritance so that Scala can merge classes with traits [8]. Scala also allows to construct an object with mixin which is applied when creating objects. Figure 5 shows a class diagram of the software with the state pattern in Scala. Figure 6 shows Scala code which constructs an similar object to a `GuiUserTitleScene` object with the mixin. Although this design adds the four traits such as the `GuiFeature`, `CuiFeature`, `UserInputFeature`, and `AIInputFeature` traits, it removes 16 classes such as the `GuiUserTitleScene` class in Figure 1 to satisfy NFR3. Moreover, this design extracts the duplicated code into the four traits to satisfy NFR3. This design is better than the designs of Figures 3 because this design does not require a similar class to the `SceneDecorator` class.

2.6. Design about immutability

Immutability is preferable because changes of variable values increase complexity of programs. Functional programming represents programs with mapping of function

application while imperative programming represents programs with changes of variable values. Thus, functional programming can easily employ immutability in programs. The `GlobalState` class and the related classes such as the `Map`, `Character` and `Player` classes in Figures 1, 2 and 3 are mutable. Objects of these classes can be easily modified from other programs.

We can employ immutable objects to prevent illegal modification completely. The `State` class in Figures 5 is immutable. Moreover, the classes referred from the `State` class are also immutable. When the referred classes are mutable, the game state can be modified partially. Thus, the `State` class have to be deeply immutable to satisfy NFR1.

The `GlobalState` class is a singleton class which represents the game state. All scene classes refer the `GlobalState` class to make progress on games. This design strongly combines the scene classes with the `GlobalState` class. For example, AI programs sometimes require enormous time to search the best action. Although executing multiple games concurrently can reduce the time, this combination makes the concurrent execution difficult. Moreover, the singleton object cannot represent multiple game states. In contrast, the `State` class is used only as a parameter of methods in the scene classes. These scene classes can treat multiple game states for the concurrent execution to satisfy NFR2 because the scene classes do not have `State` objects in fields.

3. Customizable state pattern

We extract a new DP called customizable state pattern which extends the state patterns from Figures 3 and 5. In this section, we show the description of the customizable state pattern.

3.1. Context

Program behavior changes corresponding to a state based on a state machine. A program on each state deals with various operations such as doing logic and rendering an UI. The operations differ depending on options such as a CUI and a GUI modes.

3.2. Problem

Options cause combinatorial explosion so that it drastically increases conditional branches or classes. Options also cause duplicated code and redundant code so that it reduces maintainability.

3.3. Forces

- Combinatorial explosion owing to options increases program elements linearly with the number of options.
- Source code representing options does not contain duplicated code and redundant code.

3.4. Solution

Create modules representing behavior of states with respect to each option. Note that the number of modules should be equal to the number of options except for additional modules such as a base class. Combine state modules with option modules with one of the following ways.

3.5. Implementation

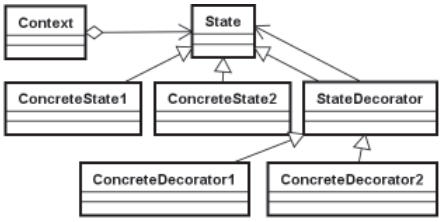


Figure 7. Class diagram of customizable state pattern with the mixin

Figure 7 shows a class diagram of the customizable state pattern with the state pattern and multiple inheritance or mixin which is applied when creating objects. The `StateFeature` modules are merged with `ConcreteState` classes corresponding to options. This implementation requires only modules corresponding to options without additional classes.

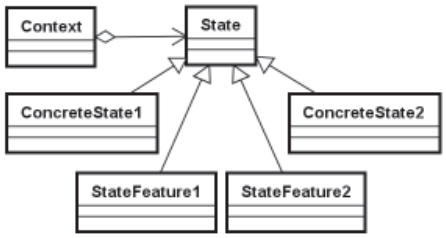


Figure 8. Class diagram of customizable state pattern with decorator pattern

Figure 8 shows a class diagram of the customizable state pattern with the state and decorator patterns. The `ConcreteDecorator` classes are corresponding to options and enhance the `ConcreteState` classes. This implementation requires additionally the `StateDecorator` class in comparison with the design of Figure 7.

3.6. Consequences

The customizable state pattern deals with the combinatorial explosion owing to options. Basically, the implementations of the options requires only the same number of the modules as the options. The modules modularize the implementations of the options well so that duplicated code and redundant code do not appear.

3.7. Related Patterns

State patter Although the state pattern does not consider how to modularize state classes which have various op-

erations, the customizable state pattern aids to modularize state classes where options are added.

Decorator pattern and bridge pattern The decorator and bridge patterns can be utilized to modularize state classes with various operations. Although multiple inheritance or mixin which is applied when creating objects is a better way to modularize them, the decorator pattern is also one of ways to modularize them in OOP languages without the multiple inheritance and the mixin.

4. Deeply immutable pattern

We extract a new DP called deeply immutable pattern which extends the immutable patterns from Figure 5. In this section, we show the description of the deeply immutable pattern.

4.1. Context

A program consists of a set of classes for representing a program state (not related to state role for the state pattern). The program state must not be illegally changed by added user programs. The program allows to clone and restore program states to search other program states. The program can concurrently work using multiple program states to speed calculations up.

4.2. Problem

It is not clear how to modularize the program state as immutable modules in OOP languages because most of major OOP languages are designed for imperative programming. Moreover, the singleton pattern is frequently used as only global variables. However, a singleton class cannot be utilized to clone and restore program states and cannot treat multiple program states.

4.3. Forces

- Program states are protected from illegal modification by user programs.
- Program states are cloned and restored to search other program states.
- The program treats the multiple program states.

4.4. Solution

Modularize the set of classes for representing a program state with a tree structure making all the fields in the classes immutable. Determine one of program state classes as a root class and the other classes as node classes. Change each class so that they have one-way connections to child node classes and all node classes can be scanned from the root class. Copy the root and node classes from a modified node class recursively when generating a new program state.

4.5. Implementation

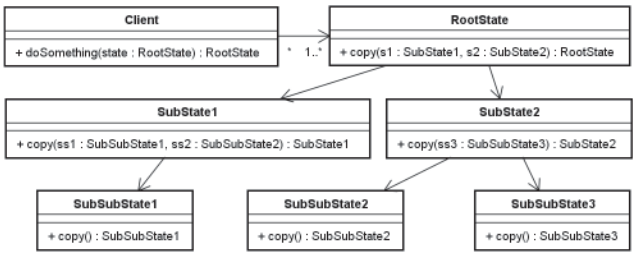


Figure 9. Class diagram of deeply immutable pattern

Figure 9 shows a class diagram of the deeply immutable pattern. The `Client` class manipulates the program state locally (e.g. acquiring a `State` object as a parameter and passing a new modified object to other methods) because a global immutable object cannot be utilized to represent changeable program states.

```
1  case class State(SubState1 sub1, SubState2 sub2) {
2    def newState() = {
3      val newSub1 = createNewSub1()
4      this.copy(sub1 = newSub1)
5    }
6  }
7  case class SubState1(SubSubState1 subSub1,
8                       SubSubState2 subSub2) {}
9  case class SubState2(SubSubState3 subSub3) {}
10 case class SubSubState1() {}
11 case class SubSubState2() {}
12
13 val state: State = initialize()
14 val newState = state.newState()
```

Figure 10. Scala code which defines the classes

Figure 10 shows Scala code which defines the classes in Figure 9. Several OOP languages with functional features such as Scala aids to implement immutable classes. Case class in Scala generates several useful methods including the `copy` method. The `copy` method copies the receiver's object accepting parameters for changing the specific fields. The Scala code in Figure 10 generates a new `State` object by modifying `sub1` field with a return value of the `createNewSub1` method.

4.6. Consequences

The deeply immutable pattern protects program states from being modified illegally. This pattern also allows to utilize immutable classes for representing changeable program state by generating and passing modified program states.

4.7. Related Pattern

Immutable pattern The immutable pattern treats only a target class without referring classes so that the target class can have relations to mutable classes. Thus, immutable pattern cannot guarantee a set of classes for

representing program states are immutable.

5. Evaluation

We evaluate our extended DPs through the sample software. Table 1 shows the numbers of required classes and relations which indicate inheritances and references, and existence of duplicated code in the state and customizable state patterns, respectively. We consider six new OOP languages on the JVM and two new OOP languages on the .NET Framework: Scala, Kotlin, Xtend, Ceylon, Fantom, Gosu, F# and Nemerle.

Let S be the number of state classes and O be the number of combinations of options, and P be the number of option classes which a state depends on. Note that the numbers of required classes and relations are expressed as formulas with concrete numbers in the case of the sample software.

The implementations with the customizable state pattern have only $S+O+1$ or $S+O$ classes with $S+O+2$ or $S+O$ relations while them without the customizable state pattern has $S+S*O$ or $S+S*O+1$ classes with $S*O$ or $S*O*(P+1)$ relations. Scala, Kotlin, and Ceylon support mixin which is applied when creating objects using an anonymous class so that they remove classes and relations about decorators. The customizable state pattern reduces approximately 50% classes and 33% relations at least for the sample software.

Table 2 shows the numbers of required implementations for immutable classes and of required copy methods. Let F be the number of fields in classes for representing a program state and C be the number of classes for representing a program state.

Xtend and Fantom support annotations for marking immutable classes. Fields in Ceylon and Nemerle are immutable by default. Fantom can check whether an immutable class is deeply immutable at compile-time. Scala and F# automatically generate copy methods for updating immutable objects. Kotlin will support immutability with copy methods similarly to Scala. Thus, several new OOP languages, in particular Scala and F#, can reduce implementation costs for applying the deeply immutable pattern.

Table 1. Evaluation of customizable state pattern

Implementation	Class Relation	Duplication
State /wo multiple inheritance and mixin in Java, Xtend, Nemerle, F#	$S + S * O$ (15) $S * O$ (12)	Exist
State /w multiple inheritance and mixin in C++, Scala, Kotlin, Gosu Ceylon, Fantom	$S + S * O + 1$ (16) $S * O * (P + 1)$ (36)	None
Customizable state with decorator in Java, C++, Xtend, Fantom, Gosu, Nemerle, F#	$S + O + 1$ (8) $S + O + 2$ (9)	None
Customizable state with mixin in Scala, Kotlin, Ceylon	$S + O$ (7) $S + O$ (7)	None

Table 2. Evaluation of deeply immutable pattern

Implementation	Implementation for immutability	Copy method
C++, Java, Kotlin, Gosu	F	C
Xtend, Ceylon, Fantom, Nemerle	0	C
Scala, F#	0	0

6. Conclusions

We proposed the customizable state pattern and the deeply immutable pattern. We found that newer OOP languages such as Scala than traditional ones such as C++ and Java can improve implementations of our DPs.

We plan to improve existing DP such as the GoF DP in new OOP languages with functional features. Moreover, we will find which language features can improve DPs and how the features can improve DP with case study.

References

[1] S. Antoy and M. Hanus. New functional logic design patterns. In *Proceedings of the 20th international conference on Functional and constraint logic programming*, WFLP'11, pages 19–34, Berlin, Heidelberg, 2011. Springer-Verlag.

[2] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional, 1995.

[3] J. a. L. Gomes and M. P. Monteiro. Design pattern implementation in object teams. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 2119–2120, New York, NY, USA, 2010. ACM.

[4] O. Hachani and D. Bardou. Using aspect-oriented programming for design patterns implementation. In *In Proc. Workshop Reuse in Object-Oriented Information Systems Design*, 2002.

[5] J. Hannemann and G. Kiczales. Design pattern implementation in java and aspectj. *SIGPLAN Not.*, 37(11):161–173, Nov. 2002.

[6] R. Lämmel and J. Visser. Design patterns for functional strategic programming. In *Proceedings of the 2002 ACM SIGPLAN workshop on Rule-based programming*, RULE '02, pages 1–14, New York, NY, USA, 2002. ACM.

[7] K. Sakamoto, A. Ohashi, H. Washizaki, and Y. Fukazawa. A framework for game software which users play through artificial intelligence programming (in japanese). *IEICE Transactions*, 95(3):412–424, mar 2012.

[8] N. Schärli, S. Ducasse, O. Nierstrasz, and A. Black. Traits: Composable units of behaviour. In L. Cardelli, editor, *ECOOP 2003 - Object-Oriented Programming*, volume 2743 of *Lecture Notes in Computer Science*, pages 248–274. Springer Berlin Heidelberg, 2003.

[9] G. T. Sullivan. Advanced programming language features for executable design patterns "better pattern through reflection", 2002.

ELCD: an efficient online cycle detection technique for pointer analysis

Fei Liu, Lulu Wang, Bixin Li

School of Computer Science and Engineering, Southeast University, Nanjing, China
Email: {fei_liu, wanglulu, bx.li}@seu.edu.cn

Abstract

Online cycle detection techniques can improve the efficiency and scalability of inclusion-based pointer analysis algorithm. Currently, many cycle detection techniques aggressively seek cycles in the constraint graph and have high overhead. LCD(Lazy Cycle Detection) technique can significantly reduce the overhead of online cycle detection, because it tries to detect cycle only when cycle is likely to be in the constraint graph. It identifies potential cycles based on cycle effect. However, most LCD cycle detection searches find no cycles because the computation of cycle effect only bases on points-to sets of two nodes on one edge, which sometimes can not assure there must be a cycle. In this paper, an improved cycle detection technique is introduced which we call ELCD(Extended Lazy Cycle Detection). Compared to LCD, ELCD can identify potential cycles based on more-refined cycle effect, which is computed based on not only points-to sets of two nodes on one edge but also the points-to set of the successor of destination node of the edge; and significantly reduce the number of cycle detection searches that find no cycles. Experimental results show that ELCD can improve efficiency of LCD without losing points-to information precision.

***Keywords*—online cycle detection; inclusion-based pointer analysis; points-to set; constraint graph**

I. Introduction

Pointer analysis is a static analysis technique. Its goal is to determine statically which memory locations one pointer variable may point to. Pointer analysis is an important technique, because it enables compilers optimizing and program analyzers for languages with pointers such as C/C++ and Java. The existing pointer analysis algorithms are all approximate and each approximate algorithm provides a trade-off between the efficiency and the precision. There are several dimensions that affect efficiency/precision trade-off of the algorithm, among which are context-sensitivity and flow-sensitivity. This paper deals with inclusion-based pointer analysis which is flow-insensitive and context-insensitive.

Inclusion-based pointer analysis is first proposed by Andersen in his ph.d thesis[2]. The primary technique to improve algorithm's efficiency and scalability is *online cycle detection*, which is first proposed in [3]. Online cycle detection techniques dynamically detect cycles in the constraint graph during the analysis and collapses all the nodes in a cycle into a representative node. Since there are several classic graph algorithms which can detect strongly connected component of directed graphs efficiently, the difficulty in online cycle detection lies in when to perform cycle detection and how to control overhead of pointer analysis algorithm. Most online cycle detection techniques aggressively seek out cycles in the constraint graph when new edge are added[6, 7]. However, Hardekopf et al. propose a novel technique of detecting cycles called Lazy Cycle Detection(LCD), which identifies potential cycles based on cycle effect. In this paper, inclusion-based pointer analysis algorithm that employs LCD technique for online cycle detection is called LCD algorithm. The *cycle effect* in LCD algorithm is defined as identical points-to sets of two nodes of an edge. When points-to sets of two nodes of an edge are identical, LCD algorithm thinks that there is a cycle in the constraint graph, and cycle detection is triggered. But LCD algorithm has the following shortcomings: firstly, the majority of cycle detection searches find no cycles; by observing experimental results of LCD algorithm, it can be found that less than 1% searches will find actual cycles[8]; secondly, there are some cycles in the constraint

Supported partially by the National Natural Science Foundation of China under Grant No. 60773105 and no. 60973149, and partially Supported by Doctoral Fund of Ministry of Education of China under Grant No. 20100092110022, and partially by National High Technology Research and Development Program under Grant No.2008AA01Z113.

Correspondence to: Bixin Li, School of Computer Science and Engineering, Southeast University, Nanjing, China. E-mail: bx.li@seu.edu.cn

graph that LCD algorithm can not find. So the efficiency of LCD algorithm may be improved if the number of cycle detection searches which find no cycles can be significantly reduced and more cycles in the constraint graph can be found.

In this paper we introduce an online cycle detection technique called ELCD, which is an improvement of LCD technique. Cycle detection of ELCD is also based on *cycle effect*. The difference between ELCD and LCD lies in the definition of *cycle effect* and the *cycle effect* of ELCD is more refined than that of LCD. Inclusion-based pointer analysis algorithm that employs ELCD for online cycle detection is called ELCD algorithm. Experimental results show that ELCD algorithm can significantly reduce the number of cycle detection searches which find no cycles and improve algorithm's efficiency without losing points-to information precision, compared to LCD algorithm.

The rest of this paper is organized as follows. In Section II we provide relevant background about inclusion-based pointer analysis and LCD algorithm. Section III introduces ELCD algorithm. Section IV describes experimental evaluation. Section V discusses related work. And section VI concludes this paper.

II. Background

This section contains three parts: preliminaries of inclusion-based pointer analysis, inclusion-based pointer analysis algorithm using constraint graph(IBPAC algorithm), and LCD algorithm proposed in [4].

A. Preliminaries of Inclusion-Based Pointer Analysis

In this paper, C language program is our target program to be analyzed. Inclusion-based pointer analysis is first proposed by Andersen in his ph.d thesis[2]. It is based on set constraint analysis and contains two phases: *constraint generation* and *constraint solving*. Constraints are derived from the statements involving pointer variable assignment and function parameter passing in the C program. Constraints derivation process is trivial and basic constraint generation rules are used. Table I illustrates generation rules and constraint types used in inclusion-based pointer analysis. Andersen's algorithm maintains the inclusion constraints in a vector and solves constraints by iterating over constraints until a fixpoint is obtained.

B. IBPAC Algorithm

More inclusion-based pointer analysis algorithms employ constraint graph to represent inclusion constraints. Inclusion-based pointer analysis algorithm using constraint graph is called IBPAC algorithm in this paper. IBPAC algorithm solves inclusion constraints by computing the dynamic transitive closure of constraint graph. In constraint graph G, there is one node for each program variable. Each node has an associated points-to set, and the points-to set of the node is initialized using base constraints: for each $p \supseteq \{q\}$, points-to set of node p contains *loc(q)*, where *loc(q)* denotes the memory location of variable *q*. A direct edge from node *p* to node *s* represents propagation of points-to set from node *p* to node *s*.

TABLE I. Basic constraint generation rules and constraint types

statement	constraint	type
$p = \&q$	$p \supseteq \{q\}$	base
$p = q$	$p \supseteq q$	simple
$p = *q$	$p \supseteq *q$	complex 1
$*p = q$	$*p \supseteq q$	complex 2

IBPAC algorithm is reduced to compute the dynamic transitive closure of the constraint graph by continually propagating points-to sets along constraint graph's edges, adding new edges when appropriate. It is illustrated in Algorithm 1.

Algorithm 1 IBPAC algorithm

```
1:  let G=⟨V, E⟩
2:  W ← V
3:  while W! = ∅ do
4:      n ← SELECT_FROM(W)
5:      for each v ∈ pts(n) do
6:          for each constraint a ⊇ *n do
7:              if v → a ∉ E then
8:                  E ← E ∪ {v → a}
9:                  W ← W ∪ {v}
10:             end if
11:         end for
12:         for each constraint *n ⊇ b do
13:             if b → v ∉ E then
14:                 E ← E ∪ {b → v}
15:                 W ← W ∪ {b}
16:             end if
17:         end for
18:     end for
19:     for each n → z ∈ E do
20:         pts(z) ← pts(z) ∪ pts(n)
21:         if pts(z) changed then
22:             W ← W ∪ {z}
23:         end if
24:     end for
25: end while
```

C. LCD Algorithm

Considering the fact that nodes in the same cycle should have identical points-to sets when IBPAC algorithm terminates, Hardekopf et al. propose LCD algorithm[4], where they propose a novel strategy for online cycle detection: before propagating points-to information along an edge of constraint graph, algorithm checks to see whether the source and destination node already have identical points-to sets or not. If the two nodes have identical points-to set, LCD algorithm uses a depth-first search to check for a possible cycle.

The LCD algorithm is an improvement of IBPAC algorithm, which is illustrated in Algorithm 2. Before a points-to set is propagated from one node to another, the algorithm needs to check whether the following two conditions are satisfied or not: (1) the points-to sets are identical; and (2) the algorithm has not triggered a search on this edge previously. If these conditions are satisfied, LCD algorithm triggers the cycle detection rooted on the destination node. If there exists a cycle, the algorithm collapses all the nodes involved together; otherwise it marks this edge to show that it is not necessary to repeat this attempt later.

Algorithm 2 LCD algorithm

```
1: let G=⟨V, E⟩
2: /* R represents edges on which LCD algorithm has
   already triggered cycle detection */
3: R ← ∅
4: W ← V
5: while W! = ∅ do
6:    Lines 4 to 18 in Algorithm 1 are added here.
7:    for each n → z ∈ E do
8:       if pts(z)=pts(n) ∧ n→ z ∉ R then
9:          DETECT_AND_COLLAPSE_CYCLE(z)
10:         R ← R ∪ {n → z}
11:      end if
12:      pts(z) ← pts(z) ∪ pts(n)
13:      if pts(z) changed then
14:         W ← W ∪ {z}
15:      end if
16:   end for
17: end while
```

LCD technique is lazy because cycle detection is not triggered until the effect of the cycle(identical points-to sets) becomes evident, rather than trying to detect cycles when the final edge that completes the cycle is inserted. The advantage of this technique is that algorithm only attempts to detect cycles when it is likely to find them. The efficiency of LCD algorithm depends on the assumption that two nodes usually have identical points-to sets only because they are in the same cycle. But this assumption does not hold at most time. Because during LCD algorithm execution, when two nodes have identical points-to sets, they are often not in the same cycle. If the assumption does not hold, that is to say, two nodes with identical points-to sets are not in the same cycle, LCD algorithm wastes time attempting to detect non-existent cycles. This is one shortcoming of LCD algorithm, and also leave some room to improve LCD algorithm both in efficiency and other aspects.

III. ELCD Algorithm

Considering the shortcoming of LCD algorithm, we propose an improvement of LCD algorithm, which we call ELCD algorithm. ELCD, that is to say, online cycle detection technique of ELCD algorithm, is also based on *cycle effect*. The difference between ELCD and LCD lies in the definition of *cycle effect*. The *cycle effect* in LCD is defined as identical points-to sets of two nodes of an edge. The *cycle effect* of ELCD is more refined than that of LCD and is defined to satisfy the following two conditions simultaneously: (1) two nodes of one edge have identical points-to sets; and (2) destination node of the edge has at least one immediate successor and the immediate successor has the same points-to set as points-to set of source node of the edge. When these two conditions are satisfied, ELCD algorithm thinks that there is a cycle in constraint graph, and cycle detection search is triggered.

Because *cycle effect* of ELCD algorithm is more refined than that of LCD algorithm, many cycle detection searches, which are triggered in LCD algorithm and find no cycles in the constraint graph, will not be triggered in ELCD algorithm. Compared to LCD algorithm, ELCD algorithm can significantly reduce the number of cycle detection searches which find no cycles and improve algorithm's efficiency without losing points-to information precision. Similar to LCD algorithm, ELCD algorithm does not trigger twice cycle detection searches on the same edge. This restriction avoids making repeated cycle detection searches which find no cycles. And it also indicates that ELCD algorithm is not guaranteed to find all cycles in the constraint graph.

ELCD algorithm is shown in Algorithm 3. Before propagating points-to information along an edge from one node to another, algorithm 3 checks to see if the defined *cycle effect* is satisfied. Only if these conditions are satisfied, ELCD algorithm triggers cycle detection rooted at the destination node. If there exists a cycle, ELCD algorithm collapses all the nodes involved together; otherwise it marks this edge so that algorithm will not repeat the attempt later.

Algorithm 3 ELCD algorithm

```
1: let G=⟨V, E⟩
2: /* R represents edges on which ELCD algorithm has
   already triggered cycle detection */
3: R ← ∅
4: W ← V
5: while W! = ∅ do
6:    Lines 4 to 18 in Algorithm 1 are added here.
7:    for each n → z ∈ E do
8:       if pts(z)=pts(n) ∧ n→ z ∉ R then
9:          if ∃ m, z→ m ∈ E∧ pts(m)=pts(z) then
10:            DETECT_AND_COLLAPSE_CYCLE(z)
11:         end if
12:         R ← R ∪ {n → z}
13:      end if
14:      pts(z) ← pts(z) ∪ pts(n)
15:      if pts(z) changed then then
16:         W ← W ∪ {z}
17:      end if
18:   end for
19: end while
```

TABLE II. Experimental benchmarks*

Name	Description	LOC	Constraints
Emacs-21.4a	text editor	169K	83,213
Ghostscript-8.15	postscript viewer	242K	169,312
Gimp-2.2.8	image manipulation	554K	411,783
Insight-6.5	graphical debugger	603K	243,404
Wine-0.9.21	windows emulator	1,338K	713,065
Linux-2.4.26	linux kernel	2,172K	574,788

* Constraints: the number of constraints generated by the CIL[1] front-end.

IV. Experimental Evaluation

The benchmarks used to compare algorithm's efficiency between ELCD and LCD are a suite of six open-source C programs, which range in size from 169K to 2.17M LOC. The benchmarks are described in Table II. We perform the experiments on a quad 2.40GHz processor with 3.24GB of memory, using the Ubuntu 9.10 Linux distribution(VMWare platform). All executables are compiled using gcc-4.4.1 and the '-O3' optimization flag.

A. Experimental Framework

Figure 1 describes the framework of the experiments conducted in this paper. Original constraint file is pre-processed in offline optimization analysis phase, and then reduced constraint file is used as algorithm's input of both ELCD and LCD.

If original constraint file generated from benchmark source code by the CIL front-end[1] is directly used as algorithm's input, both ELCD and LCD can only deal with benchmarks of emacs and gs. Larger benchmarks(such as inst, wine and lnx) run out of memory before analysis algorithm completes. So in our experiments, original constraint files are first pre-processed by offline optimization analysis techniques hvn and hru+le[5], and then reduced constraint files are used as algorithm's input.

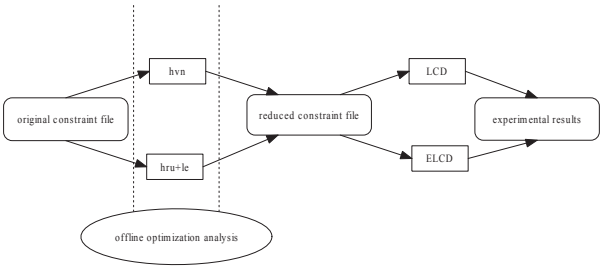


Fig. 1. Experimental Framework. hvn and hru+le each representing different offline optimization technique[5].

B. Experimental Results

1) Offline optimization analysis: Table III describes time overhead of pre-processing original constraint files using hvn and hru+le. Table IV shows the number of original constraints and the number of constraints generated by hvn and hru+le. From table III and IV, we can find that hvn and hru+le can significantly reduce the number of constraints in original constraint files with little time consumption.

2) ELCD and LCD: Considering that two offline optimization techniques are used in offline analysis phase, this section presents five corresponding tables to describe experimental results of LCD and ELCD. Table V and Table VI correspond to experimental results of LCD and ELCD under hvn. The experimental results of LCD and ELCD coupled with hru+le technique are shown in table VII and table VIII. Table IX describes efficiency improvement of LCD algorithm under hvn and hru+le.

Corresponding parameters in tables V-VIII are explained as follows: time(sec) represents algorithm's execution time; collapsed_nodes denotes number of all nodes which are collapsed into strongly connected components; false_num shows number of cycle detection searches which find no cycles; good_num represents number of cycle detection searches which actually find cycles in the constraint graph; scc denotes number of strongly connected components in the final constraint graph; total_num shows number of cycle detection searches triggered by analysis algorithm.

TABLE III. Time(s) of pre-processing original constraint files

	emacs	gs	gimp	inst	wine	lnx
time under hvn	0.24	0.516	1.484	0.784	3.2802	1.9801
time under hru+le	0.416	2.14	3.62	4.4762	10.6087	9.4885

TABLE IV. Number of constraints generated by hvn and hru+le

	emacs	gs	gimp	inst	wine	lnx
original constraints	83,213	169,312	411,783	243,404	713,065	574,788
constraints by hvn	21,460	67,310	96,483	85,375	171,237	203,732
constraints by hru+le	7,829	46,301	49,870	48,325	110,285	108,942

C. Experimental Analysis

Figure 2 and 3 show the comparison of the performance of ELCD with LCD under hvn and hru+le. For benchmarks, execution time of ELCD is almost less than that of LCD algorithm under hvn and hru+le, except for inst under hvn and lnx under hru+le. Figure 4 compares improvement of LCD under hvn and hru+le.

$$improvement = \frac{(t_{LCD} - t_{ELCD})}{t_{LCD}} * 100\%$$

On average, ELCD can make improvement of LCD by 7.51% under hvn and 5.66% under hru+le.

From experimental results, we can find the following observations and conclusions:

- Scc and collapsed_nodes of ELCD and LCD algorithm are almost identical under offline optimization technique hvn and hru+le, except for gs benchmark under hvn. ELCD frequently has the same number of collapsed nodes as LCD while always triggering less cycle detection attempts. These data shows the evidence that ELCD has the potential to perform better than LCD.
- Total_num and False_num of ELCD algorithm are less than those of LCD algorithm under hvn and hru+le. For benchmarks used in this paper, the difference between execution time of ELCD and LCD algorithm is relatively unobvious. This is because of this fact that the sizes of points-to sets of these benchmarks are very large. Large points-to sets increase the overhead of repeatedly performing set equality when ELCD algorithm executes.
- Improvements of LCD under different offline optimization techniques are different. This is because reduced constraint files are obviously different.

V. Related Work

Research of inclusion-based pointer analysis algorithm is mainly focused on how to improve efficiency and scal-

TABLE V. LCD with hvn

	time(s)	scc	collapsed_nodes	false_num	good_num	total_num
emacs	2.3401	5305	5999	6679	45	6724
gs	12.8128	28294	13563	16009	102	16111
gimp	36.9303	45466	19165	26230	59	26289
inst	32.8541	29359	21001	34501	182	34683
wine	901.908	78596	56082	66337	122	66459
lnx	268.193	86179	43555	82453	426	82879

TABLE VI. ELCD with hvn

	time(s)	scc	collapsed_nodes	false_num	good_num	total_num
emacs	2.0321	5305	5999	1438	40	1478
gs	11.7927	28293	13564	2411	94	2505
gimp	33.0621	45466	19165	7217	51	7268
inst	33.3421	29359	21001	9486	139	9625
wine	850.945	78596	56082	10260	108	10368
lnx	243.211	86179	43555	19121	361	19482

TABLE VII. LCD with hru+le

	time(s)	scc	collapsed_nodes	false_num	good_num	total_num
emacs	0.008	9076	561	214	43	257
gs	0.292	32808	4336	1892	44	1936
gimp	0.056	50291	4222	1003	30	1033
inst	2.0201	34118	6255	4923	72	4995
wine	2.9362	110823	15966	3900	42	3942
lnx	3.0522	91539	11717	4857	304	5161

TABLE VIII. ELCD with hru+le

	time(s)	scc	collapsed_nodes	false_num	good_num	total_num
emacs	0.004	9076	561	39	40	79
gs	0.28	32808	4336	514	35	549
gimp	0.056	50290	4223	300	26	326
inst	1.8561	34118	6255	1811	61	1872
wine	2.7042	110823	15966	820	36	856
lnx	4.1563	91539	11717	1301	275	1576

TABLE IX. Improvement of LCD under hvn and hru+le*

	improvement under hvn	improvement under hru+le
emacs	13.16%	50.00%
gs	7.96%	4.11%
gimp	10.47%	0.00%
inst	-1.49%	8.12%
wine	5.65%	7.90%
lnx	9.31%	-36.17%
average	7.51%	5.66%

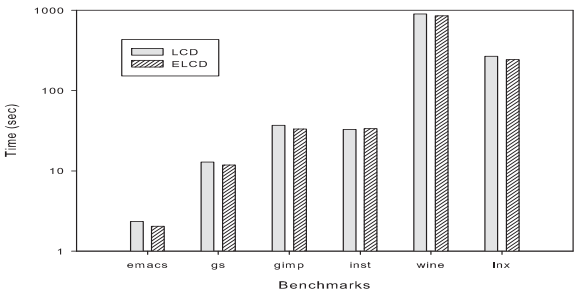


Fig. 2. Performance of ELCD versus LCD under hvn.

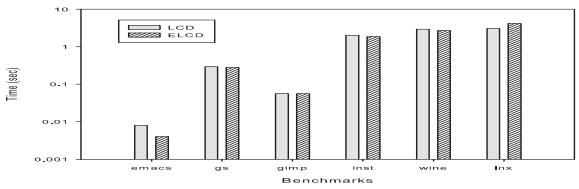


Fig. 3. Performance of ELCD versus LCD under hru+le.

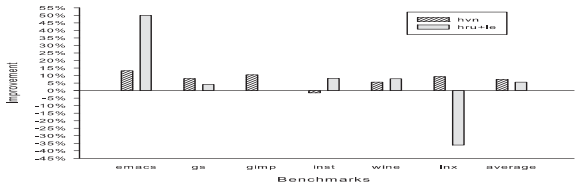


Fig. 4. Improvement of LCD under hvn and hru+le.

ability without losing points-to information precision. The existing methods used to improve algorithm's efficiency and scalability mainly contain: online cycle detection techniques and offline optimization techniques.

Online cycle detection techniques Faehndrich et al.[3] first propose online cycle detection technique. Their work performs partial online cycle detection. Heintze and Tardieu[6] performs cycle detection by graph reachability queries. Pearce et al.[7] dynamically maintains a topological ordering of the constraint graph. Only a newly-inserted edge that violates the current ordering could possibly create a cycle, so cycle detection and topological re-ordering are performed only in this case. Fernando Magno Quintao Pereira et al.[8] propose two new algorithms for solving inclusion-based pointer analysis, such as *wave propagation* and *deep propagation*. The *wave propagation* method separates the insertion of new edges in constraint graph and the propagation of points-to sets based on Pearce et al.'s algorithm[7]. The *deep propagation* algorithm maintains the invariant that, if a node w is reachable from a node v, then the points-to set of w contains the points-to set of v.

Offline optimization techniques An offline analysis is a static analysis performed prior to the actual pointer analysis. Hardekopf et al.[5] propose three offline optimization analysis techniques which dramatically reduce both the time and memory consumption of subsequent algorithm. The hvn(hash-based value numbering) technique gives each direct node a pointer equivalence label such that two nodes share the same label only if their points-to sets are identical. The hru technique extends hvn technique by interpreting the union and dereference operators. The le(location equivalence) technique assigns location equivalence labels such that le(x)=le(y) iff x and y always belong to the same points-to sets.

VI. Conclusion and Future Work

In this paper, we have discussed how to perform online cycle detect efficiently for inclusion-based pointer analysis. The main contributions include: an efficient online cycle detection technique called ELCD is proposed, which is based on more refined definition of *cycle effect*, compared with LCD technique; a series of experiments are conducted to show that ELCD algorithm can significantly reduce the number of cycle detection searches which find no cycles and improve LCD algorithm's efficiency without losing points-to information precision.

Future work should include:(1) compare algorithm's efficiency of ELCD and LCD on other benchmarks to verify the speculation: whether the execution time of ELCD is far less than that of LCD algorithm when ELCD and LCD are run on benchmarks whose points-to sizes are moderate. (2) further improve ELCD algorithm to enable ELCD to find more cycles in the constraint graph.

References

[1] cil. http://www.cs.berkeley.edu/ necula/cil/.

[2] Lars Ole Andersen. *Program analysis and specialization for the C programming language*. PhD thesis, DIKU, University of Copenhagen, May 1994.

[3] Manuel Fähndrich, Jeffrey S. Foster, Zhendong Su, and Alexander Aiken. Partial online cycle elimination in inclusion constraint graphs. In *Programming Language Design and Implementation*, pages 85–96, 1998.

[4] B. Hardekopf and C. Lin. The ant and the grasshopper: fast and accurate pointer analysis for millions of lines of code. In *Programming Language Design and Implementation*, pages 290–299, 2007.

[5] Ben Hardekopf and Calvin Lin. Exploiting pointer and location equivalence to optimize pointer analysis. In *Static Analysis Symposium*, pages 265–280, 2007.

[6] Nevin Heintze and Olivier Tardieu. Ultra-fast aliasing analysis using CLA: a million lines of C code in a second. In *Programming Language Design and Implementation*, pages 254–263, 2001.

[7] D.J. Pearce, P.H.J. Kelly, and C. Hankin. Online cycle detection and difference propagation for pointer analysis. In *Source Code Analysis and Manipulation*, pages 3–12, 2003.

[8] Fernando Magno Quintao Pereira and Daniel Berlin. Wave propagation and deep propagation for pointer analysis. In *Code Generation and Optimization*, pages 126–135, 2009.

Exploring Ensemble-Based Data Preprocessing Techniques for Software Quality Estimation

Kehan Gao
Eastern Connecticut State University
gaok@easternct.edu

Taghi M. Khoshgoftaar
Florida Atlantic University
khoshgof@fau.edu

Amri Napolitano
Florida Atlantic University
amrifau@gmail.com

***Abstract*—Software quality modeling uses software metrics such as code-level measurements and defect data in order to build classification models for identifying potentially-problematic program modules. The prediction accuracy of these classification models is heavily influenced by the quality of the input data. Two problems which can affect such data are high dimensionality (having an extremely large number of independent attributes, or features) and class imbalance (having one class with many more members than the other class). In this paper, we present a novel form of ensemble learning based on boosting that incorporates data sampling to alleviate class imbalance and feature (software metric) selection to resolve high dimensionality. As we adopt two different sampling methods (Random Undersampling (RUS) and Synthetic Minority Oversampling (SMOTE)) in the technique, we have two forms of our new ensemble-based approach: SelectRUSBoost and SelectSMOTEBoost. To evaluate the effectiveness of these new techniques, we apply them to a group of datasets from a real-world software system. We use four learners and nine feature selection techniques to build our models. In addition, we consider versions of the technique which do not incorporate feature selection, and compare all four techniques (the two different ensemble-based approaches which utilize feature selection and the two versions which use sampling only). The experimental results demonstrate that SelectRUSBoost is more effective in improving classification performance than SelectSMOTEBoost, and that the techniques with feature selection result in better prediction than using the techniques without feature selection.**

***Index Terms*—software defect prediction, software metrics, feature selection, data sampling, ensemble learning.**

I. Introduction

Software measurement data (such as code metrics, execution traces, historical code changes, and defect databases) collected during the software development process contain valuable information about a software project's status, progress, quality, and evolution. Practitioners can use this software measurement data along with various data mining techniques to perform software defect prediction, the process which utilizes software metrics and defect data in order to build classification models which estimate the quality of program modules (e.g., classify program modules as either fault-prone (*fp*) or not-fault-prone (*nfp*)) [1]. With such predictions, project resources such as quality-assurance time can be effectively allocated to the problematic modules. For example, the potentially-faulty modules receive more inspection and testing, resulting in better quality of the product.

Many studies have shown that prediction accuracy is affected by the quality of the input data. In this paper, we are interested in investigating two common problems, *high dimensionality* and *class imbalance*, that appear in many software measurement datasets. High dimensionality occurs when too many variables (features or software metrics) are available for building classification models. Several problems may arise due to high dimensionality, including longer learning time of a classification algorithm, a decline in prediction performance of a classification model, and the creation of complex models which are difficult for humans to interpret. Previous research has shown that filter-based feature ranking techniques are simple, fast, and effective methods for dealing with this problem [2].

In the context of software quality engineering, class imbalance occurs when not-fault-prone (*nfp*) modules significantly outnumber fault-prone (*fp*) modules in a given dataset. Generally, for a binary classification problem, two types of errors can occur: Type I and Type II[1]. In software engineering, a Type II error is usually far more expensive than a Type I error, as the cost of a Type I error may involve wasted effort resulting from inspecting high quality program modules, while a Type II error may indicate a missed opportunity for correcting a faulty module prior to system deployment and operation. Class imbalance usually results in more prediction errors of Type II, especially when the different costs of misclassifications are ignored during the modeling process. Data sampling, a process which modifies the input data by removing majority-class instances or adding minority-class instances to reduce the class imbalance, is frequently used to cope with this problem.

To alleviate the adverse impacts of high-dimensional, imbalanced data on the prediction models, in this paper we propose a new technique which integrates data sampling, feature selection, and ensemble learning (boosting). We use two different sampling methods (Random Undersampling (RUS) and Synthetic Minority Oversampling (SMOTE)), and thus we have two forms of this ensemble-based approach, SelectRUSBoost and SelectSMOTEBoost. In addition, we build our models using nine various filter-based feature ranking techniques, resulting in 18 different ensemble-based methods in total. We also consider the techniques which do not use feature selection (denoted RUSBoost and SMOTEBoost).

To assess the effectiveness of the proposed method, we apply the techniques to four datasets from a real-world large legacy telecommunications software system, all of which exhibit significant class imbalance between the two classes (*fp* and *nfp*). In particular for all four datasets, the proportion of the *fp* class ranges from 1% to 7% of total number of modules. The post-sampling class ratios is set to 50:50, meaning that after sampling the new dataset will contain the same number of *fp* and *nfp* modules. Four different learners are used to build classification models. We compare the two forms of the ensemble-based approach, SelectRUSBoost vs. SelectSMOTEBoost, and we also compare the techniques using feature selection with those which do not. The experimental results demonstrate that SelectRUSBoost results in better or similar prediction as SelectSMOTEBoost. In addition, the proposed techniques demonstrated much better classification behavior when feature selection is used in the data preprocessing step.

The rest of the paper is organized as follows. Section II discusses related work. Section III provides methodology, including

[1]A Type I error or misclassification refers to a *nfp* module misclassified as *fp*, while a Type II error refers to a *fp* module misclassified as *nfp*.

more detailed information about the feature selection, data sampling, ensemble-based techniques, learners, and performance metric applied in this study. The datasets 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

Feature selection is an effective technique to resolve the problem of high dimensionality, and as such it has been heavily researched. Liu et al., in their research [3], provide a comprehensive survey of feature selection and review its developments with the growth of data mining. Numerous variations of feature selection have been employed in a range of fields. Ilczuk et al. [4] highlight the importance of attribute selection in judging the qualification of patients for cardiac pacemaker implantation. In the context of text mining, Forman [5] investigates multiple filter-based feature ranking techniques.

In addition to excess number of attributes, many real-world datasets are plagued with the class imbalance problem. A considerable amount of research has been done to investigate this problem. Weiss et al. [6] provide a survey of the class imbalance problem. Two important techniques discussed for alleviating the problem of class imbalance are data sampling and boosting. Barandela et al. [7] and Han et al. [8] examine the performance of some "intelligent" data sampling techniques such as SMOTE, Borderline SMOTE, and Wilson's Editing.

Although boosting, a common form of ensemble learning, is not specifically developed to handle the class imbalance problem, it has been shown to be very effective in this regard [9]. The most commonly used boosting algorithm is AdaBoost [10]. Several variations [11], [12] have been proposed to make AdaBoost cost-sensitive or to improve its performance on imbalanced data.

While a great deal of work has been done for feature selection and data sampling separately, limited research has been done and reported on both together. In one of our recent studies [13], we use data sampling together with feature selection to deal with the high dimensionality and class imbalance problems in the context of software quality classification. The experimental results demonstrate that using feature selection along with data sampling is more effective than using each technique individually for improving software defect prediction.

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.

1) Standard Techniques: The six filter-based feature ranking techniques used in this work include: chi-squared (CS), information gain (IG), gain ratio (GR), two types of ReliefF (RF and RFW), and symmetrical uncertainty (SU). All six use the implementation found in the WEKA tool[2] [14], and use default parameters unless otherwise noted.

The chi-squared (CS) test is used to examine whether two variables are independent. CS is more likely to find significance to the extent that (1) the relationship is strong, (2) the sample size is large, and/or (3) the number of values of the two associated features is large. Information gain, gain ratio, and symmetrical uncertainty are measures based on the concept of entropy from information theory. Information gain (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, compared with the impurity of the complete set of data. A drawback of IG is that it tends to prefer attributes with a larger number of possible values, even if it is actually no more informative. One strategy to counter this problem is to use the gain ratio (GR), which penalizes multiple-valued attributes. Symmetrical uncertainty (SU) is another way to overcome the problem of IG's bias toward attributes with more values, doing so by dividing by the sum of the entropies of X and Y. Relief is an instance-based feature ranking technique. ReliefF is an extension of the relief algorithm that can handle noise and multiclass datasets. When the `WeightByDistance` (weight nearest neighbors by their distance) parameter is set as default (false), the algorithm is referred to as RF; when the parameter is set to true, the algorithm is referred to as RFW.

Algorithm 1: Threshold-Based Feature Selection

input :
1. Dataset $S = \{(\mathbf{x}_i, y_i) | i = 1, \ldots, m \text{ and } \ y_i \in \{P, N\}\}$ with features $F^j, j = 1, \ldots, n$, 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 $\omega \in \{\text{AUC, PRC}\}$.

output: Ranking $\mathbb{R} = \{r^1, \ldots, r^n\}$ where r^j represents the rank for attribute F^j, i.e., the r^j-th most significant attribute as determined by metric ω.

for $F^j, j = 1, \ldots, n$ **do**
  Normalize $F^j \mapsto \widehat{F}^j = \frac{F^j - \min(F^j)}{\max(F^j) - \min(F^j)}$;
  Calculate metric ω using attribute $\widehat{F}^j$ and class attribute $\{y_i |\ y_i \in \{P, N\},\ i = 1, \ldots, m\}$, $\omega(\widehat{F}^j)$; (The detailed formula of each metric ω is provided in Section III-A2.)

Create attribute ranking $\mathbb{R}$ using $\omega(\widehat{F}^j)\ \forall j$

2) Threshold-Based Feature Selection: The threshold-based feature selection (TBFS) technique was proposed by our research team and implemented within WEKA. The procedure is shown in Algorithm 1. Each independent attribute works individually with the class attribute, and this two-attribute dataset 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 three rates[3]. 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. Two metrics used in this study are presented as follows:

a. **Area Under the ROC Curve** (AUC). 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 (between 0 and 1) used to transform the normalized attribute values into a predicted class. AUC is used to

[2]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.

[3]Analogous to the procedure for calculating rates in a classification setting with a posterior probability, the true positive rate, TPR(t) and false positive rate, FPR(t) can be calculated at each threshold $t \in [0, 1]$ relative to the normalized attribute $\widehat{F}^j$. Precision, PRE(t) is defined as the fraction of the predicted-positive examples which are actually positive.

provide a single numerical metric for comparing the predictive power of each attribute.

b. **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 y-axis and precision on 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) [15] 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 $\text{S2N} = \frac{(\mu_P - \mu_N)}{\sigma_P + \sigma_N}$, where μ_P and μ_N are the mean values of a particular attribute for the samples from class P and class N, and σ_P and σ_N are the corresponding standard deviations. The larger the S2N value, the more relevant the feature is to the class attribute.

B. Sampling Techniques

The two data sampling techniques used in this study are random undersampling (RUS) and synthetic minority oversampling technique (SMOTE). RUS alleviates the problem with class imbalance in a dataset by randomly discarding instances from the majority class (*nfp* class for our study). SMOTE is an intelligent oversampling method proposed by Chawla et al. [16]. SMOTE adds new, artificial minority examples by extrapolating between preexisting minority instances rather than simply duplicating original examples. The newly created instances cause the minority regions of the feature-space to become fuller and more general.

C. Ensemble-Based Data Preprocessing Techniques

1) SelectRUSBoost: Before introducing the SelectRUSBoost technique, we would like to discuss an ensemble-based sampling technique called RUSBoost [12] first, as the SelectRUSBoost technique is a combination of feature selection with the RUSBoost algorithm. 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 [10], a well known boosting algorithm shown to improve the classification performance of weak classifiers. Initially, all examples in the training dataset 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, RUS is applied to the training data to achieve a more balanced class distribution.

The SelectRUSBoost technique is simply the integration of feature selection into RUSBoost. Prior to building each model, the chosen feature selection approach is applied, and then the model is built. Note that this feature selection takes place after the random undersampling step, which means that each of the separate models may have different features selected. Let $\mathbf{x}_i$ be a point in the feature space X and y_i be a class label in the set of class labels Y. Each of the m examples in the dataset (S) can be represented by the tuple $(\mathbf{x}_i, y_i)$. Let t be an iteration between 1 and the maximum number of iterations T (number of classifiers in the ensemble), h_t be the weak hypothesis (trained using some classification algorithm, L) trained on iteration t, $h_t(\mathbf{x}_i, y_i)$ be the output of hypothesis h_t for instance $(\mathbf{x}_i, y_i)$ (which may be a numeric confidence rating). Let $D_t(i)$ be the weight of the ith example on iteration t (the weights are typically normalized to sum to one). Algorithm 2 presents the complete SelectRUSBoost algorithm.

Algorithm 2: SelectRUSBoost

input : Dataset $S = \{(\mathbf{x}_i, y_i) | i = 1, \ldots, m \text{ and } y_i \in Y\}$ with minority class $y^r \in Y$, $|Y| = 2$
: Weak learner, L
: Feature ranking technique, R
: Number of features to select, k
: Number of iterations, T
: Desired percentage of total instances to be represented by the minority class, N

output: Final hypothesis for predicting the class of example $\mathbf{x}$, $H(\mathbf{x})$

Initialize $D_1(i) = \frac{1}{m}$ for all i.

for $t \leftarrow 1$ **to** T **do**

1) Create temporary training dataset S'_t with distribution D'_t using random undersampling.
2) Apply R to dataset S'_t.
3) Remove all features from S'_t which are not within the top k of R.
4) Call L, providing it with examples S'_t and their weights D'_t.
5) Get back a hypothesis $h_t : X \times Y \rightarrow [0, 1]$.
6) Calculate the pseudo-loss (for S and D_t):

$$\varepsilon_t = \sum_{(i,y):y_i \neq y} D_t(i)(1 - h_t(\mathbf{x}_i, y_i) + h_t(\mathbf{x}_i, y))$$

7) Calculate the weight update parameter:

$$\alpha_t = \frac{\varepsilon_t}{1 - \varepsilon_t}$$

8) Update D_t:

$$D_{t+1}(i) = D_t(i)\alpha_t^{\frac{1}{2}(1 + h_t(\mathbf{x}_i, y_i) - h_t(\mathbf{x}_i, y:y \neq y_i))}$$

9) Normalize D_{t+1}:

$$\text{Let } Z_t = \sum_i D_{t+1}(i)$$

$$D_{t+1}(i) = \frac{D_{t+1}(i)}{Z_t}$$

Output the final hypothesis:

$$H(\mathbf{x}) = \underset{y \in Y}{\text{argmax}} \sum_{t=1}^{T} h_t(\mathbf{x}, y) \log \frac{1}{\alpha_t}$$

2) SelectSMOTEBoost: SelectSMOTEBoost has the same mechanism as SelectRUSBoost. It is the integration of feature selection into SMOTEBoost. Analogous to RUSBoost, SMOTEBoost is produced with the process of combining synthetic minority oversampling (SMOTE) to boosting with the purpose of improving classification performance. For the complete SelectSMOTEBoost algorithm, one can refer to Algorithm 2, just replacing RUS with SMOTE.

D. Learners

In this paper, four learners are used: multilayer perceptron (MLP), k-nearest neighbor (KNN), support vector machine (SVM), and logistic regression (LR). All models were built using WEKA with default parameters unless otherwise noted.

Multilayer perceptron is a neural network of simple neurons called perceptrons. Some related parameters of MLP were set as follows. The `hiddenLayers` parameter was set to 3 to define a network with one hidden layer containing three nodes. The `validationSetSize` parameter was changed to 10 to cause the classifier to leave 10% of the training data aside to be used as a validation set to determine the stopping point for the iterative training process.

K-nearest neighbor is a type of instance-based learning, or "lazy" learning approach. No actual model is built using the training data; instead, the training data itself is the model. To predict the class of an unlabeled test instance, the model computes the distance from that instance to all the instances in the training dataset (using an appropriate distance metric such as Euclidean distance) and locates the nearest neighbors (in this study, we find the five nearest neighbors). These neighbors are grouped by their class values, and are then weighted by the value `1/Distance`. The total weight of each class is found, and the class with the greatest weight is assigned as the predicted class label for the instance being classified.

Support vector machines are a classification algorithm built from the assumption that both classes are linearly separated from each other. This assumption allows us to use a discriminant to split the instances into the two classes. A linear discriminant uses the formula $g(x|\mathbf{w}, \omega_0) = \mathbf{w}^T x + \omega_0$. In the case of the linear discriminant the only data that needs to be learned is the weight vector, $\mathbf{w}$ and the bias ω_0. One aspect that must be addressed is that there can be multiple discriminants that correctly classify the two classes. SVM is a linear discriminant classifier which assumes that the best discriminant maximizes the distance between the two classes. This is measured in the distance from the discriminant to the samples of both classes. In WEKA, the SVM classifier is implemented as SMO. For SMO, we alter two of the parameters from the default. The complexity parameter, `c`, is set to 5.0 (the default is 1.0) and the `buildLogisticModels` parameter is set to true in order to produce proper posterior probability estimates.

Logistic regression is a statistical technique that can be used to solve binary classification problems. Based on the training data, a logistic regression model is created, which is used to decide the class membership of future instances.

E. 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. 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 detection rate and false alarm rate. 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. An attribute with more predictive power results in an AUC value closer to 1.

IV. Datasets

For this study, we conduct our experiments on four datasets from a very large legacy telecommunications software system (denoted as LLTS). The LLTS software system was developed in a large organization by professional programmers using PROTEL, a proprietary high level procedural language (similar to C). The system consists of four successive releases of new versions of the system, and each release was comprised of several million lines of code. The data collection effort used the Enhanced Measurement for Early Risk Assessment of Latent Defect (EMERALD) system [17]. A decision support system for software measurements and software quality modeling, EMERALD periodically measures the static attributes of the most recent version of the software code. We refer to these four releases as SP1, SP2, SP3, and SP4. Each set of associated source code files is considered as a program module. The LLTS datasets consist of 42 software metrics, including 24 product metrics, 14 process metrics, and 4 execution metrics. The dependent variable is the class of the software module, *fp* or *nfp*. The fault-proneness is based on a selected threshold, in particular, modules with one or more faults are considered as *fp*, *nfp* otherwise. Table I summarizes the numbers of the *fp* and *nfp* modules and their percentages in each dataset.

TABLE I
DATASET SUMMARY

Dataset	*nfp*		*fp*		Total
	#	%	#	%	
SP1	3420	93.7	229	6.3	3649
SP2	3792	95.3	189	4.7	3981
SP3	3494	98.7	47	1.3	3541
SP4	3886	97.7	92	2.3	3978

V. Experiments

In this experiment, we apply the SelectRUSBoost and SelectSMOTEBoost (abbreviated as 'SRB' and 'SSB' respectively in the figures and tables) techniques to the four datasets described above and compare the classification models of the two techniques. We also consider the simple RUSBoost (RB) and SMOTEBoost (SB) methods in which no feature selection is involved and use these as the baseline for evaluating the new techniques. Because the datasets are highly imbalanced, we do not consider approaches without RB or SB to address the class balance problem.

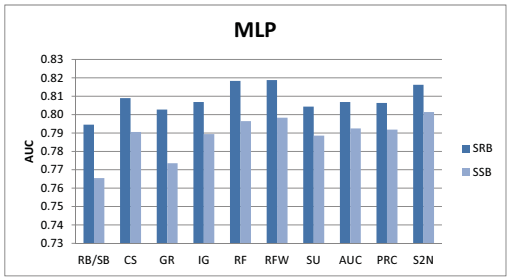
The results averaged over the four datasets (in terms of AUC) are reported in Figure 1, which contains four subfigures that describe the classification performance for four learners, MLP, KNN, SVM, and LR, respectively, each averaged across all four datasets. Every subfigure shows the results of SRB and SSB over each of the nine rankers as well as the one where no ranking technique is used (i.e., the RB and SB techniques). Note that the post-sampling ratio between *fp* and *nfp* is 50:50 after RUS or SMOTE is performed. The number of the features selected in the feature subsets is set to $\lceil \log_2 n \rceil = 6$, where n is the number of independent attributes in the original dataset ($n = 42$ in this experiment). The results show the following points.

- For the MLP and KNN learners, the SRB technique performed much better than SSB for every ranker involved. Also, RB showed much better performance than SB.
- For the SVM and LR learners, the SRB technique presented similar performance as the SSB method. RB continued to demonstrate significantly better performance than SB for the SVM learner, but had quite similar behavior to SB for the LR algorithm.

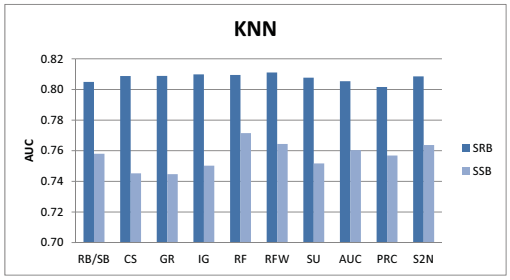
We also carried out a statistical analysis using the unpaired two tailed t-test to compare the SRB technique with the SSB method. The t-test examines the null hypothesis that the population means related to two independent group samples are equal against the

TABLE II
CLASSIFICATION PERFORMANCE FOR LLTS DATASETS

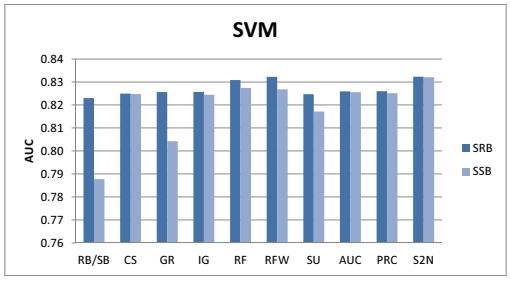
Ranker	MLP			KNN			SVM			LR		
	SRB	SSB	p-value	SRB	SSB	p-value	SRB	SSB	p-value	SRB	SSB	p-value
RB/SB	0.7945	0.7654	**0.002**	0.8050	0.7580	**0.000**	0.8230	0.7878	**0.000**	0.7650	0.7654	0.983
CS	0.8090	0.7906	**0.000**	0.8088	0.7452	**0.000**	0.8250	0.8248	0.925	0.8226	0.8250	0.310
GR	0.8028	0.7736	**0.000**	0.8088	0.7447	**0.000**	0.8256	0.8042	**0.000**	0.8255	0.8041	**0.000**
IG	0.8069	0.7894	**0.002**	0.8099	0.7502	**0.000**	0.8257	0.8244	0.619	0.8231	0.8238	0.801
RF	0.8183	0.7965	**0.000**	0.8095	0.7715	**0.000**	0.8308	0.8274	0.143	0.8300	0.8266	0.184
RFW	0.8188	0.7983	**0.000**	0.8111	0.7644	**0.000**	0.8322	0.8268	**0.018**	0.8282	0.8260	0.400
SU	0.8043	0.7886	**0.008**	0.8077	0.7517	**0.000**	0.8247	0.8171	0.067	0.8229	0.8168	0.147
AUC	0.8068	0.7925	**0.008**	0.8054	0.7604	**0.000**	0.8259	0.8255	0.853	0.8236	0.8247	0.644
PRC	0.8063	0.7918	**0.007**	0.8017	0.7569	**0.000**	0.8260	0.8251	0.640	0.8215	0.8255	0.078
S2N	0.8162	0.8014	**0.003**	0.8086	0.7637	**0.000**	0.8323	0.8321	0.932	0.8310	0.8319	0.736



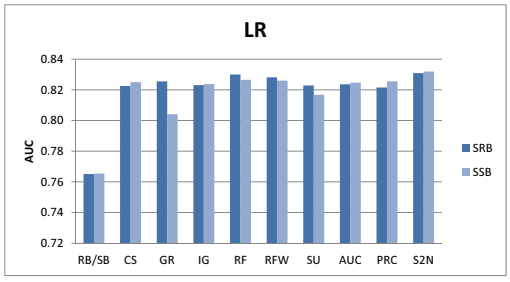
(a) MLP



(b) KNN



(c) SVM



(d) LR

Fig. 1. Classification performance for LLTS datasets

TABLE III
ANOVA FOR LLTS DATASETS

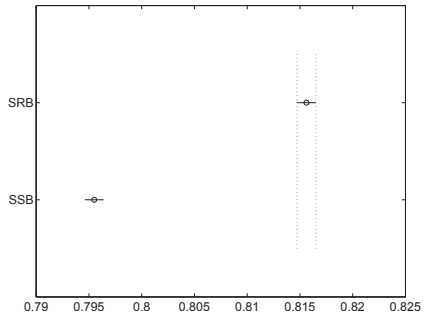
Source	Sum Sq.	d.f.	Mean Sq.	F	p-value
A (sampler)	0.3239	1	0.3239	489.26	0.000
B (learner)	0.8560	3	0.2854	431.01	0.000
C (ranker)	0.2488	9	0.0276	41.75	0.000
Error	2.1092	3186	0.0007		
Total	3.5379	3199			

alternative hypothesis that the population means are different. The underlying assumptions of the t-test were examined and validated prior to statistical analysis. Both the AUC results and this statistical analysis are presented in Table II, with the p-values provided for each pair of comparisons. The significance level is set to 0.05. When the p-value is less than 0.05, the two group means are significantly different from each another, and are highlighted with **bold** in the table. As can be seen, for the MLP and KNN learners, SRB performed better than SSB at significance level of 0.05 for all cases; for the SVM and LR learners, however, SRB and SSB demonstrated similar classification performance in most cases except for the GR ranker and RFW ranker (for SVM) as well as the no ranker case (for SVM), where SRB continued performing better than SSB.

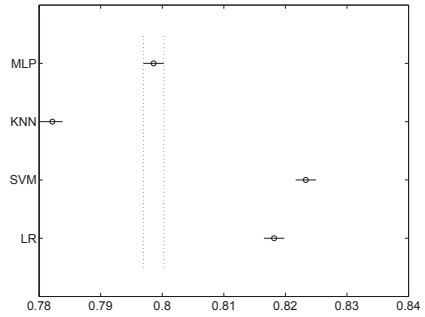
We also conducted a three-way analysis of variance (ANOVA) F test on the classification performance to examine if the performance difference (better/worse) is statistically significant or not. Three factors are designed as follows: Factor A represents the two versions of the ensemble-based method, Factor B represents the four learners used in this study, and Factor C represents the ten feature ranking techniques (including the case in which no ranking technique is involved). The null hypothesis for the ANOVA test is that all the group population means are the same, while the alternate hypothesis is that at least one pair of means is different. Table III 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.

We further carried out a multiple comparison test on each main factor with Tukey's honestly significant difference (HSD) criterion. Figure 2 shows the multiple comparisons for Factors A, B, and C. The figures display graphs with each group mean represented by a symbol (○) 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:

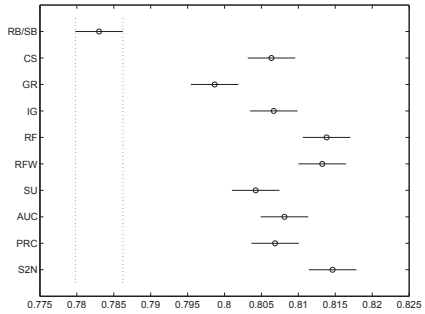
- SRB performed significantly better than SSB.
- For the four learners, the order in terms of their performance from highest to lowest is SVM, LR, MLP, and KNN.



(a) Factor A: Sampler



(b) Factor B: Learner



(c) Factor C: Ranker

Fig. 2. Multiple comparison for LLTS datasets

- Among the nine filter-based feature ranking techniques, S2N, RF, and RFW performed better than other rankers. CS, IG, SU, AUC, and PRC demonstrated average performance. GR performed worst. The study also shows that the classification models produced significantly better predictions when using a feature selection technique compared to the RB or SB techniques where no feature selection was considered.

VI. Conclusion

High dimensionality and class imbalance are the two major problems affecting software defect prediction datasets. In this study, we proposed two forms of an ensemble-based data preprocessing technique, SelectRUSBoost and SelectSMOTEBoost, to overcome these two problems. The new technique is the integration of feature selection, data sampling and ensemble learning (boosting). In our experiments, we applied these techniques to a group of datasets from a very large legacy telecommunications software system. We evaluated the effectiveness of the new techniques and compared the two forms, each in conjunction with nine filter-based feature ranking techniques as well as the case of no feature selection. We built classification models using four learners. The results demonstrate that the SelectRUSBoost technique performed better than or similarly to the SelectSMOTEBoost method. Also, the techniques always demonstrated better performance when feature selection is taken into account. In addition, among the nine feature selection techniques, signal to noise ratio and ReliefF resulted in better prediction performance than other rankers. Of the four learners, support vector machine and logistic regression performed better than multilayer perceptron and k-nearest neighbor. 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] T. M. Khoshgoftaar, K. Gao, and A. Napolitano, "An empirical study of feature ranking techniques for software quality prediction," *International Journal of Software Engineering and Knowledge Engineering*, vol. 22, no. 2, pp. 161–183, 2012.

[3] 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.

[4] G. Ilczuk, R. Mlynarski, W. Kargul, and A. Wakulicz-Deja, "New feature selection methods for qualification of the patients for cardiac pacemaker implantation," in *Computers in Cardiology, 2007*, Durham, NC, USA, 2007, pp. 423–426.

[5] G. Forman, "An extensive empirical study of feature selection metrics for text classification," *Journal of Machine Learning Research*, vol. 3, pp. 1289–1305, March 2003.

[6] G. M. Weiss, "Mining with rarity: A unifying framework," *SIGKDD Explorations*, vol. 6, no. 1, pp. 7–19, 2004.

[7] R. Barandela, R. M. Valdovinos, J. S. Sanchez, and F. J. Ferri, "The imbalanced training sample problem: Under or over sampling?" *In Joint IAPR International Workshops on Structural, Syntactic, and Statistical Pattern Recognition (SSPR/SPR'04), Lecture Notes in Computer Science 3138*, no. 806-814, 2004.

[8] H. Han, W. Y. Wang, and B. H. Mao, "Borderline-SMOTE: A new over-sampling method in imbalanced data sets learning," in *International Conference on Intelligent Computing (ICIC'05). Lecture Notes in Computer Science 3644*. Springer-Verlag, 2005, pp. 878–887.

[9] C. Seiffert, T. M. Khoshgoftaar, J. Van Hulse, and A. Napolitano, "Building useful models from imbalanced data with sampling and boosting," in *Proceedings of the* 21^{st} *International Florida Artificial Intelligence Research Society Conference (FLAIRS-2008)*. Coconut Grove, Florida: AAAI Press, May 2008, pp. 306–311.

[10] 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.

[11] Y. Sun, M. S. Kamel, A. K. C. Wong, and Y. Wang, "Cost-sensitive boosting for classification of imbalanced data," *Pattern Recognition*, vol. 40, no. 12, pp. 3358–3378, 2007.

[12] 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.

[13] T. M. Khoshgoftaar, K. Gao, and L. A. Bullard, "A comparative study of filter-based and wrapper-based feature ranking techniques for software quality modeling," *International Journal of Reliability, Quality and Safety Engineering*, vol. 18, no. 4, pp. 341–364, May 2011.

[14] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd ed. Morgan Kaufmann, 2005.

[15] 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.

[16] 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.

[17] J. P. Hudepohl, S. J. Aud, T. M. Khoshgoftaar, E. B. Allen, and J. Mayrand, "EMERALD: Software metrics and models on the desktop," *IEEE Software*, vol. 13, no. 5, pp. 56–60, September 1996.

Comparison of SRGMs and NNEs on Multiple Data Sets

Catherine Stringfellow Sreya Reddy Raaji Vedala-Tiramula Swetha Myneni
Department of Computer Science
Midwestern State University
Wichita Falls, TX 76308
catherine.stringfellow@mwsu.edu

***Abstract*-Several studies have been performed using SRGMs and neural nets to predict the number of total defects in a system. Most of these studies are limited in that they were only applied to data from one system; for example, SRGMs were applied to a large medical record system and neural nets were applied to a real-time control system. In this study, both techniques, including some alternative SRGMs, are applied to data from two different types of systems. This paper describes detailed results of the techniques and concludes by suggesting the best suitable one for the given data sets.**

Keywords-defect prediction; SRGMs; neural nets

I. INTRODUCTION

Testing is an essential activity in software development to ensure the quality of software products. Earlier studies estimated testing can consume fifty percent, or even more, of the development costs [2]. For large and complex software, testing becomes even more critical due to the lack of practical techniques for proving software correctness [8].

Testing is also one of the basic methods of assessing the reliability of software. Various software reliability models have also been proven effective for estimating the remaining defects in the software [6, 14, 16, 17]. This in turn helps in making release decisions during testing, which may consider tradeoffs between cost, time and quality. Software reliability growth models or SRGMs are essentially curves which fit the experimental defect data and have traditionally been used to approximate the growth of reliability during testing. Different studies have concluded that certain software reliability growth models or SRGMs are better than others, but those studies may not have considered some that have worked well in other studies. Most recently, it has been proposed to apply neural nets to make defect predictions, as they do not have the same problems with ensuring assumptions are met.

This paper is organized as follows: Section 2 discusses the various methods and techniques used for making software development decisions, specifically making predictions about the number of defects so as to be able to determine if a software product is ready for release. Section 3 describes the two data sets used in previous case studies and the software reliability growth models that were used on the two different data sets. Section 4 presents the results of running alternative SRGMs on those data sets, as well as applying neural nets to those data sets and compares those techniques in terms of relative error. Section 5 summarizes the paper.

II. BACKGROUND

Many proposed SRGMs are used to estimate the relationship between software reliability and time and other factors. These models are divided into two main categories, parametric and non-parametric [18]. Parametric models estimate the models' parameters based on the assumptions about the nature of software faults, behavior of the software failure process and the development environments. Non-parametric models, such as neural networks, are not only more flexible as they predict reliability metrics based only on fault history, they often do so with better predictive quality than parametric models. Both kinds of models have been applied to various data sets to predict the remaining defects in the software product and have been show to work quite well in actual practice.

Various techniques have been used to select the optimal SRGMs. Almering, van Genuchten, Cloudt and Sonnemans applied a Laplace trend test and log-log plots to their failure log data to determine which SRGMs to investigate [1]. Their results show the Log Poisson execution time model (PET) and the Log power model (LP) performed best as soon as 20% of the testing effort was completed, the GO model was good at 42% and the delayed S-shaped model did not start performing well until 71% of the testing effort had been completed.

Sharma, Garg, Nagpal and Garg propose a distance based approach (DBA) method to evaluate, select and rank seven SRGMs based on four common criteria often used to compare the models [11]. They applied the DBA method to two data sets. The best SRGMS for their data using this method were the P-N-Z model, the Yamada imperfect debugging model, the Yamada exponential and then the G-O model. The delayed S-shaped model ranked seventh. The data in their paper is used as data set 4 in this study.

Hewett, Kulkarni, Seker and Stringfellow identify SRGMs that are effective at estimating remaining defects using defect data collected during system testing in software

development [5]. They used post release defect data to validate models. Rather than trying to determine from the beginning which models to apply, since the underlying model assumptions are often violated in practice, they apply several as the failure/fault data comes in and then empirically make decisions as to which models to continue with and which ones to reject. The strategy applied various software reliability models starting after 60% of the expected testing effort. Selected models fit the data well, as measured by goodness of fit or R^2 value, the prediction stability, and the difference between the actual and predicted number of defects. The difficulty is that if the initial choice of the SRGM is wrong, then the estimates only become worse. These conclusions were drawn by conducting the study on one kind of data, i.e., those obtained from a medical record database system [14].

Hewett, Kulkarni, Stringfellow and Andrews attempted to predict time for fixing defects during testing using three learning algorithms for data analysis- decision tree learner, Naïve Bayes classifier and a neural network approach [6]. They used data from [14] and identified thirty attributes, twelve of interest and in particular *fixing time*, the one they wanted to predict. The results of all the predictive models from the various algorithms were promising - the average of the best accuracies was over 90%. Estimating fixing time has potential benefits for planning and scheduling in software testing management. The authors, however, only applied the techniques to one release of their data. Using an *ensemble* of neural networks may also improve the estimates.

There have been several studies that have experimented with neural networks to assess reliability. Sitte determined that neural networks were better or just as good as parametrically-recalibrated SRGMS and were easier to use [13]. Cai, Cai, Wang, Wu, and Zhang investigated the back propagation network method (BPNN) to predict next failure time [3]. They experimented with changing the neural net architecture and concluded the effectiveness depends on the nature of the data.Tian and Noore used an evolutionary approach to find the optimal architecture with good results [15]. Khatri, Trivedi, Kant and Dembla integrated the Goel-Okumoto SRGM with a neural network approach using stochastic differential equations appropriate for data where the number of faults decreases as testing progresses, since defects are detected and eliminated [7]. They state that this approach can applied to any SRGM to improve it. Singh and Kumar performed a comparison analysis on feed-forward neural networks (FFNS) and three parametric SRGMs using seven data sets. Their results showed that FFNNs had better prediction capability [12].

Using an *ensemble* of neural networks (NNE) may also improve estimates. Hansen and Salamon implement an NNE that performs well in classifying a given input pattern using a consensus scheme to decide the collective classification [4]. Zheng also compared NNEs to single neural nets and three SRGMs (G-O, Duane, S-Shaped) using two data sets collected from a command and control application and a single-user workstation. Results showed that the NNE achieved lower prediction errors [18].

Only a few studies have applied both kinds of models to various data sets to predict the remaining defects in the software product [6, 7, 12, 13]. In this paper, two different data sets (one from three releases of a large medical record system and another from a real time control system) are used. Various software reliability growth models and neural networks (NNs) are applied to the data sets.

III. APPROACH

This section describes the data sets and the replication approach used in this study.

A. Data

Data sets 1-3 for this study come from three releases of a large medical record system as reported in [14], while data set 4 came from a real-time control system [11]. The medical record system originally had 173 software components and 15 components were added over the three releases. System testers entered defect reports for each defect found into a defect tracking system; the date of each defect was automatically recorded. This study (and the study in [14]) grouped the number of defects found by week. The number of cumulative defects found each week was used as input data into the defect estimation models.

Data set 4 includes, for each failure, the time between failure (TBF) and the cumulative time between failures (CTBF), respectively. As data set 4 is really large, this study combined failures in groups of 500 cumulative time units.

The following sections describe this replication study, which applies the defect estimation methods to the data sets on which they had not been previously applied.

B. Replication Approach

First, the SRGM models described in [14], as well as the Duane and the Y-Inflection models [11] were applied to data set 4 [11].

Second, the Duane and Inflection S-shaped SRGM models were applied to data sets 1-3 from [14]. Formulas for these models are presented in [11]. Zheng claims that the Duane model and the Inflection S-shaped model both work well on the data in their study [18]. This study seeks to show the external validity of these models by running them on other data.

Last, neural nets are run on all four data sets. The Neural Net toolbox from MATLAB provides four neural net functions

that may be appropriate for making predictions [9]. Zheng used forward-feed neural nets (FFNNs) and Elman nets [18]. This study uses the newff() function (a FFNN) and the newlrn function (which supersedes the Elman net in the toolbox). In addition this study uses two delay networks that are recommended for making predictions [4]. Table 1 briefly describes the neural net functions used in this study. These neural net functions were applied to all four data sets.

TABLE I. NEURAL NETWORK FUNCTIONS LIST.

Function	Purpose
newdtnn()	Creates distributed time delay network.
newff()	Creates feed-forward back propagation network.
newfftd()	Creates feed- forward input time delay propagation network.
newlrn()	Creates layered- recurrent network.

IV. RESULTS

In order to determine the best techniques for predicting the number of defects in a system, a variety of SRGMs and neural nets were applied to the four data sets from two different systems. Table 2 shows the best of the results of applying the SRGM techniques and the neural nets to data sets 1-4. In accordance to [14], models are recommend to be rejected (indicated by an **R)** if the goodness of fit of the their estimates, that is the R-value, is below 0.95 or if their estimates are less than the actual number of failures found in a test week – with the goal of being conservative. Models are not considered to be useful until their estimates stabilize, that is, when an estimate is within 10% of the previous week's estimate (indicated by an **S**, and a **D** if they destabilize). The second column shows the test week the models were applied and the third the actual number of defects found. The following sections provide a discussion on the results.

A. Results for alternative SRGMS on data sets 1-3

Table 2 shows the results of the best original SRGMs [14] and the alternative Duane and Inflection S-shape SRGMs on data sets 1-3. The best of all models and good estimates are highlighted. The best original SRGM model(s) performed better than the SRGMs not originally applied to the data. The Duane model worked well, but its estimates do not approach an asymptote and so may greatly overestimate, depending on how many weeks in advance one needs to predict. The Inflection S-shaped model was rejected in data sets 2 and 3, because it underestimated or diverged. In data set 1, it did not perform well in estimating the number of defects that would eventually be found (which is more important to know than the number of defects already found). Figures 1-3 show the results of the SRGMs on data sets 1-3, respectively.

B. Results for SRGMs on Data Set 4

Table 2 also shows the results of applying SRGMS on data set 4. Of the original SRGM models [14], the Gompertz model performed the best, although according to the empirical selection model in [14], it would have been rejected for underestimating in week 28. It did stabilize in week 29, but occasionally underestimated failures until week 33. It had very good R-values ranging from 0.985 to 0.989. The G-O Musa model diverged. The delayed S-shaped model underestimated and had a bad R-value. The Duane model performed very well, while the Inflection S-shaped model was overly pessimistic. Of the six models, the Inflection S-shaped model was the only acceptable model according to the criteria in [14], although if one more week had elapsed before applying the models, the Gompertz and the Duane models would have been the best. This study suggests that the empirical SRGM selection model proposed in [14] should be modified, so as not to reject an SRGM too early just because it is underestimating defects by just a little. The tolerance for underestimating could be increased as testing time elapses. Figure 4 shows the results of the SRGMs on data set 4.

C. Results for NN on both data

The last four columns of Table 2 show the results of applying four neural net functions on the four data sets. Newdtdnn, newlrn turned to be the first and second best functions on all data sets. For data set 1 newdtdnn and newlrn networks estimated 211 and 297, respectively – these should be compared to the actual number of defects (231). In data set 2 these functions estimated 269 and 408 (compare to 245) and in data set 3 they estimated 103 and 88 (compare to 83). Finally on data set 4 the newdtdnn and newlrn networks estimated 128 and 120 (compare to 136). The best estimates are highlighted in Table 2. The estimates could never really be said to be stable, although the newdtdnn did perform the best. Figures 1-4 show the results of the two best neural nets on data sets 1-4.

TABLE II. RESULTS OF MODELS APPLIED TO DATA SETS.

Data Set			Best of old SRGMs				New SRGMs				NNs			
	Test Week	Failures Found	Delayed S-shaped		Gompertz		Duane		Inflection S-shaped		newdtdnn	newlrn	newfftd	newff
			Est	R-val	Est	R-val	Est	R-val	Est	R-val	Est	Est	Est	Est
	11	139	830	0.953	2431	0.970	142	0.953	347	0.959				
	12	152	562	0.962	796	0.975	**157S**	0.964	205	0.976				
	13	164	451	0.970	412	0.979	171	0.970	**205S**	**0.976**				
1	14	164	345	0.972	276	0.979	179	0.970	188	0.979	47	38	43	47
	15	165	287	0.971	227	0.979	184	0.967	180	0.981	36	36	48	48
	16	168	255	0.972	**207S**	0.979	188	0.964	177	0.983	62	68	49	52
	17	170	**236S**	0.973	197	0.980	192	0.961	177	0.984	223	180	219	106
	18	176	226	0.974	193	0.981	196	0.960	177	0.986	211	297	332	156
	post28	211					302							
	post38	231					408							
	Test Week	Failures Found	Yamada Exponential		Gompertz		Duane		Inflection S-shaped		newdtdnn	newlrn	newfftd	newff
			Est	R-val	Est	R-val	Est	R-val	Est	R-val	Est	Est	Est	Est
	11	192	262	0.967	200	0.964	205	0.959	200	0.971				
	12	192	**283S**	0.970	198	0.966	**207S**	0.951	203S	0.971				
2	13	192	284	0.970	197	0.969	207	0.943	210	0.971	58	63	48	57
	14	192	**320D**	0.971	196	0.969	207	0.934	198	0.972	47	49	53	51
	15	203	286	0.972	**197R**	0.969	210	0.937	**200R**	0.973	54	80	94	52
	16	203	**265S**	0.973	198	0.969	211	0.937	202	0.974	154	204	120	168
	17	204	249	0.973	199	0.969	213	0.937	204	0.975	269	408	310	347
	post26	245					238							
	Test Week	Failures Found	Delayed S-shaped		Gompertz		Duane		Inflection S-shaped		newdtdnn	newlrn	newfftd	newff
			Est	R-val	Est	R-val	Est	R-val	Est	R-val	Est	Est	Est	Est
	9	70	84	0.980	77	0.983	73	0.971	-	**'-R**	38	38	39	38
3	10	75	**86S**	0.984	**80S**	0.986	**78S**	0.974	-	-	35	35	38	35
	11	76	85	0.986	81	0.988	81	0.973	78	0.987	31	41	40	63
	12	76	84	0.987	80	0.989	84	0.968	**78S**	0.988	85	62	41	44
	13	77	83	0.988	80	0.990	85	0.964	78	0.989	103	88	105	96
	post13	83					85							
	Test Week	Failures Found	Delayed S-shaped		Gompertz		Duane		Inflection S-shaped		newdtdnn	newlrn	newfftd	newff
			Est	R-val	Est	R-val	Est	R-val	Est	R-val	Est	Est	Est	Est
	28	119	**103R**	0.95	**114R**	0.988	**118R**	0.991	148	0.994				
	29	123	**105S**	0.945	**118S**	0.986	**120S**	0.992	**150S**	0.992				
	30	123	107	0.942	121	0.985	122	0.992	153	0.992				
	31	123	108	0.94	123	0.985	124	0.993	155	0.991				
	32	127	110	0.937	126	0.985	126	0.993	158	0.991				
4	33	129	111	0.935	129	0.985	127	0.994	161	0.99				
	34	129	113	0.993	131	0.985	129	0.994	163	0.99				
	35	129	114	0.932	132	0.985	131	0.994	164	0.99				
	36	130	115	0.932	134	0.986	132	0.994	165	0.99				
	37	130	117	0.932	134	0.986	133	0.994	166	0.99				
	38	132	118	0.931	136	0.987	135	0.994	168	0.991				
	39	133	119	0.931	136	0.987	136	0.994	168	0.991				
	40	133	120	0.931	137	0.987	137	0.994	169	0.991				
	41	134	121	0.931	138	0.989	138	0.994	170	0.991	36	36	37	36
	42	135	122	0.931	138	0.988	139	0.994	171	0.991	36	36	35	36
	43	135	123	0.931	139	0.988	140	0.994	171	0.992	34	29	52	41
	44	135	123	0.932	139	0.989	141	0.993	171	0.992	51	45	47	110
	45	136	126	0.932	139	0.989	142	0.993	172	0.992	128	120	80	117

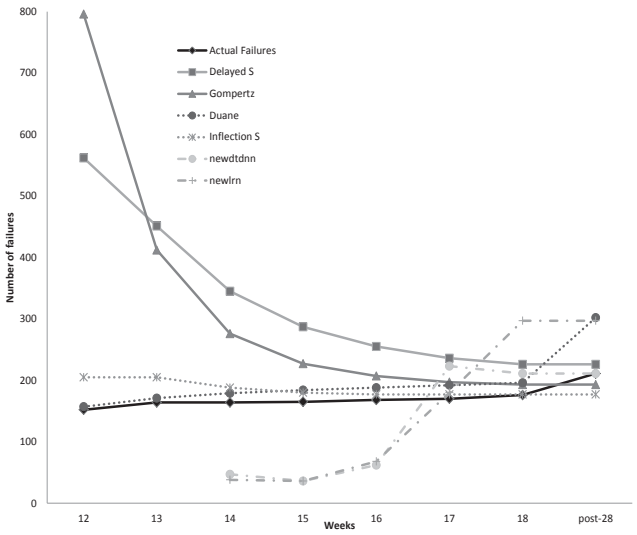


Figure 1. SRGMs and NNs on data set 1.

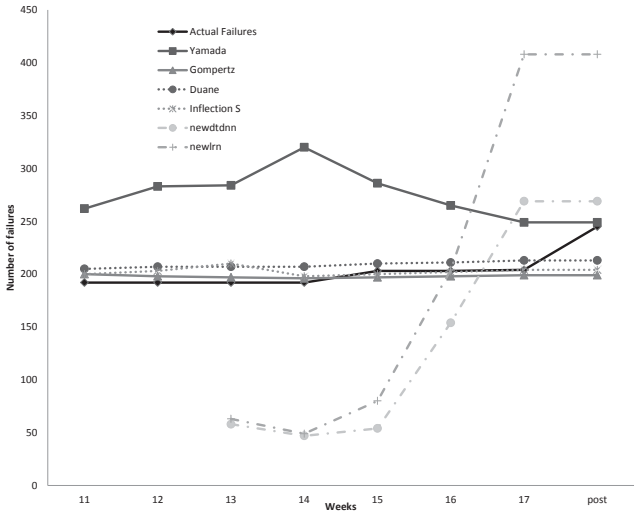


Figure 2. SRGMs and NNs on data set 2.

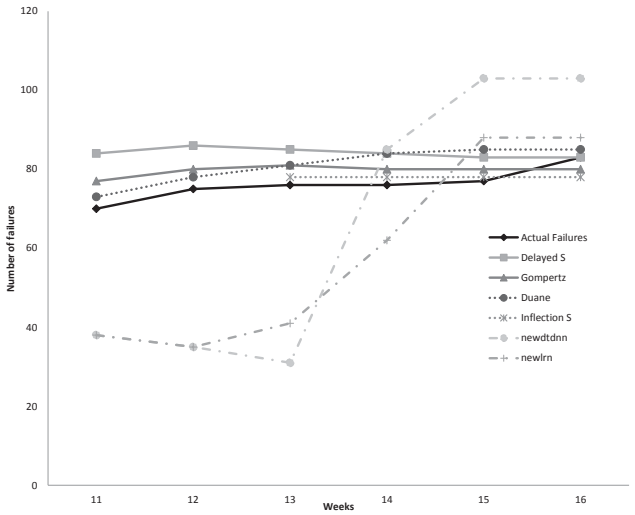


Figure 3. SRGMs and NNs on data set 3.

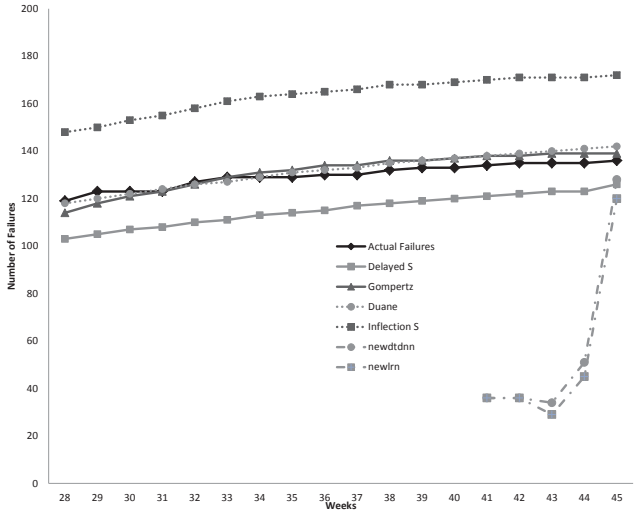


Figure 4. SRGMs and NNs on data set 4.

V. CONCLUSION

Software reliability growth models (SRGMs) provide good predictions of the total number of failures or the number of remaining defects. This study found that at least two SRGMs models work well on all four data sets. While the Duane model performed well in fitting the data and predicting defects a few weeks in advance, it does not work well too much farther than that. This study concludes that the alternative SRGMs could be included in the empirical SRGM selection model from [14], but the SRGM selection model would have to be a bit more tolerant of underestimations in earlier weeks.

Neural nets worked well in [11], and showed promise in this study when applied to data in [14] *as long as enough* testing time had elapsed. Finally among all neural net functions used, newdtnn() was the best, followed by newlrn(). The NNs seem to require more data and more testing time to have elapsed, probably because some of the data is used for training the net, and the rest is used for prediction. Perhaps data from [14] should be analyzed on a daily basis, rather than a weekly basis to get more data points. Another possibility to improve the predictions is to alter the parameters in the neural network functions, such as the number of hidden layers and the number of nodes in the layers. This is work for the future.

REFERENCES

[1] Almering, V., Genuchten, M., Cloudt, G., Sonnemans, P., "Using software reliability growth models in practice," IEEE Software, Nov/Dec 2007, pp. 82-88.

[2] Bezeir, B., Software Testing Techniques, 2nd ed., Van Nostrand Reinhold Co., New York, NY, 1990.

[3] Cai, K., Cai, L., Wang, W., Zhou, Y., Zhang, D., "On the neural network approach in software reliability modeling," J. Systems and Software, vol 58, 2001, pp. 47-62.

[4] Hansen, K., Salamon, P., "Neural network ensembles", IEEE Transactions on Pattern Analysis and Machine Intelligence, 12(10), Oct. 1990, pp. 993-1001.

[5] Hewett, R., Kulkarni, A., Seker, R., and Stringfellow, C., "On effective use of reliability models and defect data in software development," Proceedings of 2006 IEEE Region 5 Technology and Science Conference, San Antonio, TX, April 2006, pp. 67-71.

[6] Hewett, R., Kulkarni, A., Stringfellow, C., Andrews, A., "Software defect data and predictability for testing schedules," Intl Conf on Software Engineering and Knowledge Engineering, Redwood City, CA, July 2006, pp. 499-504.

[7] Khatri, S., Trivedi, P., Kant, S., Dembla, N., "Using artificial neural-networks in stochastic differential equations based software reliability growth modeling," J. Software Engineering and Applications, 4, 2011, pp. 596-601.

[8] Littlewood, B., Strigimi, L., "Validation of ultra high dependability for software-based systems," CACM, 36(11), 1993, pp. 69-80.

[9] Neural Net Toolbox, MatLab.

[10] Musa, J., Iannino, A., and Okumoto, K., Software Reliability: Measurement, Prediction, Application, McGraw-Hill, Inc., New York, NY, USA, 1987.

[11] Sharma, K., Garg, R., Nagpal, C., Garg, R., "Selection of optimal software reliability growth model using distance based approach," IEEE Trans. on Reliability, 29(2), June 2010, pp. 266-276.

[12] Singh, Y., Kumar, P., "Application of Feed-Forward Neural Networks for Software Reliability Prediction," *ACM SIGSOFT Software Engineering Notes,* (35)5, Sept. 2010, pp. 47-62.

[13] Sitte, R., "Comparison of software-reliability-growth predictions: neural networks vs parametric-recalibration," IEEE Trans. on Software Reliability, 48(3), Sept 1999, pp. 285-291.

[14] Stringfellow, C., Andrews, A., "An empirical method for selecting software reliability growth models," Empirical Software Engineering, 7(4), Dec. 2002, pp. 319-343.

[15] Tian, L., "Evolutionary neural network modeling for software cumulative failure time prediction," J. Reliability Engineering and System Safety, 87, 2005, pp. 45-51 .

[16] Yamada, S., Ohba, M., and Osaki, A., "S-shaped reliability growth modeling for software error detection," IEEE Transaction on Reliability, 32(5) 1983, pp. 475-478.

[17] Yamada, S., Ohtera, H., Narihisa, H., "Software reliability growth models with testing effort," IEEE Transaction on Reliability, 35(1), 1986, 19-23.

[18] Zheng, J., "Predicting software reliability with neural network ensembles", Expert Systems with Applications, 36(2), Mar. 2009, pp. 2116-2122.

HESA: The Construction and Evaluation of Hierarchical Software Feature Repository

Yue Yu, Huaimin Wang, Gang Yin, Xiang Li, Cheng Yang
National Key Laboratory for Parallel and Distributed Processing
School of Computer Science, National University of Defense Technology
Changsha, China
yuyue_whu@foxmail.com, whm_w@163.com, {jack.nudt, shockleylee}@gmail.com

***Abstract*—Nowadays, the demand for software resources on different granularity is becoming prominent in software engineering field. However, a large quantity of heterogeneous software resources have not been organized in a reasonable and efficient way. Software features, a kind of important knowledge for software reuse, are ideal materials to characterize software resources. Our preliminary study shows that the effectiveness of feature-related tasks, such as software resource retrieval and feature location, will be greatly improved, if a multi-grained feature repository is available. In this paper, we construct a *Hierarchical rEpository of Software feAture* (HESA), in which a novel hierarchical clustering approach is proposed. For a given domain, we first aggregate a large number of feature descriptions from multiple online software repositories. Then we cluster these descriptions into a flexible hierarchy by mining their hidden semantic structures. Finally, we implement an online search engine on HESA and conduct a user study to evaluate our approach quantitatively. The results show that HESA can organize software features in a more reasonable way compared to the classic and the state-of-the-art approaches.**

***Keywords*—Software reuse; Mining Software repository; Feature-ontology; Clustering;**

I. Introduction

Software reuse is widely recognized as an effective way to increase the quality and productivity of software [1]. With the development of software industry, the degree of software reuse is deeper than previous years and the demand for resources on different granularity becomes more prominent. For example, when developing a new large software system, we may reuse some API calls from the third party to accomplish the core functions and the mature open source software as the basic framework. Additionally, some code fragments or components can be reused to meet other additional demands. The reusable resources are multi-grained, consisting of API calls (the finest level of granularity), code fragments, components (higher than API calls) and software systems (much higher than others).

However, considering the large-scale, heterogeneous and multi-grained software resources, it is a great challenge for stakeholders to retrieve the suitable one. With the evolution of open source ecosystems, more than 1.5 million open source software projects are now hosted in open source communities [2]. Reusable resources [3] are manifold, including code bases, execution traces, historical code changes, mailing lists, bug databases and so on. All of these valuable resources have not been reorganized in a reasonable and efficient way to assist in the activities of software development.

As a kind of attributes which capture and identify commonalities and differences in a software domain, software feature [4] [5] is an ideal material to characterize the software resources. Constructing a feature repository of a flexible structure can make a great contribution to multi-grained reuse.

However, classic feature analysis techniques, such as Feature Oriented Domain Analysis (FODA) [6] and Domain Analysis and Reuse Environment (DARE) [7], are heavily relied on the experience of domain experts and plenty of market survey data. Hence, the feature analysis is a labor-intensive and error-prone process.

In recent years, more and more stakeholders develop, maintain and share their software products on the Internet. In order to promote their products to users, project managers write some market-oriented summaries, release notes and feature descriptions on the profile pages via natural language. The large number of online software profiles can be treated as a kind of repository containing a wealth of information about domain-specific features. Although researchers propose several automatic methods to mine features from the web repository [8] [9] [10], the problems have not completely be solved, specifically in organizing features as flexible granularity.

In this paper, we are trying to address the above problems by proposing a novel approach to construct a *Hierarchical rEpository of Software feAture* (HESA). First of all, we extract a massive number of feature descriptions from online software profiles and mine their hidden semantic structure by probabilistic topic model. Then, we present an *improved Agglomerative Hierarchical Clustering* (iAHC) algorithm, seamlessly integrated with the topic model, to build the feature-ontology of HESA. Finally, we implement an online search engine[1] for HESA to help retrieve features in a multi-grained manner, which can support multiple reuse requirements. By conducting a user study, we demonstrate the effectiveness of our system with quantitative evaluations comparing to the classic and the state-of-the-art approaches.

The rest of this paper is or organized as follows. Section II introduces the overview of our work. Section III describes how to construct HESA in detail. Experiments and analysis can be found in Section IV. Finally, we present some related

[1] http://influx.trustie.net

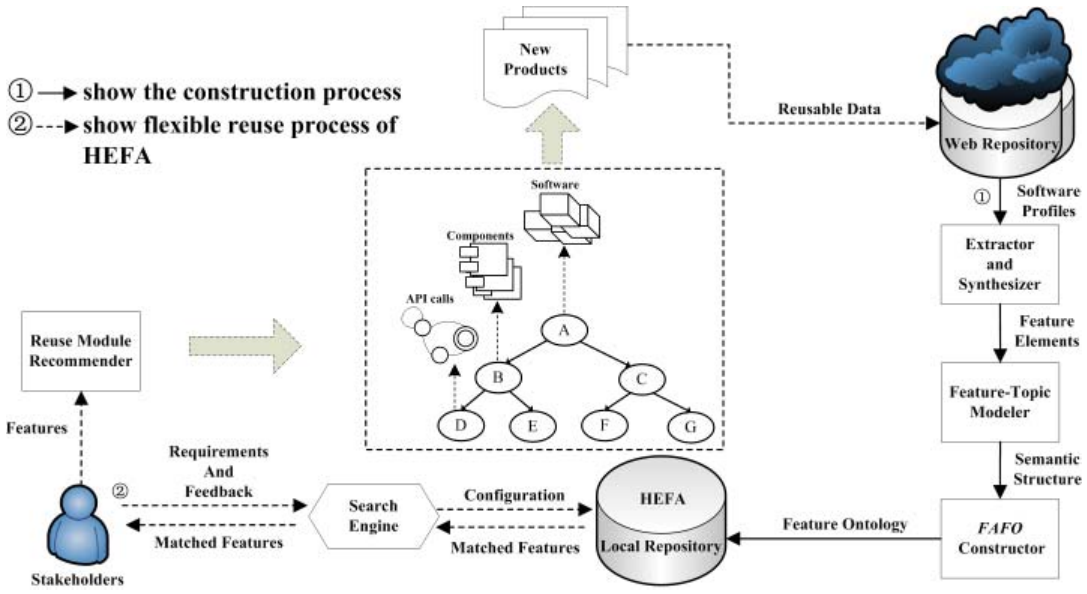


Figure 1. Overview of the construction and use of HESA

work in Section V and draw our conclusions in Section VI.

II. Approach Overview

First of all, we give the definitions of some concepts used in this paper.

Feature element: Feature element is a kind of raw descriptions which can indicate a functional characteristic or concept of the software product.

Feature: Feature is an identifier of the cluster about feature elements, where the cluster is an intermediate output of *improved Agglomerative Hierarchical Clustering* (iAHC).

Feature-ontology: Feature-ontology is a kind of hierarchical structure induced from feature elements by iAHC.

HESA: The assembly of all feature-ontologies of the different categories is the *Hierarchical rEpository of Software feAture*.

The objective of this paper is to build a hierarchical structure of feature as a flexible granularity. In the top layers, the features in coarse granularity may be mapped to the corresponding software resources such as mature applications, design patterns, and superclasses. In the bottom layers, the features can be mapped to some related code fragments, API calls and subclasses.

Before describing the specific details of the underlying algorithms, an architectural overview of approach will be provided as below. There are actually two processes concerning the application of our method, i.e., the construction process and the use process of HESA by stakeholders. In this paper, we only focus on the construction process owing to space limitations.

As depicted in Figure 1, the construction process consists of three primary modules and the input is software profiles data collected and updated continuously by a web crawler. The first module is called the *Extractor and Synthesizer*. We use the Extractor component to extract feature elements. Then, after running preprocessing tasks, the Synthesizer component will automatically located these feature elements, into a unified category. Especially, the word "domain" will be replaced by "category" for they are sharing the same meaning in the rest of this paper.

The second module, the *Feature-Topic Modeler*, is responsible for mining the semantic structure hidden in feature elements. We will merge the synonymic feature elements in terms of their semantic structure in the next step.

The last module, the *FAFO (Flexible grAnularity Feature-Ontology) Constructor*, is a critical part of the construction process. In this module, we present a novel algorithm iAHC to construct the feature-ontology and more details can be found in Section III. The major functionalities of this module are listed as below:

(1) The synonymic feature elements are merged based on the semantic structure outputted by the Feature-Topic Modeler;

(2) For each cluster, a significant group of feature elements is selected as the medoid used to generate feature;

(3) A feature-ontology is learned and features can be retrieved in terms of flexible granularity.

After all the raw data under our category are disposed, the construction process of HESA is finished.

The HESA can perfectly support the multi-grained resource reuse. For example, when a company plans to enter a new domain such as Antivirus, the stakeholders want to know a few general features about this domain, e.g. *"Anti-rootkit"*, *"Heuristic scanning"*, and *"File backup"*. Inputting requirements and use the search engine of HESA, the matched features will be returned. Based on the feature in coarse granularity, they can find some mutual software systems. Furthermore, to know this field more clearly and locate some reusable code fragments or packages, some fine granularity

(a) Bullet-point lists of features in *Softpedia.com*

(b) Bullet-point lists of features in *Sourceforge.com*

(c) Release notes in *Freecode.com*

Figure 2. Examples of feature elements in the software pages

features can be retrieved from HESA, e.g., *"Automatic detection of downloaded files and Lock Autorun.inf, virus cannot execute."*

III. Construction of HESA

A. Feature Elements in Software profiles

The online software profiles contains a wealth of information about domain-specific features. In this paper, all the feature elements are extracted from software profiles in Softpedia.com[2], Freecode.com[3] and Sourceforge.com[4]. As depicted in Figure 2(a), there is a bullet-point list of some key features about the software resource in Softpedia.com and Sourceforge.com. Another type of raw descriptions being used is the Release Notes in Freecode.com. A product has many release versions about bug fixes, performance optimizations and feature enhancements. As depicted in Figure 2(b), we extracted feature elements from the release notes about feature enhancement, which contain some related tags or key words, such as "add", "support for" and "new feature".

To allocate different feature elements, which extracted from three different websites, into a unified domain category, the two categories of Softpedia.com and Sourceforge.com was combined into a new one. Then, all software and their feature elements are automatically classified into the unified category according to softwares tags or descriptions using the method of paper [2].

B. Feature Element Analysis

Because different people describe the functions in terms of their personal understanding in an open environment, feature elements are unstructured and disordered. To illustrate these problem clearly, the feature elements in Antivirus category are used as examples in this paper.

[2]http://www.softpedia.com
[3]http://freecode.com
[4]http://sourceforge.net

Hybrid Semantic-level: The problem of hybrid semantic-level is that different feature elements describe a common theme in different semantic level, such as the following descriptions:

(1) *"Email Scanner enhanced email protection"*;

(2) *"Email scanning for Microsoft Outlook, Outlook Express, Mozilla Thunderbird, Windows Live Mail, Windows Mail, and other POP3/IMAP mail clients, ensuring your email is free of viruses and other threats"*;

(3) *"Blocks spam mails, phishing attack mails, junk mails and porn mails before they reach your inbox"*;

The first sentence describes the theme of email protection in a general level. However, the last two sentences present more details including what type of mail clients would be supported and what kind of message would be filtered.

According to sampling statistics of our datasets, there are 25.7% feature elements in a relative high semantic-level, 33.9% feature elements in a relative specific semantic-level, and 40.4% in the intermediate-level.

On one hand, the massive number of feature elements in different semantic-level are good materials for the construction of flexible granularity ontology. On the other hand, it is a great challenge for the traditional methods to cluster and reorganize feature elements.

Synonymic Feature Element: The problem of synonymic feature element happens when two features are used to describe some common or very similar functional attributes. Some feature elements are almost the same with each other, such as the four feature elements below:

(1) *"Kills the core of AdPower and not only symptoms"*;

(2) *"Kills the core of BANCOS.D and not only symptoms"*;

(3) *"Kills the core of Dyfuca and not only symptoms"*;

(4) *"Kills the core of eBot and not only symptoms"*;

The difference between these feature elements is the name of the malicious code, such as "AdPower", "BANCOS.D" and "Dyfuca". However, all of them present a common functional attribute about the ability of killing various popular viruses.

Another typical problem is that each pair only shares few core words, such as the following:

(1) *"Ability to update that does not require downloading full package"*;

(2) *"Incremental database updates and often to include information about latest threats"*;

(3) *"Incremental updating system minimizes the size of regular update files"*;

These three feature elements present the common attribute about incremental updating, but only the word "***update***" is shared by the two sentences. According to the statistics, there are 33.7% of synonymic feature elements in Antivirus category, 28.9% in File-manger category, and 41.6% in Audio-Player category. Thus, feature elements should be merged together by an effective method.

Latent Semantic Structures*:* According to our observation, one feature element may relate to several specific topics. Take five feature elements of *Mozilla Firefox*[5] and three topics of "*Browse with Security*", "*Protect your Privacy*" and "*Easy to Use*" as an example. Figure 3 illustrates the relationship among these feature elements and topics. Each feature element connects with one or two topics and this connection can be exposed by the key words or phrases. For example, the feature element "*Control the level of scrutiny you'd like Firefox to give a site with a variety of customized settings*" is related to the topics of "*Browse with Security*" and "*Protect your Privacy*". The phrases "*Control the level of*" and "*scrutiny*" reflects that it is possible to associate with the topic of security, and "*you'd like*" and "*customized settings*" reflects the relevancy with the topic about user experience. The relationship between topic and feature element is a kind of latent semantic structures which is useful for the clustering of feature element and the construction of feature-ontology.

C. Feature-Topic Model

Problem Formalization*:* In a specific category, such as Antivirus, all the feature elements in the corpus can be represented as $\mathbf{F}_m = \{f_1, f_2, \ldots, f_i, \ldots, f_m\}$, where f_i denotes the ith feature elements in the corpus. Assuming that K latent topics $\mathbf{T}_k = \{t_1, t_2, \ldots, t_j, \ldots, t_k\}$ are implicit in the feature elements, where t_j denotes the jth topic. Although a feature element can be bound up with several topics, it may put more emphasis on some topics than the others. The topic degree within feature element f_i can be represented as a *K*-dimensional vector $\boldsymbol{\upsilon}_i = (p_{i,1}, p_{i,2}, \ldots, p_{i,j}, \ldots, p_{i,k})$, where $p_{i,j}$ is a topic weight describing the extent to which the topic t_j appears in feature element f_i. When $p_{i,j} = 0$, f_i is irrelevant to t_j. Thus, the $\boldsymbol{\upsilon}_i, i \in [1, m]$, represented by $\mathbf{V}_m$, can be used to indicate the semantic structure implied in feature elements. If the $\mathbf{V}_m$ can be obtained, the thematic similarity measure would be induced for each pair of feature elements and the synonymic feature elements would be merged together. Because topic models answer what themes or topics a document relates to and quantify how strong such relations are, it is a effective way to learn $\mathbf{V}_m$.

[5]http://www.mozilla.org/en-US/firefox/security

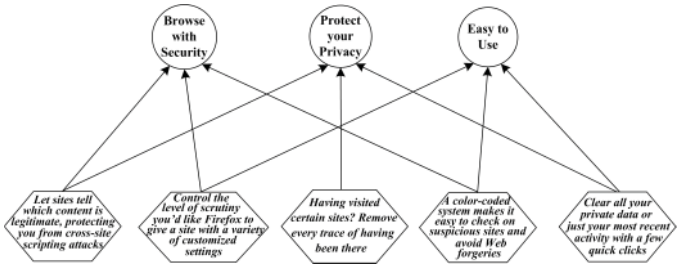


Figure 3. Feature elements mapping to topics

Topic Modeling Technique*:* A topic model provides a means to automatically analysis the semantic structures within unstructured and unlabeled documents. In this paper, we choose Latent Dirichlet Allocation (LDA) [11] because it has been shown to be more effective for a variety of software engineering purposes [12] [13] than other topic models like LSI. In LDA, each word w_i in a document d is generated by sampling a topic z from *document-topic* distribution, and then sampling a word from *topic-word* distribution. More formally, a latent topic $z = j$ is modeled as an unlabeled *topic-word* distribution $\phi^{(j)} = P(w|z = j)$, which was drawn from a dirichlet prior distribution $\mathbf{Dirichlet}(\beta)$. The number of topics *K* is specified beforehand to adjust the granularity. Each document d is a mixture of topics $\theta^{(d)} = P(z)$ with a dirichlet prior distribution $\mathbf{Dirichlet}(\alpha)$. The generative process of each word in d is an essentially draw from the joint distribution: $P(w_i) = \sum_{i=1}^{K} P(w_i|z_i = j)P(z_i = j)$. Given the observed documents, Gibbs Sampling algorithm [14] is widely used for posterior inference. Finally, the *word-topic* $\boldsymbol{\phi}$ and topic-document $\boldsymbol{\theta}$ distribution can be approximated.

However, document is a generalized concept which can be any textual resource. In this paper, a feature element f_i can be viewed as a document which is preprocessed by removing commonly occurring words and then by stemming the remaining words to their root form. According to category, we apply LDA to process the documents using the MALLET tool [15] which is an implementation of the Gibbs sampling algorithm. Then, the *topic-feature* distribution $\mathbf{V}_m$ can be trained, which is the same as $\boldsymbol{\theta}$.

D. iAHC : improved Agglomerative Hierarchical Clustering

To support multi-grained reuse environment, the semantic similar feature elements should be merged and reorganized as a flexible hierarchical structure defined as feature-ontology. In this paper, we present an iAHC algorithm (**Algorithm 1**) integrated with the LDA.

Initially, every feature elements is a distinct cluster. Line 4-7 finds the closest two clusters c_i and c_j in the current cluster set M, and merge them into a new cluster c and update M. The proximity used to measure the distance between every two clusters, defined as below:

$$proximity(c_i, c_j) = \frac{\sum_{fi \in c_i} \sum_{fj \in c_j} similarity(fi, fj)}{|c_i| \times |c_j|} \quad (1)$$

Algorithm 1 improved Agglomerative Hierarchical Clustering

Require:
$\mathbf{F}_m = \{f_1, f_2, \ldots, f_i, \ldots, f_m\}$;
feature-topic distribution $\mathbf{V}_m$;
Ensure:
The construction of feature-ongtology;
1: $M \leftarrow D$
2: $featureSet \leftarrow \emptyset$
3: **repeat**
4: $\langle c_i, c_j \rangle = \mathbf{findTwoClosestClusters}(M)$
5: merge c_i and c_j as c
6: delete c_i and c_j from M
7: add c to M
8: $centroid = \mathbf{calculateCentroid}(c)$
9: **for** $c_i \in c$ **do**
10: $value_s = \mathbf{Similarity}(c_i, centroid)$
11: $degree_t = \mathbf{calculateTopicDegree}(c_i)$
12: $score_m = \kappa \times value_s + \lambda \times degree_t$
13: add $score_m$ to $MedoidScore$
14: **end for**
15: $medoid_C = \mathbf{findMaximumScores}(MedoidScore)$
16: $score_F = \mathbf{Similarity}(medoid_C)$
17: $feature_C = \mathbf{mergeMedoid}(medoid_C, score_F)$
18: $\mathbf{saveFeaturetoHESA}(M, feature_C)$
19: **until** $|M| = 1$

Where $c_i, c_j \subseteq \mathbf{F}_m$, $similarity(f_i, f_j)$ used to calculate the divergence between any two data point. Based on LDA, the divergence can be understood as the thematic space coordinate distance between the two feature elements. There are several ways to calculate the divergence between any two *feature-topic* distributions, such as *Jenson-Shannon* divergence, *cosine similarity* and *KL* divergence. Taking cosine similarity as an example, the Equation is shown as below:

$$similarity(f_i, f_j) = \frac{\boldsymbol{v}_i \cdot \boldsymbol{v}_j}{||\boldsymbol{v}_i||\ ||\boldsymbol{v}_j||} = \frac{\sum_{r=1}^{k} p_{ir} \times p_{jr}}{\sqrt{\sum_{r=1}^{k} p_{ir}^2} \times \sqrt{\sum_{r=1}^{k} p_{jr}^2}} \quad (2)$$

Where k is the topic number and p is the probability value of $\boldsymbol{v}$.

Line 8-14 pick out a set of feature elements from the new cluster, defined as *medoid*, which can be used to represent the theme of c. Two metrics, *similarity value* and *topic degree*, are used to determine the medoid. Firstly, to get the $value_s$, we calculate the similarity between $c_i \in c$ and the *centroid* of c through Equation 2, where the vector $\boldsymbol{v}_{\bar{c}}$ of centroid is calculated by $\boldsymbol{v}_{\bar{c}} = \frac{\sum_{i=1}^{|c|} \boldsymbol{v}_i}{|c|}$. Then, the Equation 3 is used to calculate the $degree_t$ based on the following two important observations of feature-topic distribution V_m in our datasets.

$$degree_t = x_{max} + \frac{1}{e^{\sqrt{\frac{\sum_{r=1}^{\hat{k}} (x_{max} - p_{ir})^2}{\hat{k}}}}} \quad (3)$$

Where x_{max} is the maximum value of $\boldsymbol{v}_i$, and $\hat{k}$ is the frequency when $\boldsymbol{v}_i$ not equal to zero, and p_{ir} is any value that not equals to zero in $\boldsymbol{v}_i$.

Observation 1 The most probable topic reflects the most prominent theme that the document (feature element) is about.

Observation 2 The more widely and evenly distributed its topics are, the higher-level the document (feature element) is.

In brief, Equation 3 can ensure the feature element in the coarsest granularity have the highest score $degree_t \in (0, 2]$, where $x_{max} \in (0, 1]$ can reflect the emphasis topic and the formula $\frac{1}{e^{\sqrt{\sum_{r=1}^{\hat{k}} (x_{max} - w_{ir})^2}}} \in (0, 1]$ can reflect the semantic generality.

The $score_m$ is used to measure the *medoid* calculated as the Equation of line 12, where κ and λ is the empirical coefficients.

Finally, the *medoid* with the highest $score_m$ would be selected. Measuring the similarity for the each pair of elements in $medoid_C$ (line 15), the $feature_C$ (line 17) can be formed by merging distinguished feature elements whose similarity score below a threshold (set to 0.38). Each iteration in the repeat clause saves the M and $feature_C$ to HESA. On the termination of the algorithm, a feature-ontology (Figure 4) for the category is constructed.

E. The Retrieval Method of HESA

Figure 4 depicts an example of the construction process and result with 6 data nodes using the iAHC algorithm. Each cluster consists of several nodes and the top node (the red color one in Figure 4) is the feature of the cluster. The concept *layer* is defined as below:

(1) The layer 0 consists of the bottom nodes which are the original feature elements;

(2) The layer i consists of the feature node of cluster i and all nodes in layer $i-1$ except those being merged in cluster i.

For example, the layer 3 consists of the features of cluster 3, cluster 2 and cluster 1, because all the nodes in different clusters.

The most important advantage of the feature-ontology is that the nodes in a layer are the most representative features under a given similarity threshold. If the stakeholder needs a generalized feature of the category, the feature in the top layer can be selected. Assuming that the category of Figure 4 can be covered by three features, the feature nodes of cluster 3, cluster 2 and cluster 1 would be retrieved step by step. From top down, the semantic granularity is finer and finer accompanying with the increasing number of features, which can satisfy the requirements of multi-grained reuse environments.

Algorithm 2 is a flexible method to retrieve features in terms of quantity, which demonstrates the advantages of the feature-ontology. The input is the quantity of feature you need for a specific domain. Line 1-6 show the process of finding the suitable layer. Then, all the nodes in the same layer are selected out in the repeat clause (line 9-20). An online search engine has been implemented based on it in this paper.

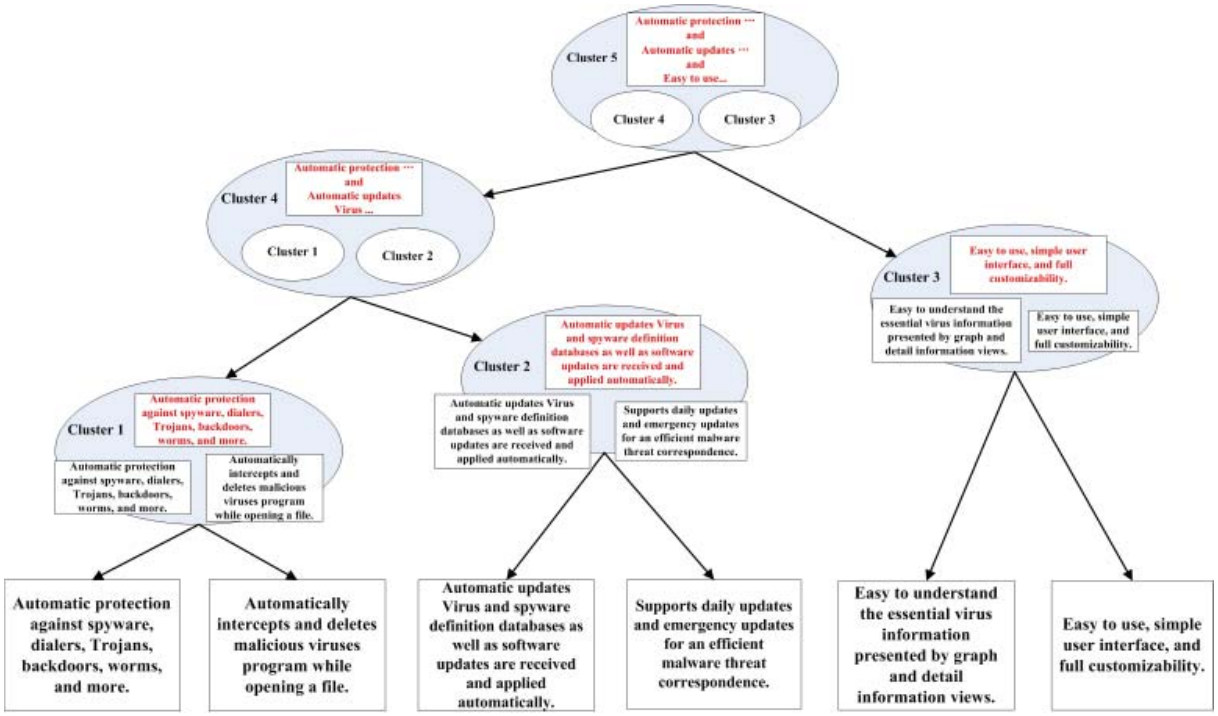


Figure 4. An example of the feature-ontology

IV. Empirical Evaluation

In this section, we present our dataset and experiment setting, research questions and answers, and describe some threats to validity.

A. Dataset and Experimental Setting

Dataset: We have collected 187,711, 432,004 and 45,021 projects profiles from Softpedia.com, Sourceforge.com and Freecode.com respectively. Compared with that of the other two communities, the quantity of projects from Freecode.com is relatively small. Thus, we just adopt projects in Softpedia.com and Sourceforge.com. The feature elements have been classified into 385 categories and we randomly choose the data of 3 unique categories to evaluate our method including Antivirus, Audio-Player and File-manger. Furthermore, the feature elements are preprocessed by removing commonly occurring words and then by stemming the remaining words to their root form. To ensure the quality of data, we omit the preprocessed feature elements with less than 6 words. Table 1 presents the details about our dataset.

Parameter setting: As shown in Table 1, for LDA, the number of topics K was empirically set as different value, and the hyper-parameters α and β were set with $\alpha = 50/K$ and $\beta = 0.01$ respectively, and the iteration of Gibbs Sampling was set as 1000. In addition, the coefficients κ and λ of **Algorithm 1** were set with $\kappa = 0.7$ and $\lambda = 0.3$.

B. Research Questions

To demonstrate the effectiveness of the approach in this paper, we are interested in the following research questions:

Table I
PREPROCESSED EXPERIMENT DATASETS

Category	#Softpedia	#Sourceforge	#Total	#Topic
Antivirus	2919	1105	4024	50
Audio-Player	3714	1283	4997	60
File-Manager	2270	970	3240	40

RQ1 How the resultant feature-ontology looks like?

RQ2 Does the iAHC algorithm achieve better clustering results than the simple but classical method and the state-of-the-art approach?

RQ3 How accurate is the feature-ontology? Is the structure reasonable?

C. Cross-Validation Design of the User Study

The cross-validation limits potential threats to validity such as fatigue, bias towards tasks, and bias due to unrelated factor. We randomly divided the 45 students from computer school of NUDT into three groups to evaluate RQ2 and RQ3. The 2 questions and the 3 categories of dataset can be composed to 6 tasks. Each group randomly picks up 2 of them and finishes in one day, and then we summarize the result.

RQ1: Feature-ontology: Figure 4 shows an example feature-ontology of the Antivirus category which is a very reasonable structure. The features (red color) in different layer can be mapped to resources on different granularities. In addition, the feature is relatively representative for each cluster.

RQ2: Clustering Results: We choose the K-Medoids (tf-idf), a classic and widely used clustering algorithm, and

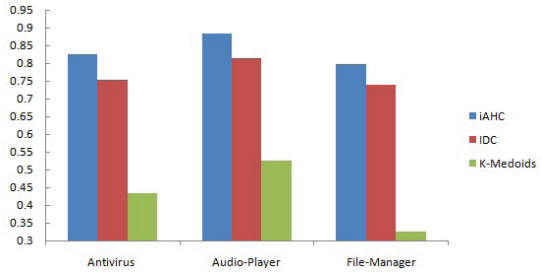
Algorithm 2 a flexible granularity retrieval method

Require:
k_f the quantity of feature you need;
T a hierarchical structure consisting of n nodes;
Ensure:
$featureSet$ a set of features;
1: $layer \leftarrow n - k$
2: **if** $layer = 0$ **then**
3: **return** the nodes of $T[0]$
4: **end if**
5: $i \leftarrow 0$
6: $featureSet[i] \leftarrow$ the node of $T[layer]$
7: $layer \leftarrow layer - 1$
8: $i \leftarrow i + 1$
9: **repeat**
10: **if** $T[layer]$ is a subtype of $featureSet[i]$ **then**
11: $layer \leftarrow layer - 1$
12: **else**
13: $featureSet[i] \leftarrow$ the node of $T[layer]$
14: $i \leftarrow i + 1$
15: $layer \leftarrow layer - 1$
16: **if** $layer = 0$ **then**
17: **return** the rest nodes of T
18: **end if**
19: **end if**
20: **until** $i = k_f$
21: **return** $featureSet$

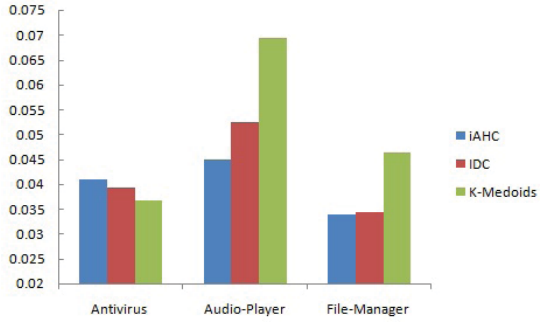
the Incremental Diffusive Clustering (IDC), the state-of-the-art technique proposed in paper [8], as the baseline. Especially, the IDC use the feature descriptions from Softpedia.com which is the same as our dataset. We also use the modified version of Cans metric [8] to compute the ideal number of clusters. Then, we retrieve the corresponding number of clusters from HESA for comparison. Precision is a percent of the reasonable feature elements in a cluster. Figure 5 shows the average value and standard deviation of the judgments given by different groups. We can see that our approach achieves the highest precision in all three categories and relatively low deviations. The precisions and deviations are comparatively stable across different categories, which shows the probability that our approach is more generalizable in different domains. We plan to conduct more experiments to study this issue in the future.

RQ3: Accuracy of the feature-ontology: According to the three categories, participants randomly choose 15 clusters in different layers from HESA using the online search engine respectively. Each participant is randomly assigned 3 layers and asked to provide a 3-point Likert score for each cluster to indicate whether they agree if the feature is the most representative of all terms. Score 3 means very reasonable, Score 2 means reasonable but also have better one, Score 1 means unreasonable.

Table 2 shows that 35.03% features are reasonable, 49.57% partially reasonable and only 15.40% unreasonable. The mean of Likert score is 2.20, which means that the feature selected



(a) the average value



(b) the standard deviation

Figure 5. The clustering results for each category

Table II
EVALUATION OF FEATURE-ONTOLOGY QUALITY

Category	Score-3	Score-2	Score-1	Likert
Antivirus	33.3%	50.0%	16.7%	2.17
Audio-Player	39.1%	46.3%	14.6%	2.25
File-Manager	32.7%	52.4%	14.9%	2.18
Average	35.03%	49.57%	15.4%	2.20

out by our approach is reasonably meaningful.

D. Threats to validity

First, the participants manually judge the clustering results and their ratings could be influenced by fatigue, prior knowledge and the other external factors. These threats were minimized by randomly distributing participants to the various groups and dividing the tasks into multiple parts. Second, due to our limited datasets, parameters used in our approach, the evaluation is not comprehensive enough.

In the feature, with the help of our online search engine, we plan to adopt the idea of crowdsourcing and upload a lot of tasks about use study on the Internet with a low price, such as 5 cent. Therefore, a comprehensive and reliable evaluation result can be obtained.

V. Related Work

Recently, mining software repository has been brought into focus and many outstanding studies have emerged that use this data to support various aspects of software development [16]. However, fewer previous works have been done for mining software feature and especially construction of feature-ontology (defined in this paper), to the best of our knowledge.

In this section, we review some previous works about feature analysis and ontology learning.

In feature analysis area, most approaches involve either the manual or automated extraction of feature related descriptions from software engineering requirements and then use the clustering algorithm to identify associations and common domain entities [7] [17] [18]. Niu et al. [19] propose an on-demand clustering frame-work that provided semi-automatic support for analyzing functional requirements in a product line. Mathieu Acher et al. [9] introduced a semi-automated method for easing the transition from product descriptions expressed in a tabular format to feature models. A decision support platform is proposed in paper [20] to build the feature model by employing natural language processing techniques, external ontology (such as WordNet), and MediaWiki system. However, the quantity of the existing documents is so limited that the brilliance of data mining techniques cannot be fully exploited. To address this limitation, paper [8] and [10] proposed the Incremental Diffusive Clustering to discover features from a large number of software profiles in Softpedia.com. Based on the features, a recommendations system is build by using association rule mining and the k-Nearest-Neighbor machine learning strategy. Compared with these studies, the clustering algorithm presented in this paper is more effective by mining the semantic structures from feature elements and especially focus on the construction of feature-ontology.

Ontology learning (also called ontology extraction) from text aims at extracting ontological concepts and relation from plain text or Web pages. Paper [21] developed an ontology learning framework using hierarchical cluster and association rule for ontology extraction, merging, and management. Several researches have attempted to induce an ontology-like taxonomy from tags. Jie Tang et al. [22] proposed a generative probabilistic model to mine the semantic structure between tags and their annotated documents, and then create an ontology based on it. Xiang Li et al. [23] enhance an agglomerative hierarchical clustering framework by integrating it with a topic model to capture thematic correlations among tags. Based on tens of thousands of software projects and their tags, Shaowei Wang et al. [24] propose a similarity metric to infer semantically related terms, and build a taxonomy that could further describe the relationships among these terms. In this paper, to support multi-grained reuse, emphases of the feature-ontologys construction is on the measure of similarity and granularity instead of generality.

VI. Conclusion and Future Work

The continuing growth of open source ecosystems creates ongoing opportunities for mining reusable knowledge. In this paper, we have explored the idea of mining large scale repositories and constructed the HESA to support software reuse. In the future, we plan to improve the performance of our method and aggregate richer features from software repositories. In addition, we will design several representative applications based on HESA, such as software resource recommendation system, to support the reuse of multi-grained resources.

VII. Acknowledgement

This research is supported by the National High Technology Research and Development Program of China (Grant No. 2012AA011201) and the Postgraduate Innovation Fund of University of Defense Technology (Grant No.S120602).

References

[1] W. B. Frakes and K. Kang, "Software reuse research: Status and future," *IEEE Trans. Softw. Eng.*, vol. 31, no. 7, pp. 529–536, Jul. 2005. [Online]. Available: http://dx.doi.org/10.1109/TSE.2005.85

[2] T. Wang, G. Yin, X. Li, and H. Wang, "Labeled topic detection of open source software from mining mass textual project profiles." in *Software Mining*, 2012, pp. 17–24.

[3] A. E. Hassan and T. Xie, "Mining software engineering data." in *ICSE (2)*, 2010, pp. 503–504.

[4] S. Apel and C. Kastner, "An overview of feature-oriented software development." 2009, pp. 49–84.

[5] K. Lee, K. C. Kang, and J. Lee, "Concepts and guidelines of feature modeling for product line software engineering." in *ICSR*, 2002, pp. 62–77.

[6] 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.

[7] W. B. Frakes, R. P. Dłaz, and C. J. Fox, "Dare: Domain analysis and reuse environment." 1998, pp. 125–141.

[8] H. Dumitru, M. Gibiec, N. Hariri, J. Cleland-Huang, B. Mobasher, C. Castro-Herrera, and M. Mirakhorli, "On-demand feature recommendations derived from mining public product descriptions." in *ICSE*, 2011, pp. 181–190.

[9] M. Acher, A. Cleve, G. Perrouin, P. Heymans, C. Vanbeneden, P. Collet, and P. Lahire, "On extracting feature models from product descriptions." in *VaMoS*, 2012, pp. 45–54.

[10] C. McMillan, N. Hariri, D. Poshyvanyk, J. Cleland-Huang, and B. Mobasher, "Recommending source code for use in rapid software prototypes." in *ICSE*, 2012, pp. 848–858.

[11] D. Blei, A. Ng, and M. Jordan, "Latent Dirichlet Allocation," *Journal of Machine Learning Research*, vol. 3, pp. 993–1022, 2003. [Online]. Available: http://www.cs.princeton.edu/ blei/lda-c/

[12] K. Tian, M. Revelle, and D. Poshyvanyk, "Using latent dirichlet allocation for automatic categorization of software." in *MSR*, 2009, pp. 163–166.

[13] H. U. Asuncion, A. U. Asuncion, and R. N. Taylor, "Software traceability with topic modeling." in *ICSE (1)*, 2010, pp. 95–104.

[14] T. Griffiths, "Gibbs sampling in the generative model of Latent Dirichlet Allocation," Stanford University, Tech. Rep., 2002. [Online]. Available: www-psych.stanford.edu/ gruffydd/cogsci02/lda.ps

[15] A. K. McCallum, "Mallet: A machine learning for language toolkit," 2002, http://mallet.cs.umass.edu.

[16] A. E. Hassan, "The road ahead for mining software repositories." 2008.

[17] S. Park, M. Kim, and V. Sugumaran, "A scenario, goal and feature-oriented domain analysis approach for developing software product lines." 2004, pp. 296–308.

[18] V. Alves, C. Schwanninger, L. Barbosa, A. Rashid, P. Sawyer, P. Rayson, C. Pohl, and A. Rummler, "An exploratory study of information retrieval techniques in domain analysis." in *SPLC*, 2008, pp. 67–76.

[19] N. Niu and S. M. Easterbrook, "On-demand cluster analysis for product line functional requirements." in *SPLC*, 2008, pp. 87–96.

[20] E. Bagheri, F. Ensan, and D. Gasevic, "Decision support for the software product line domain engineering lifecycle." 2012, pp. 335–377.

[21] A. Maedche and S. Staab, "Learning ontologies for the semantic web." in *SemWeb*, 2001.

[22] J. Tang, H. fung Leung, Q. Luo, D. Chen, and J. Gong, "Towards ontology learning from folksonomies." in *IJCAI*, 2009, pp. 2089–2094.

[23] X. Li, H. Wang, G. Yin, T. Wang, C. Yang, Y. Yu, and D. Tang, "Inducing taxonomy from tags: An agglomerative hierarchical clustering framework," in *Advanced Data Mining and Applications*. Springer Berlin Heidelberg, 2012, vol. 7713, pp. 64–77.

[24] S. Wang, D. Lo, and L. Jiang, "Inferring semantically related software terms and their taxonomy by leveraging collaborative tagging." in *ICSM*, 2012, pp. 604–607.

Class Diagram Retrieval with Particle Swarm Optimization

Wesley Klewerton Guez Assunção[1,2], Silvia Regina Vergilio[1]
[1]DINF - Federal University of Paraná (UFPR), CP: 19081, CEP: 81531-980, Curitiba, Brazil
[2]COINF - Technological Federal University of Paraná (UTFPR), CEP: 85902-490, Toledo, Brazil
{wesleyk, silvia}@inf.ufpr.br

Abstract

***Abstract*—An important software reuse task is the retrieval of models, such as UML class diagrams. The space for searching the most adequate model may be huge, and search based algorithms can efficiently solve the problem. Works in the literature generally represent the diagrams as graphs, and are based on Genetic Algorithms (GA). Differently of the existing works, the approach described in this paper uses the Particle Swarm Optimization (PSO) algorithm. The idea is to better explore the search space composed by diagrams from diverse applications with a collective intelligence. We introduce a representation to the problem, and describe how the PSO and GA algorithms were implemented. An experiment was conducted and the results of both algorithms are compared. We observe that PSO presents a better performance in more complex cases.**

***Keywords*—Model reuse; UML class diagram; particle swarm optimization; genetic algorithm**

1 Introduction

Software reuse is a well established strategy that is based on the use of existing artifacts to develop new software. Its main advantages are: (i) increased productivity; (ii) increased quality; and (iii) reduced costs. Any artifact built in the software development can be reused, including software models. Models play an important role in the software engineering and model reuse brings many benefits: allows early reuse when compared to code reuse; eases complexity management and modifications; and reduces faults being propagated in future development phases.

A common reuse problem, addressed by many organizations, is to find and retrieve from a repository the most suitable model for reusing. In most cases there is an initial draft to represent the software being developed. This draft is used as a query to search in a large repository the most related complete diagram to be reused. Many works use methods based on "brute force". But when we are exploring a large repository the search space can be exponential and a huge number of candidate solutions need to be analyzed. The complexity of such analysis can be illustrated with a simple example. Considering we have a class diagram in a repository with one hundred of classes and a query with eight classes, there is a number of 7.50E+15 possibilities of different permutations with eight classes to be explored to find a similar fragment from that repository. In this scenario if each computational comparison between a fragment and the query, to analyze their similarity, spends ten milliseconds, it will be necessary 2.08E+07 hours.

Another limitation found in the model retrieving is the nature of the artifact. The models in general do not have a standard or syntactic verification, and they are very subjective. So, the retrieval process needs to be based on characteristics that allow exploring efficiently the repository, permitting a good way to analyze the similarity between query and candidate artifact, such as the structural and semantic characteristics of the models. There are works that address the problem as a graph matching problem [1]. To capture semantic characteristics, other works are based on ontologies [2, 3, 4].

Those limitations motivated some authors to use search based techniques, particularly Genetic Algorithm (GA) to properly deal with the model retrieval problem. In the work of Salami and Ahmed [5] the retrieving of models is solved as a matching graph problem. UML class diagrams are rep-

resented by graphs, where nodes correspond to classes and edges to the relationships. The fitness function uses similarity measures based on the name of classes and on the graph topology. Differently, in our work we investigated the application of another metaheuristic for the same problem, the PSO (Particle Swarm Optimization) [6]. The main motivation to do this is that GA and PSO implement different optimization strategies. GA is a competitive algorithm and PSO is based on collective and cooperative intelligence. PSO has been widely used in optimization tasks [6], including software engineering ones with promising results. For example, in [7] PSO was applied in the software testing task, and outperformed GA for complex cases.

Considering those facts, this work introduces a PSO approach for the retrieving problem. A discrete implementation of PSO is described, as well as results from comparison with the GA approach. The main contribution is to better explore the search space of different applications, by using global and local memories, and improve the retrieval process. The global memory is used to indicate the systems in the repository that are the most similar to the query. The local memory is used to explore a specific class diagram.

The remainder of this paper is organized as follows: Section 2 presents the representation to the problem and discusses implementation aspects. Section 3 describes experimental setup. In Section 4 we perform analysis of the obtained results. Section 5 reviews some works similar to ours and Section 6 concludes the paper.

2 Search based solution

This section describes the search based solution adopted to the retrieving problem: population representation, cost evaluation function and algorithms implementation.

2.1 Problem Representation

Two elements of the retrieving problem need to be represented: the query and the repository. The query is usually a fragment (draft) of a diagram being developed. The repository is composed by many class diagrams of different systems. The task here is to find the best mapping between the query and one class diagram from the repository. The best mapping is the candidate (solution) to be reused.

The query is represented as an array of strings. Each element of the array corresponds to a class of the query. To illustrate such representation consider Figure 1. The query has four classes: `Customer`, `Order`, `SpecialOrder` and `NormalOrder`. The individual (solution) is also represented by an array containing two parts. In the first part each element of the array is mapped to an element in the query. This part has a size equal to the query size. The second part contains the name of the diagram in the repository where the individual is included. For example, Individual 1 is from the diagram CD01. The population of individuals is represented by a matrix. The names of the classes present in the matrix are sufficient as the class diagram reference and to consider existing relationships.

Query	Customer	Order	SpecialOrder	NormalOrder	
Individual 1	Customer	Order	SchippingInfo	Product	CD01
Individual 2	Customer	Order	Product	OrderDetail	CD01
Individual 3	Product	User	Customer	Administrador	CD01

Figure 1. Population Representation

2.2 Cost evaluation function

Since we have represented the population, we need to evaluate each solution. Different measures can be used to evaluate the similarity between individual and query. Similarly to the related work [5], we use an aggregation of two similarities measures: (i) the relationship similarity (RS) among the classes, and (ii) the name similarity (NS) of the classes in the query and in the possible mapping, given by:

$$S(X) = \alpha * RS(X) + (1 - \alpha) * NS(X) \qquad (1)$$

where α is between 0 and 1 and is used to prioritize each measure. If α has high values near 1, the measure RS will be prioritize in the search, on the other hand if it has low values, the measure NS will be prioritize.

RS and NS are described below. To exemplify the measures we consider the query and Individual 2 represented in Figure 1. The corresponding diagrams are in Figure 2, which presents the relationships in the query and in the mapped fragment.

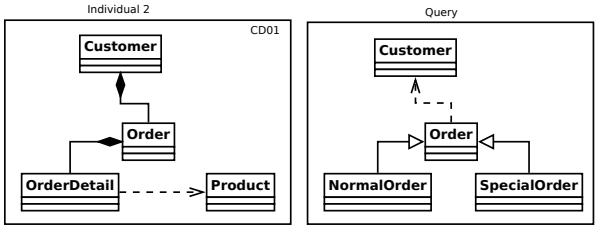


Figure 2. Individual 2 and Query

Relationship Similarity (RS): To calculate RS we use a matrix of Relationship Distances (RD) (Table 1) adapted from [2, 5]. In the table we have the types of relationships: A = Association or Aggregation, D = Dependency, G = Generalization, and R = Realizations. The values in the cells, obtained by applying questionnaires to experts, represent the cost to transform a relationship in another one. For example to transform an Association to a Dependency the effort is 0.45. If the relationships are the same, the effort is 0. If there is not relationship associated, the effort is maximal and equal to 1.

Table 1. RD Matrix (adapted from [2])

From Rel.	To Rel. A	D	G	R	None
A	0	0.45	0.45	0.66	1
D	0.49	0	0.28	0.21	1
G	0.49	0.28	0	0.49	1
R	0.83	0.34	0.62	0	1
None	1	1	1	1	0

Given n the size of a query Q and of an individual X, to determine the relationship similarity RS we use Equation 2. All the relationships of each class in Q are considered, as well as, the relationships of each class in Individual X,

$$RS(X) = \sum_{i=1}^{n} \sum_{j=1}^{n} RD[Rel(X_i, X_j), Rel(Q_i, Q_j)], i \neq j \quad (2)$$

Considering Figure 2, Table 2 presents the RS calculation. We can observe that in four cases the relationship in the query is different from the relationship in Individual 2, so Table 1 was used to measure the cost.

Table 2. RS Calculation Example

Individual Source→Target	From	Query Source→Target	To	RD Cost
Customer→Order	-	Customer→Order	-	0
Customer→Product	-	Customer→SpecialOrder	-	0
Customer→OrderDetail	-	Customer→NormalOrder	-	0
Order→Customer	A	Order→Customer	D	0.45
Order→Product	-	Order→SpecialOrder	-	0
Order→OrderDetail	-	Order→NormalOrder	-	0
Product→Customer	-	SpecialOrder→Customer	-	0
Product→Order	-	SpecialOrder→Order	G	1
Product→OrderDetail	-	SpecialOrder→NormalOrder	-	0
OrderDetail→Customer	-	NormalOrder→Customer	-	0
OrderDetail→Order	A	NormalOrder→Order	G	0.45
OrderDetail→Product	D	NormalOrder→SpecialOrder	-	1
			Total (RS)	**2.90**

Name Similarity (NS): NS is related to how similar are the name of the classes. It is calculated with the Levenshtein Distance (LD) [8]. LD measures the number of characters in a string that need to be changed to obtain another string. We use 0 if the strings are the same, and is 1 if all the characters in a string need to be changed to obtain the other string desired, ie, the strings are totally different. Equation 3 shows how to calculate this measure for individual X. NS is the sum of all the LD values for the classes mapped in an individual. An example is presented in Table 3, again considering Individual 2.

$$NS(X) = \sum_{i=1}^{n} LD(NameX_{(i)}, NameQ_{(i)}) \quad (3)$$

Table 3. NS Calculation Example

Individual Class Name	Query Class Name	Cost
Customer	Customer	0
Order	Order	0
Product	SpecialOrder	0.83333
OrderDetail	NormalOrder	0.81818
	Total(NS)	**1.65152**

2.3 Algorithms

Both algorithms, PSO and GA, were implemented by using versions available in the framework jMetal [9]. In such framework, a continuous implementation of PSO is available, so we adapted it to a discrete version. The main modifications are related to the way that velocity is calculated and to particles movement. They are described next.

PSO works with a population-based heuristic inspired by the social behavior of bird flocking aiming in finding food. It was introduced by Kennedy and Eberhart [10]. In PSO the system initializes with a set of solutions, possibly random, and searches for optima by updating generations. The set of possible solutions is a set of particles, called swarm. Each swarm moves in the search space, in a cooperative search procedure. These moves are performed by an operator that is guided by a local and a social component [11]. This operator is called velocity of a particle and moves it through the search space based on the leader positions (global component), and on their own best position (local component). The best particles are found based on the fitness function, which is the problem objective function.

For continuous problems, each particle p_i, at a time step t, has a position $\overrightarrow{x}(t)$, which represents a possible solution. The position of the particle at time $t+1$ is obtained by adding its velocity, $\overrightarrow{v}(t)$, to $\overrightarrow{x}(t)$:

$$\overrightarrow{x}(t+1) = \overrightarrow{x}(t) + \overrightarrow{v}(t+1) \quad (4)$$

The velocity of a particle p_i is based on the best position already fetched by the particle, $\overrightarrow{p}_{best}(t)$, and the best position already fetched by the all swarm, $\overrightarrow{g}_{best}(t)$, which is the leader. The velocity update function, in time step $t+1$ is defined as follows:

$$\overrightarrow{v}(t+1) = \varpi * \overrightarrow{v}(t) + (c_1 * \phi_1) * (\overrightarrow{p}_{best}(t) - \overrightarrow{x}(t)) + (c_2 * \phi_2) * \overrightarrow{g}_{best}(t) - \overrightarrow{x}(t)) \quad (5)$$

In Equation 5, ϕ_1 and ϕ_2 are coefficients that determine the influence of the particle best position, $\overrightarrow{p}_{best}(t)$, and the particle best global position, $\overrightarrow{g}_{best}(t)$; constants c_1 and c_2 indicate how much each component influences on the velocity. The coefficient ϖ is the particle inertia, and controls how much the previous velocity affects the current one.

In our discrete implementation we use $\overrightarrow{g}_{best}(t)$, the leader, to guide the particles to the class diagram mapped by $\overrightarrow{g}_{best}(t)$. So, considering a great inertial value ϖ when the algorithm starts, and decreasing this value over the interactions, the probability of $1-\varpi$ is used to make a particle to go to the same position of $\overrightarrow{g}_{best}(t)$. The inverse value of the inertia aims to avoid the premature convergence of the particles to a specific class diagram, preventing local optima. $\overrightarrow{p}_{best}(t)$ is used to explore the class diagram mapped by the particle, since it maintains the best mapping of the particle achieved along the search process. The velocity is an array whose elements values are in [0,1]. It has the same size of the vector of strings, and is used as a probability to change the mapped class to another one of the same class diagram. In our discrete implementation, we defined the local and global influence, besides the random values ϕ_1 and ϕ_2, as input parameters represented by P_b and G_b, respectively. The new velocity equation is:

$$\overrightarrow{v}(t+1) = (\varpi * \overrightarrow{v}(t) + (c_1 * \phi_1 * (1 - P_b)) + (c_2 * \phi_2 * (1 - G_b)))/(\varpi + c_1 + c_2) \quad (6)$$

where the division by $(\varpi+c_1+c_2)$ is performed to maintain the velocity in [0,1].

For GA implementation, we adapted the jMetal genetic operators. We used as a base the implemented permutation operators. The crossover strategy used was two-point crossover, where the classes mapped by two individuals are changed between them. To avoid the generation of invalid solutions, the crossover is applied only for parents that map the same class diagram. The mutation follows the bit flip strategy. During the mutation, all the genes of the individual can mutate, so, when the probability reaches the mutation rate the, class in the gene position is changed to another from the same class diagram. The selection operator was the binary tournament with the original jMetal implementation.

3 Experimental Setup

We built a repository of class diagrams from systems in the same domain generated by reverse engineering from source code using the tool called ObjectAid UML Explorer[1] The systems used in the experiment are found in Sourceforge[2], in category "Business", subcategories "Enterprise Financial" and "Point-Of-Sale". To standardize the experiment two filters were applied: (i) "Translations: English"; and (ii) "Programming Language: Java". From the returned results, we selected only those with source code[3].

[1]http://www.objectaid.com/class-diagram

[2]http://sourceforge.net/directory/business-enterprise/financial/pointofsale/

[3]The characteristics of the selected systems, as well as the results obtained by GA and PSO can be found at http://dl.dropbox.com/u/28909248/sekeSite.pdf

The created queries are fragments extracted from a textbook example [12]. It describes a point-of-sale terminal system from the same domain of the systems in the repository. They vary in the number of classes and types of relationships according to Table 4.

Table 4. Queries

Name	Classes	A	D	G	R
Query 1	5	0	1	3	0
Query 2	5	2	0	2	0
Query 3	8	9	0	0	0
Query 4	13	13	0	2	0

The algorithms were configured in order to use the same computational resources. The population size of both algorithms was set with a value twenty times greater than the size of the query. The number of evaluations, which was set one hundred times greater than the population size, was used as stop criterion.

The input parameter of PSO and GA was empirically calibrated using Query 1. For PSO, the inertial weight (ϖ) starts with 1.0 and decreases until 0.4; the constants c_1 and c_2 are both set to 2; the P_b value was set to 0.3 and the G_b value was set to 0.7. For GA, the crossover and mutation rates used are respectively 0.95 and 0.3. Both algorithms were executed thirty times with the same parameters. For the cost evaluation function, the value for α was set to 0.5 to give for the two measures the same importance.

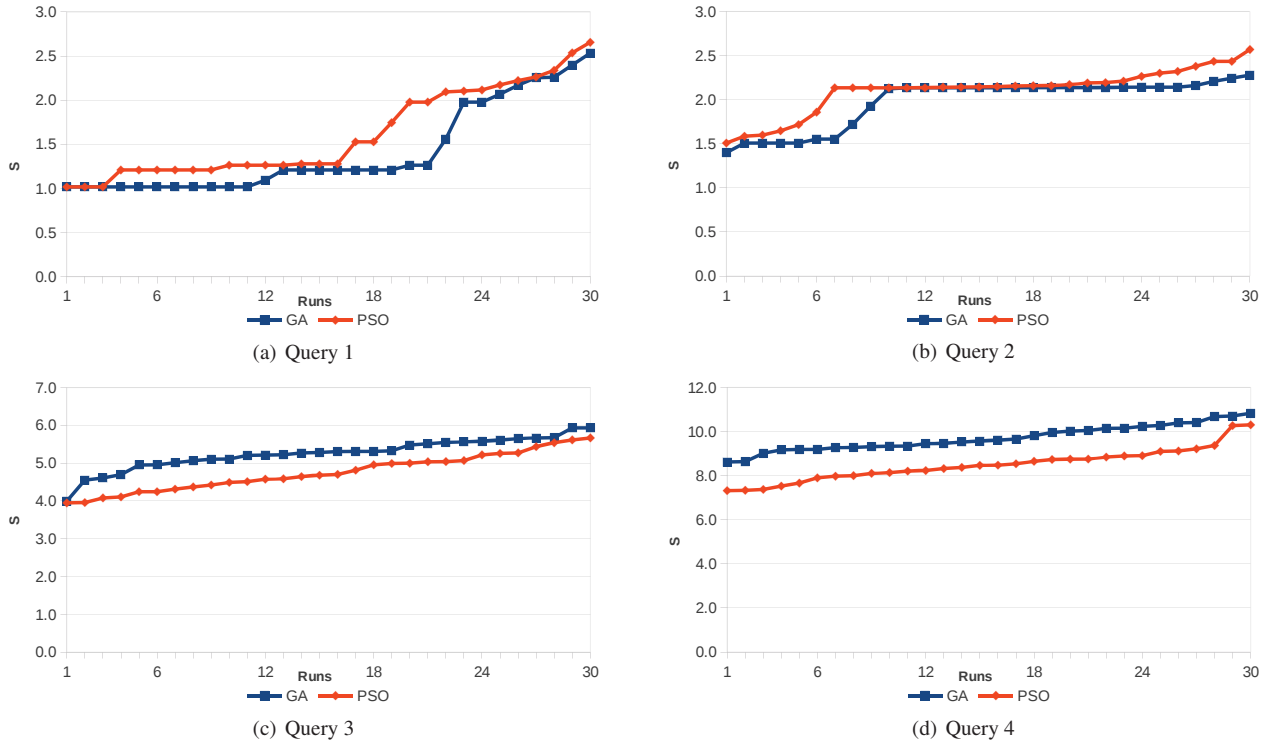
4 Results and Analysis

In this section the results are presented and possible differences between PSO and GA are analyzed. This analysis is based on the best solution found by each algorithm. Each algorithm has thirty solutions obtained, one for each run.

To better show the differences between the algorithms, we depicted in Figure 3 the costs of the best solutions (measure S). In spite of GA be better in Query 1, it is observed that some solutions of PSO have the same costs. In Query 2 we observe the same, with numerous solutions in the same points of the chart. This situation is not observed in charts of Queries 3 and 4. Almost all solutions are found by PSO. We can confirm the complexity of each query, since for Queries 1 and 2, the algorithms achieved solutions with cost near to 1. On the other hand, for the other queries the cost of the achieved solutions are greater.

Taking into account the mean cost of each algorithm we can not observe difference between them. To support our analysis, we use the Wilcoxon Mann-Whitney test [13] to verify statistically differences, considering 95% of confidence (*p-value* lower than the significance level α = 0.05).

The statistical test presents differences between the algorithms for the four queries. For Queries 1 and 2, the simplest



(a) Query 1 (b) Query 2

(c) Query 3 (d) Query 4

Figure 3. Cost of Solutions Found in Each Run

ones, the algorithm GA is the best. For Queries 3 and 4, the most complex ones, the algorithm PSO is the best.

The analysis of the algorithms runtime is performed based on Table 5. For all queries PSO was faster than GA. The percentage that PSO is faster than GA is almost 18%, 9.5%, 46%, and 62.5% for Query 1 to 4, respectively. Considering runtime PSO is better than GA.

Table 5. Runtime (Milliseconds)

Query	GA		PSO	
	Mean	Std. Dev.	Mean	Std. Dev.
Query 1	1444.9	502.97	1181.37	191.73
Query 2	1553.3	239.74	1406.27	333.33
Query 3	12681.87	6176.63	6846.27	4798.61
Query 4	87703.93	22235.68	32965.5	22895.64

5 Related Work

Information and documents retrieval is an active research area, where GAs and other algorithms such as ACO have been used successfully [14, 15, 16, 17]. The works most related to ours address retrieval of UML diagrams. The work of Block et al [18] is dedicated to Use Case (UC) retrieving. The similarity measure is based on the UC's event flows. UCs are also addressed by the work of Morales et al [4], which is based on ontologies and semantic web.

In the work of Ali and Du [19] the retrieval process is based on a conceptual graph and on associated similarity measures. The works of Gomes et al [3, 20, 21, 22] are based on CBR. The diagrams are expressed as cases, and associated to classifiers. The similarity between the classifiers concepts was first calculated by using a taxonomy [21, 22] and later in [3] an ontology. Similarly, the work of Robles et al [2] is also based on ontology. The distances in their work are calculated with a shortest path algorithm.

Kolovos et al. addressed the problem as a graph matching problem [23] considering different measures, including measures based on similarity, similar to ours. However, the work uses exact algorithms that need users configuration.

Another work that is also based on graph matching is the one reported in [5]. The UML diagrams are represented by graphs, where nodes represent the classes and edges the relationships between them. In this way, the retrieving problem corresponds to a matching graph problem, to be solved by a GA. This work is the most similar to ours.

We observe that in the works addressing class diagram retrieving, the structure of the diagrams are usually represented by graphs, and the semantic aspects by taxonomies or ontologies. The matching graph problem and GA based solutions present promising results. However, PSO has not been yet used in this context.

6 Concluding Remarks

This work implemented a discrete PSO to deal with the problem of model based retrieval, with focus on UML class diagrams. The benefits of the global and local information provided by PSO collective intelligence were used to better explore the search space (repository). The social memory works to guide the particles to the best class diagram mapped in the repository, and at the same time the local particle memory is used to explore such diagram.

Results from the study comparing the proposed implementation with a GA show that PSO has a better performance in more complex cases, that is, with more complex queries. Concerning runtime, PSO is always the fast one.

Threats to validity of our study are related to the repository size, since we used a limited number of systems. Another point is the proposed discretization of the algorithm PSO. Other discretization approaches existing in the literature should be evaluated and can get different results. The input parameters of the algorithms are also a threat to validity, despite of we use an empirical calibration, different parameters could impact on the results.

As future work we intend to apply other collective intelligence based algorithms, such as Ant Colony Optimization (ACO), Bee Colony Optimization, and so on. We should also investigate the use of other similarity measures to better evaluate the solutions with respect to the software development environment and software engineer requirements.

References

[1] W. N. Robinson and H. G. Woo, "Finding reusable UML sequence diagrams automatically," *IEEE Software*, vol. 21, pp. 60–67, 2004.

[2] K. Robles, A. Fraga, J. Morato, and J. Llorens, "Towards an ontology-based retrieval of UML Class Diagrams," *Information and Software Technology*, vol. 54, pp. 72–86, 2012.

[3] P. Gomes and A. Leitão, "A tool for management and reuse of software design knowledge," in *Conference on Managing Knowledge in a World of Networks*, 2006, pp. 381–388.

[4] B. Bonilla-Morales, S. Crespo, and C. Clunie, "Reuse of use cases diagrams: An approach based on ontologies and semantic web technologies," *International Journal of Computer Science*, vol. 9, no. 2, 2012.

[5] H. O. Salami and M. A. Ahmed, "A framework for class diagram retrieval using genetic algorithm," in *Conference on Software Engineering and Knowledge Engineering (SEKE)*, 2012.

[6] M. Clerc, *Particle Swarm Optimization*. John Wiley & Sons, 2010.

[7] A. Windisch, S. Wappler, and J. Wegener, "Applying particle swarm optimization to software testing," in *Genetic and Evolutionary Computation Conferences (GECCO)*, 2007, pp. 1121–1128.

[8] V. Levenshtein, "Binary Codes Capable of Correcting Deletions, Insertions and Reversals," *Soviet Physics Doklady*, vol. 10, pp. 707–710, 1966.

[9] J. J. Durillo and A. J. Nebro, "jMetal: A Java framework for multi-objective optimization," *Advances in Engineering Software*, vol. 42, pp. 760–771, 2011.

[10] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *IEEE International Conference on Neural Networks*. IEEE Press, 1995, pp. 1942–1948.

[11] ——, *Swarm intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001.

[12] C. Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*, 3rd ed. Prentice Hall, 2004.

[13] F. Wilcoxon, "Individual Comparisons by Ranking Methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945.

[14] J.-T. Horng and C.-C. Yeh, "Applying genetic algorithms to query optimization in document retrieval," *Information Processing and Management*, vol. 36, pp. 737–759, 2000.

[15] R. K. Bhatia, M. Dave, and R. C. Joshi, "Retrieval of most relevant reusable Component using genetic algorithms," in *Software Engineering Research and Practice*, 2006, pp. 151–155.

[16] ——, "Ant colony based rule generation for reusable software component retrieval," *ACM SIGSOFT Software Engineering Notes*, vol. 35, no. 2, pp. 1–5, 2010.

[17] N. Kaur and J. S. Budwal, "Hybrid Approach to Retrieval of Reusable Component from a Repository Using Genetic Algorithms and Ant Colony," in *International Conference on Genetic and Evolutionary Methods (GEM)*, 2008, pp. 2–7.

[18] M. Blok and J. L. Cybulski, "Reusing UML specifications in a constrained application domain," in *Asia Pacific Software Engineering Conference*, 1998, pp. 196–202.

[19] F. Ali and W. Du, "Toward reuse of object-oriented software design models," *Information and Software Technology*, vol. 46, pp. 499–517, 2004.

[20] P. Gomes, P. Gandola, and J. Cordeiro, "Helping software engineers reusing UML class diagrams," in *International Conference on Case-Based Reasoning (ICCBR)*, 2007, pp. 449 – 462.

[21] P. Gomes, F. Pereira, P. Paiva, N. Seco, P. Carreiro, J. Ferreira, and C. Bento, "Case retrieval of software designs using WordNet," in *European Conference on Artificial Intelligence (ECAI)*, 2002.

[22] P. Gomes, F. C. Pereira, P. Paiva, N. Seco, P. Carreiro, J. L. Ferreira, and C. Bento, "Using wordnet for case-based retrieval of UML models," *AI Communications*, vol. 17, no. 1, pp. 13–23, Jan. 2004.

[23] D. S. Kolovos, D. Di Ruscio, A. Pierantonio, and R. F. Paige, "Different models for model matching: An analysis of approaches to support model differencing," in *ICSE Workshop on Comparison and Versioning of Software Models (CVSM)*, 2009, pp. 1–6.

Towards a strategy for analysing benefits of Software Process Improvement programs

Cristiane Soares Ramos, Ana Regina Rocha
COPPE/UFRJ
Federal University of Brazil
Rio de Janeiro, RJ, Brasil
cristianesramos@unb.br, darocha@cos.ufrj.br

Káthia Marçal de Oliveira
LAMIH, UMR CNRS 8201
University of Valenciennes
Valenciennes, France
kathia.oliveira@univ-valenciennes.fr

***Abstract* — Software processes improvement (SPI) program have been used by several organizations that aim to have better competitive advantage. Since it involves high costs (time and money), it is essential to have some estimate about the return on investment when applying SPI programs. After a large study of the literature, by performing a systematic mapping, we concluded that despite of the existence of several initiatives, the estimate of return on investment for SPI is not very applied in practice, and it is still an open and challenging issue. In general, the existing approaches propose to perform the estimation using a monetary unit, which makes the process difficult to be applied. Based on the results of this study, we propose a light approach to estimate the return on investment for SPI, that considers not only economics aspects but also any kind of benefits obtained by the organization aligned to their business strategy. In this paper, we present the systematic mapping, our conclusions and this approach that we called a strategy for analyzing benefits of SPI programs, defined based on the results of the systematic mapping.**

Keywords - return on investment; software process improvement.

I. INTRODUCTION

Software organizations have increasingly invested in the improvement of their software processes aiming at obtaining software product quality, reducing production costs and enhancing customer satisfaction. In this context, the lack of evidence about the return on investment in software process improvement (SPI) is a critical factor that weakens the commitment of senior management with improvement programs [1]. Galinac [2] points out that small and median enterprises believe that SPI programs require a large investment, but do not always guarantee good results. A survey conducted by Khurshid et al [3] showed that the lack of clarity about the benefits achieved from the implementation of the Capability Maturity Model Integration (CMMI) SPI model is a major reason that impedes the organizations to implement it. In fact, although several approaches have been proposed to support the return on investment (ROI) of SPI, they are not really used in practice in the organizations.

Considering this scenery we performed a rigorous analysis of the literature using systematic mapping procedures [4] to identify what is been considered in the definition and application of ROI for SPI programs. In this study we found that the authors report return not necessarily related to the economic value of the investment. We can suppose, therefore, that other tangible and intangible benefits are also valued by organizations. Based on this finding and the results of this study we define a strategy for the analysis of return on investment about SPI program, but in a larger viewer than economic aspects, therefore we prefer call this approach of an strategy for the analysis of benefits of SPI. This strategy is aligned to the organizations expectations and business objectives in a practical and integrated approach to software processes.

The international maturity model CMMI-DEV [5] and the Brazilian one, MR-MPS [6], are usually used by the organizations to institutionalize SPI programs. Typically, organizations hire specialized consulting and/or sets a quality team dedicated to SPI activities. The strategy proposed in this paper is a tool to support these professionals.

This paper presents the procedures and results of the systematic mapping performed as well as the proposed strategy derived from this study.

In next sections, we begin (section II) by presenting a brief review of the literature on return on investment in SPI. Then, we describe our systematic mapping on the subject (section III) and the resulting proposal from this study (section IV). Finally, in Section V we present our conclusions and ongoing works.

II. RETURN ON INVESTMENT IN SPI

Despite the fact that SPI is widely explored in the literature and applied in industry, the analysis of benefits obtained with SPI still needs further studies.

Several approaches for the analysis of return on investment have been proposed; we can quote for example: ROI in SPI [7], that quantifies and analyses the economic impact of a SPI program; and ROI of software quality [8], that considers ROI as the cost saving and schedule benefits. The main feature of those approaches is the focus on economic aspects.

SPI programs should be guided by the alignment between the investment in process improvement and improvement in business measurement [9]. In this context, several authors [9, 10] defend that SPI must be aligned to business goals of the organization.

The ROI of an improvement program can also be measured by the benefits achieved by the organization. Studies show that various benefits may be obtained from SPI programs [11-15], such as: productivity; customer satisfaction and cost reduction. Therefore, it is necessary to investigate and characterize how these benefits are identified and measured, what tools and methods of engineering economics are used and what is considered return on investment in SPI.

III. A SYSTEMATIC MAPPING ABOUT ROI FOR SPI

Systematic mapping is a kind of systematic review [4], which provides a structure in which the studies are classified by the type of contribution of the research and the results are displayed within this framework. To perform our systematic mapping, we followed the following phases: planning the review, conducting the review and reporting the review.

The main result of the *planning the review* phase is the definition of a review protocol to reduce the possibility of researcher bias. This review protocol specifies: the research question(s) being addressed, the methods that will be used to perform the review, the search strategy with the keyword strings and sources used, and the explicit inclusion and exclusion criteria to assess each potential study. Our review protocol is presented in Fig. 1.

In the *conducting the review* phase, the review is performed based on the defined protocol for the selection of studies. We applied the search string in all digital libraries in November 2012. The search was done in titles, abstracts and keyword. We got at the end a total of 396 papers. After eliminating the 58 duplicated papers, we include the information of the selected 338 papers in EndNote tool. Then, by reading the abstracts of all papers and applying the three first inclusion criteria, we selected 112 papers for the full text analysis. From this group, 13 were eliminated by the criteria 5. After the full text reading, 28 papers were selected considering the criteria 2, 3 and 4.

After performing the paper analysis, we extracted the data answering each one of the secondary questions to be used in the *reporting the review* phase.

Answering the first secondary question (Q1), we found 8 methods (Table I). We note that ROI is the most cited method. However, the results of ROI should be analyzed in conjunction with other methods, such as NPV that considers the ROI based on the risks and necessary spent time for the investment [16, 17]. Some authors [18] claim that NPV should be the most important method for the analysis of ROI because it captures the financial pressure for the small and media enterprises.

TABLE I. METHODS COLLECTED FOR Q1

METHODS	REFERENCES
Return on Investment (ROI)	[14], [15], [18], [19], [20], [12], [21], [22], , [23], [24], [25], [26]
Net Present Value (NPV)	[18], [16], [17], [15], [24], [25], [26]
Ratio of benefits to costs (B/CR)	[15], [21], [27], [26]
Benefit	[15], [24], [26]
Cost	[15], [24],[26]
Break Even Point (BEP)	[15], [26], [24]
Profitability Index	[16]
Internal Rate of Return (IRR)	[16]

Quality focus: Identify the elements that are used to analyze the return on investment in SPI programs.

Main research question: "What are the elements used to analyze the return on investment in software process improvement software?".

Secondary research questions:

- Q1. What methods, processes and tools are used to analyze the return on investment in SPI programs?
- Q2. What measures and other elements are used for analysis of return on investment in SPI programs?
- Q3. What return on investment in SPI programs that organizations have reported?

Population: Published papers about ROI in SPI

Sources selection: SCOPUS, ACM and IEEE

Keywords and synonyms:
Software process
Return on investment= ROI, investment analysis, money, profit, engineering economics, investment.
Improvement = SPI
CMMI = CMM, CMMI-DEV[1]
MR-MPS = MPS, MPS.Br
Maturity level = Capability level

Search string:
("return on investment" OR roi OR "investment analysis" OR money OR profit OR "engineering economics" OR investment) AND ((("software process" AND improvement) OR SPI) OR (cmm OR "CMMI-DEV" OR CMMI OR "MR-MPS" OR MPS OR "MPS.Br" OR "maturity level" OR "capability level"))

Inclusion (IC) and exclusion (EC) criteria:
1. IC - Papers should be written in English;
2. IC – Papers that present approaches about analysis of ROI or benefits in SPI even that are not based on maturity models.
3. IC – Papers that present some kind of study about methods, tools, process, measures or any other item applied for the analysis of ROI or benefits in SPI;
4. EC – Papers that present methods, tools, process, measures that are not related with SPI.
5. EC - Papers not available on the internet.

Intervention: studies involving return on investment in SPI.

Data extraction: methods, tools, process, measures or any other item used for the analysis of ROI or benefits in SPI;

Figure. 1. Research Protocol

[1] We included as keywords the main international and national (Brazilian) models for SPI: the Capability Maturity Model Integration (CMMI) from Software Engineering Institute (and all their synonyms) and the Brazilian SPI model (MPS.Br and its synonyms)

For the second secondary question (Q2), we collected any element related to some benefit (tangible or intangible) of SPI programs. We classified those elements in two categories: the benefits related to measures and general elements.

Table II presents some examples of benefits related to measures. Those measures are used during the whole software process lifecycle, therefore, can be useful for different stakeholders (managers, software developers, testes, client, etc.). Most of the elements classified in the category "general elements", are related to intangible benefits. Table III presents some examples of intangible benefits.

The extracted data for the third secondary question (Q3) indicates the benefits for which results were presented (table IV), that means, the benefits were not only mentioned in the articles, but the papers also discuss the extent to which these benefits were achieved in the organizations. The results of this question showed that to know the value of the return on investment that organizations had programs in SPI is not relevant because it is not reliable to compare results without context information, for example, without knowing which elements were considered to calculate the cost of the program. However, the lack of information on these results shows the existence of a gap as the collection of measurements and analysis results.

TABLE II. EXAMPLES OF BENEFITS COLLETTED FOR Q2

Benefits	References
Productivity	[14], [18], [19], [12], [21], [22], [15], [23], [28], [27], [29], [30], [25], [11]
Product quality	[14], [31], [19], [20], [12], [21], [32], [15], [23], [1], [28], [29], [30]
Customer satisfaction	[13], [18], [31], [20], [12], [21], [32], [33], [23], [28], [11]
On time Deliverables / Schedule fidelity	[14], [19], [21], [22], [34], [15], [23], [27], [29], [25]
Cost Performance improvement	[14], [20], [12], [21], [34], [28], [29], [30], [11]
Early detection of errors /defects	[14], [20], [21], [22], [15], [30], [30], [11]
Cycle time	[12], [21], [32], [23], [28]
Rework	[20], [21], [22], [1]
Consultant fee	[12], [32], [25]
Training cost	[12], [23], [15]
Appraisal cost	[18], [12]
Cost of software process improvement	[23, 25]
Software Process Improvement Effort	[18], [25]

TABLE III. EXAMPLES OF INTANGIBLE BENEFITS

Intangible Benefits	References
More stable work environment	[34]
Improves employee morale	[26]
Software engineering role and responsibility definitions	[25]
Software team's quality awareness increase	[25]
Quality of work life	[34]
Standardized processes	[31]

TABLE IV. BENEFITS COLLETED FOR Q3

Benefits	References
Return on investment	[13], [14], [19], [20], [12], [23], [28]
Cost performance	[20], [12], [34], [30]
Productivity	[12], [30], [25]
Product quality	[12], [23], [1], [30]
Cycle time	[12], [23]
Rework	[1], [20], [21], [23]
Customer satisfaction	[20], [12]

It is known that in the context of economic engineering analysis, ROI refers to the conversion of benefits into a monetary value. However, the results of this systematic mapping showed that in software organizations the return on investment in SPI programs is also characterized by other tangible and intangible benefits that are of interest to the organization (see tables II and III).

Finally, in a systematic mapping it is recommended to categorize the studies using research facets [4] to better quantify the results. We used the facets suggested by [4]:

- Validation Research (VR): The techniques investigated are novel and have not yet been implemented in practice but only in controlled experiments;
- Evaluation Research (ER): Techniques are implemented in practice;
- Solution Proposal (SP): A solution for a problem is proposed and it is shown by a small example or some argumentation;
- Philosophical Papers (PP): These papers sketch a new way of looking at existing things by structuring the field in a form of taxonomy or conceptual framework;
- Opinion Papers (OP): These papers express the personal opinion of somebody;
- Experience Paper (EP): Experience papers explain on what and how something has been done in practice.

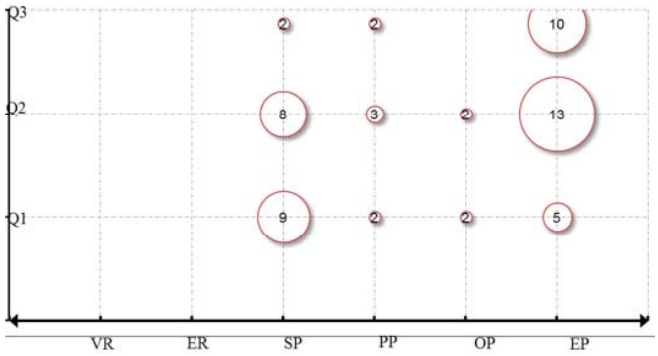


Figure. 2. Papers per Questions and research facets

We classified all papers in each one of this facets and we cross them with the research questions. The bubble-plot graph presented in Fig. 2 shows this analysis, where we highlighted that:

- No proposition was used in practice or validated with some kind of experimentation;
- From papers that present methods (9 papers - 32%) are solution proposal with some kind of application, 5 (17%) presents description of some application in practice.
- Almost half of the studies (46%) that answer Q2 (the use of measures) are experience paper. However, although they report benefits in SPI programs, the authors do not discuss how the initial expectations of the stakeholders were treated, and how the benefits achieved reached those expectations.
- Only 35% (10) of papers show some result of the ROI with some application and 2 (7%) presents some proposition in this subject. These two papers present a methodology to compute the cost saving and the ROI of a SPI program [23], and the results of the use of a framework [27] to the implementation of SPI in small and media enterprises.

Based on the results of this systematic mapping, we identified the need of defining a strategy lighter for the definition of return on investment in SPI that do not use only economic aspects, but that shows the benefits that an organization can reach with those programs aligned to their previous expectations.

IV. A LIGHT STRATEGY FOR THE ANALYSIS OF BENEFITS OF SPI PROGRAMS

A SPI program should be aligned to the business goals of the organization. However, it is very common, that organizations start SPI programs without first defining what are their expectations of benefit and quantifying the expected values for these benefits. Moreover, often, organizations do not assess later whether expectations were actually met.

The strategy proposed in this section should support consultants and process improvement teams working in organizations that will use the MR-MPS [6] or the CMMI-DEV [5] to implement the SPI program. Fig. 3 shows the general view of this strategy that is composed of four main phases. In the first phase of the strategy the consultants/SPI team should (A) *identify expected benefits* for the organization. After that, it should be analyzed if the expectation is aligned to the business goals; otherwise, it is not possible to go ahead to the next phase. It is also analyzed if the expected benefits are attainable, that means it is possible to reach the benefit with the defined maturity level (form CMMI-DEV or MR-MPS) that the organization intends to implement. By assuring these two aspects, the strategy can be continued in a sequence of the three other phases: (B) *identify set of measures*, (C) *define and institutionalize the software process* and (D) *monitor and assess the expected benefits*. The following sub-sections describe briefly each one of these phases.

A. Identify expected benefits

In this phase the organization's expectation for the institutionalization of a SPI program is identified considering potential benefits of a SPI program. Different stakeholders (technical team, managers, etc.) should participate in this activity since the expectation of benefits is a key element in this strategy. Thus, an analysis should be performed to align expectations of benefits to the organization's business goals, and to verify the feasibility of achieving those benefits considering the expectation of the benefits and the software process practices that will be implemented in the organization.

To support this activity, we are defining a set of potential benefits of SPI programs based on the results of the systematic mapping previously presented (see tables II and III). We plan also to perform a survey with organizations in Brazil that implemented SPI program to identify what benefits they obtained. To that end, we will use and adaptation of SERVQUAL [35].

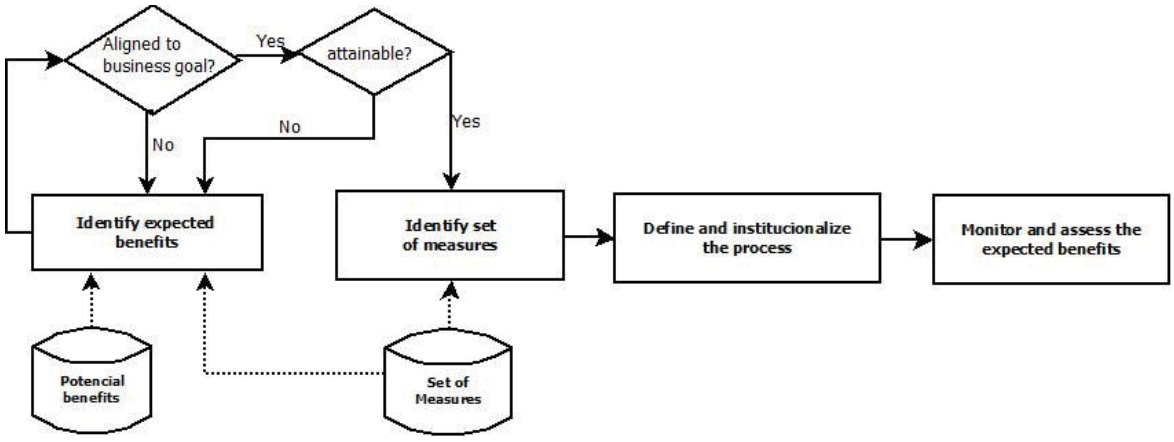


Figure. 3 - Strategy for the analysis of benefits in SPI

These potential benefits are associated with practices (from CMMI-DEV) or expected outcomes (from MR-MPS). In this way, it is possible to identify what should be incorporated into the standard software process of the organization to enhance the achievement of expected benefits. This mapping between benefits and practices/outcomes allows also evaluating if the expectation of benefits identified is feasible considering the maturity level it will be implemented in the organization.

Fig. 4 shows an example of a potential benefit generated from the data collected in the systematic mapping and its mapping with practices of CMMI-DEV.

B. Identify set of measures

This phase aims at supporting the identification of measures that should be collected to monitor the benefits obtained by the SPI program. The measures must be incorporated into the organization measurement plan that is required since the first maturity levels of the SPI models. These measures must be consistent with the expectations of benefits established in the previous activity.

To support this activity, we are mapping a set of measures that can be used to support the measurement of the potential benefits and costs of the improvement program. We are using a previous work [36] where we defined a set of indicators with their software process measures based on a detailed literature review that considers more the 500 different measures for software process evaluation. Table V presents some examples of indicators with the corresponding measures and the potential benefit that they impact. The complete description of the indicators and measures can be found in [36].

Example 1

Some collected elements from the systematic mapping:

- defect density
- quality improvement of the products/artifacts
- defects found during formal testing

Potential benefit defined: Quality of the software product

CMMI Practices (PA Validation - examples):

- SP 1.2 Establish the Validation Environment;
- SP 1.3 Establish Validation Procedures and Criteria
- SP 2.1 Perform Validation

Example 2

Some collected elements from the systematic mapping:

- greater visibility, though not yet complete, into projects and their progress

Potential benefit defined: Visibility of project progress

CMMI Practices (PA Project Monitoring and Control - examples):

- SP 1.1 Monitor Project Planning Parameters
- SP 1.2 Monitor Commitments
- SP 1.3 Monitor Project Risks
- SP 1.4 Monitor Data Management
- SP 1.5 Monitor Stakeholder Involvement
- SP 1.6 Conduct Progress Reviews
- SP 1.7 Conduct Milestone Reviews

Figure. 4. Definition of Potential Benefits

TABLE V. EXAMPLES OF MEASURES AND BENEFITS

Indicator	Measure	Potential benefit
Defect density	Total defect density Defect density by type	Quality of the software product
Test coverage	Test coverage	Quality of the software product
Earned value	Cost performance index Schedule performance index Cost variance Estimate at completion Estimate to complete Variance at completion	Cost performance index
Cost of quality	Cost of quality Percentage cost of quality	Rework cost

C. Define and institucionalize the process

This phase corresponds to the traditional activity of defining the standard software process for the organization as required in the maturity models (CMMI-DEV and MPS.BR). The outcome of this phase is the complete institutionalization of the standard software process that means its real use in the software projects development.

It is important to highlight here that the measures and indicators defined in the previous phases are correlated with process areas (from CMMI-DEV) as defined in [36] and with the process (from MR-MPS). In this way, by defining the standard software process, we know which measures we can use to evaluate the benefits.

D. Monitor and assess the expected benefits

In this phase, the expected benefits are continually evaluated based on the measures defined in the phase 2. Reports of indicators and measures supports the analysis of benefits being obtained with the SPI programs.

V. CONCLUSION

This paper presented the results of a systematic mapping analysis on ROI for SPI. After analyzing the main findings, we concluded that it is necessary to structure a strategy that facilitates the monitoring and analysis of benefits in SPI programs, that considers tangible and intangible benefits that may be of interest to various stakeholders in the improvement program. The main contribution of the strategy is: (i) defining the expectations of benefits, (ii) alignment of expectations to the business objectives of the organization, and (iii) monitoring the results of the improvement program based on the expected benefits.

We are now working on the definition of all information necessary to support the proposed strategy. A real study case in a Brazilian company is also planned.

REFERENCES

[1] D. Peixoto, V. A. Bastista, R. Resende, and C. Pádua, "A case study of Software Process Improvement implementation," Redwood City, CA, 2010, pp. 716-721.

[2] T. Galinac, "Empirical evaluation of selected best practices in implementation of software process improvement," *Information and Software Technology,* vol. 51, pp. 1351-1364, 2009.

[3] N. Khurshid, P. L. Bannerman, and M. Staples, "Overcoming the first hurdle: Why organizations do not adopt CMMI," vol. 5543 LNCS, ed. Vancouver, BC, 2009, pp. 38-49.

[4] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic mapping studies in software engineering," *EASE '08: Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering, University of Bari, Italy,* 2008 2008.

[5] SEI, *CMMI for Development, Version 1.3*: Software Engineering Process Management Program 2010.

[6] SOFTEX, *MPS.BR Melhoria de Processo do Software Brasileiro*: Associação para Promoção da Excelência do Software Brasileiro, 2011.

[7] D. Rico, *ROI of Software Process Improvement: Metrics for Project Managers and Software Engineers*: J. Ross Publishing, Inc, 2004.

[8] K. Emam, *The ROI from Software Quality*, 1a ed.: Auerbach Publications, 2005.

[9] T. Birkholzer, C. Dickmann, and J. Vaupel, "A Framework for Systematic Evaluation of Process Improvement Priorities," in *Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on*, 2011, pp. 294-301.

[10] J. G. Guzmán, H. A. Mitre, A. Amescua, and M. Velasco, "Integration of strategic management, process improvement and quantitative measurement for managing the competitiveness of software engineering organizations," *Software Quality Journal,* vol. 18, pp. 341-359, 2010.

[11] M. Unterkalmsteiner, T. Gorschek, A. Islam, C. Cheng, R.Permadi, and R. Feldt, "Evaluation and measurement of software process improvement-A systematic literature review," *IEEE Transactions on Software Engineering,* vol. 38, pp. 398-424, 2012.

[12] A. Ferreira, G. Santos, R. Cerqueira, M. Montoni, A. Barreto, A. Rocha, A. Barreto, and R. Silva Filho, "ROI of software process improvement at BL informática: SPIdex is really worth it," *Software Process Improvement and Practice,* vol. 13, pp. 311-318, 2008.

[13] R. Asato, S. M. Mesquita, I. Costa, and S. Farias, "Alignment between the business strategy and the software processes improvement: A roadmap for the implementation," in *Management of Engineering & Technology, 2009. PICMET 2009. Portland International Conference on*, Portland, OR, 2009, pp. 1066-1071.

[14] M. Campo, "Why CMMI maturity level 5?," *CrossTalk,* vol. 25, pp. 15-18, 2012.

[15] L. Lazić and N. Mastorakis, "Software economics: Quality-based return-on-investment model," Iasi, 2010, pp. 25-39.

[16] W. Harrison, D. Raffo, J. Settle, and N. Eickelmann, "Technology Review: Adapting Financial Measures: Making a Business Case for Software Process Improvement," *Software Quality Journal,* vol. 8, pp. 211-231, 1999.

[17] W. Harrison, "Using the economic value of the firm as a basis for assessing the value of process improvements," in *Software Engineering Workshop, 2001. Proceedings. 26th Annual NASA Goddard*, 2001, pp. 123-127.

[18] P. Colla and J. M. Montagna, "Framework to evaluate software process improvement in small organizations," vol. 5007 LNCS, ed. Leipzig, 2008, pp. 36-50.

[19] M. Diaz and J. Sligo, "How software process improvement helped Motorola," *Software, IEEE,* vol. 14, pp. 75-81, 1997.

[20] A. Ferreira, G. Santos, R. Cerqueira, M. Montoni, A. Barreto, A. Soares Barreto, and A. R. Rocha, "Applying ISO 9001:2000, MPS.BR and CMMI to achieve software process maturity: BL informatica's pathway," Minneapolis, MN, 2007, pp. 642-651.

[21] D. Galin and M. Avrahami, "Do SQA programs work - CMM works. A meta analysis," Herzlia, 2005, pp. 95-100.

[22] D. Galin and M. Avrahami, "Are CMM program investments beneficial? Analyzing past studies," *IEEE Software,* vol. 23, pp. 81-87, 2006.

[23] T. McGibbon and D. Nicholls, "Making the (Business) case for software reliability," Seattle, WA, 2002, pp. 285-292.

[24] R. Solingen and D. Rico, "Calculating Software Process Improvement's Return on Investment," vol. 66, M. V. Zelkowitz, Ed., ed, 2006, pp. 1-41.

[25] R. Solingen, "Measuring the ROI of software process improvement," *IEEE Software,* vol. 21, pp. 32-38, 2004.

[26] R. Solingen, "A follow-up reflection on software process improvement ROI," *IEEE Software,* vol. 26, pp. 77-79, 2009.

[27] L. Scott, R. Jeffery, L. Carvalho, J. D'Ambra, and P. Rutherford, "Practical software process improvement - the IMPACT project," in *Software Engineering Conference, 2001. Proceedings. 2001 Australian*, 2001, pp. 182-189.

[28] G. Santos, M. Kalinowski, A. Rocha, G. H. Travassos, K. C. Weber, and J. Antonioni, "MPS.BR: A tale of software process improvement and performance results in the Brazilian software industry," Porto, 2010, pp. 412-417.

[29] G. Scott, "Can software engineering afford to improve the process?," *SIGSOFT Softw. Eng. Notes,* vol. 17, pp. 39-42, 1992.

[30] M. Smith and P. Sperling, "The fire support software engineering division achieves CMMI level 5," *CrossTalk,* pp. 16-19, 2004.

[31] T. Cromer and J. Horch, "From the many to the one-one company's path to standardization," in *Software Engineering Standards, 1999. Proceedings. Fourth IEEE International Symposium and Forum on*, 1999, pp. 116-121.

[32] P. Keil and M. Kuhrmann, "An approach to model the return on investment of organization-wide improvement projects using the concept of external effects," presented at the Proceedings of the 2006 international workshop on Economics driven software engineering research, Shanghai, China, 2006.

[33] D. P. Kelly and B. Culleton, "Process improvement for small organizations," *Computer,* vol. 32, pp. 41-47, 1999.

[34] W. S. Humphrey, T. R. Snyder, and R. R. Willis, "Software process improvement at Hughes Aircraft," *Software, IEEE,* vol. 8, pp. 11-23, 1991.

[35] A. Parasuraman, V. Zeithaml, and L. L. Berry, "SERVQUAL - A Multiple-Item for Scale for Measuring Consumer Perceptions of Service Quality," *Journal of Retailing,* vol. 66, pp. 12-40, 1988.

[36] L. Monteiro and K. Oliveira, "Defining a catalog of indicators to support process performance analysis.," *Journal of Software Maintenance and Evolution: Research and Practice,* vol. 23, pp. 395–422, 2011.

How Does Refactoring Affect Understandability of Business Process Models?

Ricardo Pérez-Castillo, Maria Fernández-Ropero, Mario Piattini

Instituto de Tecnologías y Sistemas de Información (ITSI),
University of Castilla-La Mancha,
Paseo de la Universidad 4, 13071, Ciudad Real, Spain
[ricardo.pdelcastillo, marias.fernandez,
mario.piattini]@uclm.es

Danilo Caivano

Department of Informatics, University of Bari,
Via E. Orabona, 4, 70126 Bari, Italy
caivano@di.uniba.it

***Abstract*—Business process refactoring techniques have been often provided for business process manually modeled. Unfortunately, no many refactoring techniques lie in reversing business process models obtained from existing information systems, which need, even more, to be refactored. Hence, there is no strong empirical evidence on how the understandability of business process models is affected by this kind of refactoring techniques. This paper is aimed at providing a case study with two real-life information systems, from which 40 business process models were obtained by reverse engineering. The empirical study attempts to quantify the effect to the understandability of the order of refactoring operators as well as the previous refactoring actions. The main implication of the obtained results are a set of rules that may be used to optimize the understandability by means of the prioritization and configuration of refactoring techniques specially developed for business process models retrieved by reverse engineering.**

Keywords-Business Process, Refactoring, Understandability

I. INTRODUCTION

Business process models depict the sequence of coordinated activities that an organization carried out to achieve their business goal [22]. Business processes models are considered one of the most important assets for organizations due to two main reasons. An appropriate management of business process models first helps companies to quickly adapt their business goals and structures to environmental changes while maintaining or even improving their competitiveness [10]. Secondly, from a software engineering viewpoint, business process models are the starting point for obtaining the requirements of new-development or maintenance projects [19].

Since business processes exist within organization in an intangible way, business process modeling provides tangible descriptions of them allowing their management. Unfortunately, not all business processes are modeled in the organization, or when business processes are modeled, these might be out of date and therefore could be misaligned regarding the enterprise information systems that give support to such processes [9]. Similarly to the chicken-and-egg dilemma, there is no way to truly know which came first, business process models or enterprise information systems. In fact, outdated and misaligned business process models (together with organizations that deal with business process modeling at the first time) are the key motivations for reverse engineering techniques devoted to retrieving the actual business process models supported by the existing information systems [17, 20].

Reverse engineering techniques for obtaining business process models are often less error-prone and time-consuming than manual (re-)modeling from scratch. However, reverse engineering techniques imply an inherent semantic loss due to the abstraction increase [2]. As a result, although outdated and misalignment problems are addressed, quality of the retrieved models is eroded. Reverse engineering techniques could retrieve, for example, incomplete or inaccurate business process models (i.e., with missing and wrong elements), or even modes with inadequate understandability and modifiability levels (e.g., with a vast amount of fine-grained and ambiguity elements) [7].

In order to cope with understandability and modifiability faults, refactoring of business process models has been widely used [7]. These techniques change the internal structure of business process models without altering or modifying their external behavior. There exist in literature several refactoring approaches to be applied with business process models [3, 11, 21]. Unfortunately, there are no refactoring techniques specially developed for those models obtained through reverse engineering and some of their peculiarities such as missing elements, mining of non-relevant elements, fine granularity, and so on. In addition to this drawback, the main problem is that current refactoring techniques often apply several refactoring operators to deal with different *bad smells*, i.e., refactoring opportunities (e.g., non-relevant elements, fine-grained elements, etc.). The application of different refactoring operators is commonly done in an arbitrary way [7]. Nevertheless, it has been demonstrated that the order and subset of refactoring operators lead to different results in terms of the understandability and modifiability gain [6].

This paper therefore focuses on the assessment and optimization of the understandability of business process models during refactoring. Hence, this paper tries to provide a set of arguments and insights through empirical validation so

that the community can have a better answer to the question: *how affect refactoring to the business process model understandability?* In order to provide the mentioned insights for such answer this paper conducts a case study with two industrial information systems, from which 40 business process models were first obtained by reverse engineering. After that, those models were refactored by using IBUPROFEN [6], a refactoring approach, by setting up different orders and subsets of refactoring operators. IBUPROFEN is used in this study since this approach and its supporting tools were specially developed for refactoring business process models obtained by reverse engineering from existing source code. Finally, all the obtained business process models are inspected to evaluate the understandability gains and determine the best configurations.

The remainder of this paper is organized as follows. Section II briefly presents related work. Section III introduces IBUPROFEN, the approach used for refactoring. Section IV explains the case study in detail. Finally, Section V discusses conclusions and future work.

II. Related Work

Business process management has become a valuable activity for managing organizations from an operational perspective. *Dijkman et al.* [4] provide various techniques for improving their management as merging, mining, refactoring, re-use, among other. Particularly, refactoring has been used for several authors in literature for improving the quality degree of business process models. For example, *Weber et al.* [21] collect a catalogue of process model *smells* for identifying refactoring opportunities and provide a set of behavior-preserving techniques for refactoring to avoid redundancies and increase in the complexity of the model. Similarly, *Dijkman et al.* [3] show a development of a technique based on metrics to detect refactoring opportunities and *La Rosa et al.* [11] identify patterns to reduce the model complexity through compacting, compositing, merging, amoung other. *Leopold et al.* [12], for their part, focus on refactoring of activity labels in a business process model following a verb-object style.

Concerning to the order of application of the refactoring operators or the selection of a sub-set of operators, previous approaches rely on the expert decision, or simply define an arbitrary sub-set and order. Although *Gambini et al.* [8] propose the automation of de business process models refactoring through a technique for automatically fixing the refactoring scenarios using Petri nets, the order of application is not mentioned. *Fernandez-Ropero et al.* [6] demonstrate that the order of application of refactoring operators affect the understandability and modifiability. However, that preliminary work does not assess the best sub-sets or application orders to achieve the highest understandability.

III. IBUPROFEN

IBUPROFEN [6] (Improvement and BUsiness Process Refactoring OF Embedded Noise) is a framework with which to refactor business process models particularly retrieved by reverse engineering. IBUPROFEN allows applying different refactoring operators taking into account the assessment of various measures related to the modifiability and understandability of business process models [7] such as density, size, connectivity, separability, etc.

IBUPROFEN is supported by a tool specially designed for business process models represented according to the BPMN (Business Process Modeling Notation) [14]. The tool has being implemented as an Eclipse™ plug-in [1]. Hence, the supporting tool can be used in combination with other Eclipse™ plug-ins aimed, for example, at obtaining business process models from the source code of existing information systems.

IBUPROFEN provides a set of ten refactoring operators (see TABLE I) grouped into three categories in terms of the bad smells that the operators address: (i) relevant elements maximization; (ii) fine-grain granularity reduction; and finally, (iii) completeness maximization.

A. Relevant Elements maximization

This category groups five refactoring operators (R1 to R5) responsible for removing non-relevant elements found in business process models as isolated tasks, sheet tasks and inconsistencies. Moreover, nested gateways can origin an increase in the complexity of business process models, thus these are replaced by equivalent, light-weight structures.

R1 removes nodes (i.e., tasks, gateways or events) in the business process model that are not connected with any other node in the business process model. **R2** discards elements in the business process model that are considered sheet nodes. These nodes can be gateways or intermediate events that have no successor nodes. In turn, **R3** merges consecutive gateways of the same type when the first gateway has only one output and the second has only one input, i.e., nested gateways. **R4** removes sequence flows in the business process model that are considered as inconsistent. When two tasks are connected through a cut node, as an intermediate event or a gateway, and through a direct sequence flow this sequence flow are removed. Finally, **R5** removes gateways that connected only two nodes, i.e. with one input and one output. Such gateways are removed and a direct sequence flow is created between related nodes.

B. Fine-grained granularity reduction

The different granularity of business tasks and callable units in existing information systems constitutes another important challenge [17]. According to the approach proposed by *Zou et al.* [24], each callable unit in an information systems is considered as a candidate business task. However, existing systems typically contain thousands of callable units, some of which are large ones supporting the main business functionalities of the system, while many are very small and do not directly support any business activity. In other situations a set of small callable units together supports a business activity. As a consequence, this category provides two refactoring operators (R6 and R7) to deal with large sets of fine-grained business tasks and data objects:

R6 transforms each task in a compound task when the task *T* has several subsequent tasks which are in turn connected with a round-trip sequence flow to the task *T*. This scenario is due to each callable unit is transformed as a task during the reverse engineering stage when a certain callable unit can invoke another callable unit returning a value to the first one. In this case, the refactoring operator creates a compound task with a

start and end event connected with each subsequent task through the respective split and join exclusive gateways. Additionally, **R7** combines data objects that are input and/or output of a task. The combination is possible when those data objects are exclusively used (written or read) for that task. The combination is done when the number of data objects is above a threshold. In order to mitigate the collateral semantic loss, all the names of the grouped data objects are saved in the documentation attribute defined by the BPMN specification.

C. Completeness Maximizatioin

Any reverse engineering technique implies an increase of the abstraction degree, and therefore a semantic loss. For this reason, R8 to R10 operators are provided to deal with semantic loss by means of the incorporation of further elements. The refactoring operators are the following:

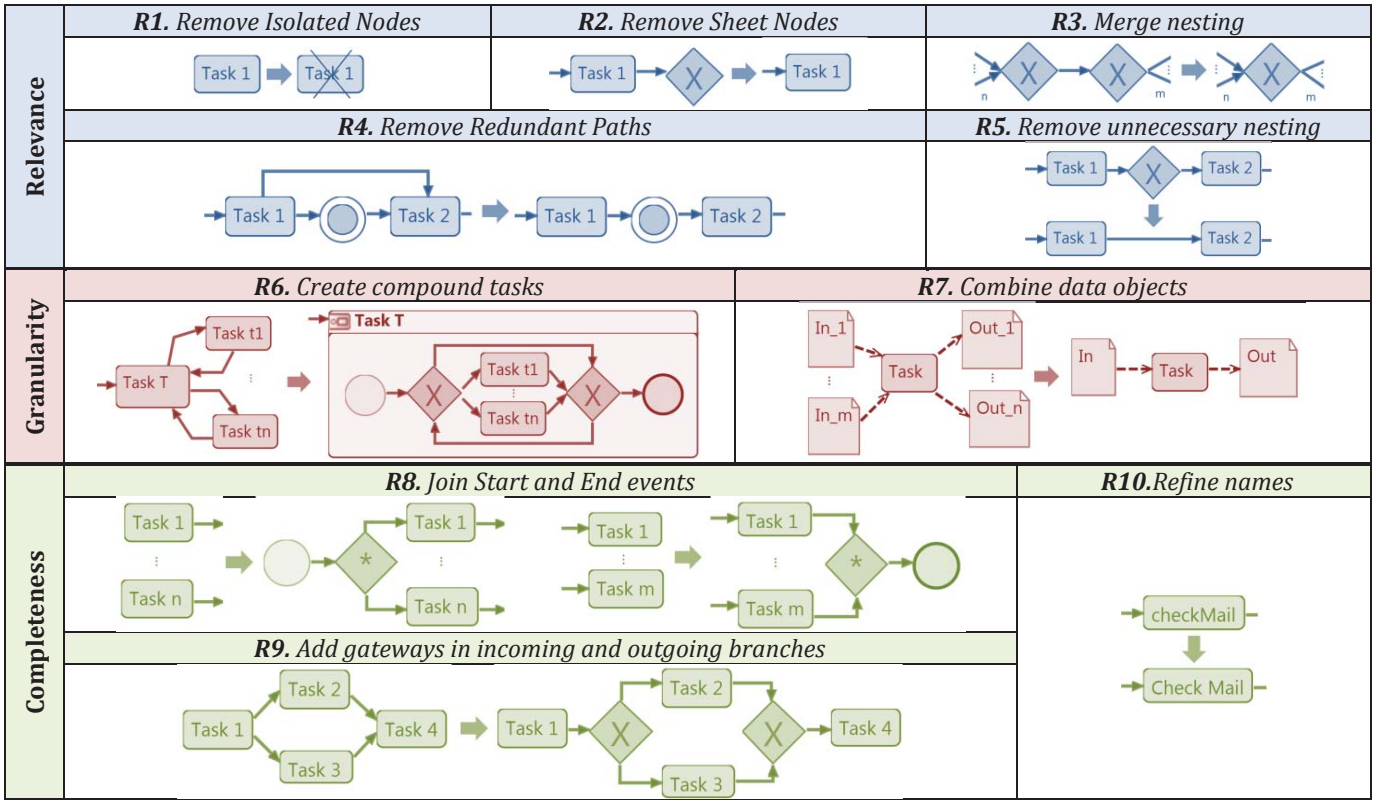
R8 joins the start and end event with the starting and ending tasks, respectively. These events are created whether such events were not created by reverse engineering. When there are several starting tasks the refactoring operator adds a split complex gateway between the start event and starting tasks. Similarly, if there are several ending tasks, the refactoring operator adds a join complex gateway between ending tasks and the end event [13]. Furthermore, due to the usage of reverse engineering to retrieve business process models, it is possible to obtain models without following some of the modeling guidelines in accordance with the BPMN specification with regard to the gateways. **R9** therefore adds a join and split exclusive gateways when a certain task respectively has several precursor or subsequent tasks. Finally, **R10** improves names and labels of business tasks that were obtained almost directly from methods or functions of legacy source code through reverse engineering. These labels usually follow the camel case format (i.e., the concatenation of various capitalized words) in accordance with naming conventions present in most programming approaches. In an effort to have more understandable names, this refactoring operation split these labels into ones with various words.

IV. Case Study

This section provides a case study with two real-life information systems. The case study has been conducted by following the formal protocol developed by *Runeson et al*. [18] for conducting and reporting case studies in the software engineering field. Hence, the following sections show the stages proposed in the formal protocol: case study design, case selection procedure, execution procedure and data collection, analysis and interpretation, and finally, threats to the validity.

The *object* of this case study is the understandability of business process models after refactoring and the *purpose* of this case study is to evaluate how the execution order of the different refactoring operators and previous refactoring actions affect to the understandability. Taking into account the object and purpose of the study two main research questions are provided.

TABLE I. IBUPROFEN'S REFACTORING OPERATORS



RQ1: *How does the order of the application of refactoring operators affect to the understandability of business process models?*

RQ2: *How does previous refactoring affect to the understandability achieved with the application of certain refactoring operators?*

A. Case Study Design

The case study follows the *embedded case study* design according to the classification proposed by *Yin* [23], whereby the case study consists of a multi case (i.e., it focuses on two information systems) but considers several analysis units as *independent variable* within the case, i.e., all the different business processes models retrieved from both information system. Therefore, the study consists of applying the three refactoring categories: relevance (R), granularity (G) and completeness (C) in different combinations and obtaining business process models. Such models are in turn analyzed to evaluate understandability in accordance with RQ1 and RQ2. In order to quantify understandability, size, connectivity, separability and density [5] measures are used as *dependent variables*.

Size is the number of nodes in a business process model (i.e., business tasks, gateways, data objects and events). This measure affects negatively to the understandability, i.e. a higher size difficult the understandability of a certain business process model [13]. **Connectivity** measures the ratio between the total number of arcs in a business process model (i.e., sequence flows and associations) and the total number of nodes. This measure negatively affects the understandability since a lower connectivity implies business process models more understandable due to a lower intricacy. **Separability** represents the ratio between the number of cut-vertices in a business process model (i.e. nodes that serve as bridges between otherwise strongly-connected components) and the total number of nodes. Separability positively affects to the understandability. **Density** is the ratio between the total number of arcs in a business process model and the theoretical maximum number of possible arcs regarding the number of nodes. The lower density, more understandable business process models.

B. Case Selection Procedure

To select the case under study a set of selection criteria were formulated in order to rigorously select the source system: (1) the system should be a real-life information system currently in production; (2) and with a considerable size (to avoid toy programs) which ensure that the system supports a great number of business processes; (3) the system should be written in Java language to be able to use the MARBLE tool [15]. MARBLE is the tool used to recover business process models from existing Java code. This tool was selected because is released as an Eclipse plug-in and it therefore can be easily integrated with the IBUPROFEN tool.

After analyzing various information systems of partner companies, two cases were selected in accordance with the mentioned criteria: *Tabula* and *XCare*. *Tabula* is a web application of 33.3 KLOC (thousands of lines of code) devoted to create, manage and simulate decision tables for associating conditions with domain-specific actions. *XCare* is a mobile application of 9.9 KLOC intended for diabetes patients, which analyzes blood (through an external device) and suggests diet plans.

C. Execution Procedure and Data Collection

The procedure to be performed to execute the case study consists of a set of steps. (i) A sample of 40 business process models are mined, by using MARBLE [16], from the source code from both information systems under study. (ii) After that, IBUPROFEN refactoring operators are executed in all the possible orders in terms of the three categories, so six different execution orders are considered (i.e., RGC, RCG, CRG, CGR, GCR and GRC). (iii) The mentioned measures are computed through IBUPROFEN tool after the execution of each category as well as before refactoring (i.e., four measurements for each execution order are taken). These semiautomatic steps are executed in a computer with a 2.66 GHz dual processor and 4.0 GB RAM.

Data collected during execution is used to compute the normalized gains after the execution of each category. TABLE II presents the normalized gains for each previous combination of refactoring category. This data represents the gain evolution for all the measures in accordance with the position in which a category is executed and regarding to the previous refactoring actions. Size, density, connectivity and separability cells are mean values computed for all the 40 business process models. The whole data, including base data directly obtained from the execution of the study is online available[1].

TABLE II. GAIN ON AVERAGE FOR EACH CATEGORY WITH DIFFERENT ORDERS

Cat.	Pre-Act.	Size	Density	Connectivity	Separability
Relevance	-	0.390	-3.959	-0.597	0.470
	G	0.500	-7.798	-0.954	0.548
	C	0.127	-0.669	-0.171	0.154
	GC	0.157	-0.896	-0.231	0.184
	CG	0.142	-0.848	-0.207	0.172
Granularity	-	0.269	0.051	0.218	0.064
	R	0.231	-0.199	0.146	0.070
	C	0.072	-0.067	0.022	0.068
	RC	0.107	-0.114	0.028	0.059
	CR	0.085	-0.103	0.009	0.078
Correctness	-	-0.476	-0.318	-0.601	-0.325
	R	-0.341	0.163	-0.082	-0.152
	G	-0.530	-0.793	-1.252	-0.373
	RG	-0.477	0.080	-0.315	-0.280
	GR	-0.459	0.140	-0.225	-0.256

D. Analysis and Interpretation

The inspection of data collected in TABLE II suggests that results highly vary with regards to the order in which each refactoring category is applied. These values also depend on the previous refactoring applied. However, in order to figure out whether these observations reflect a common pattern rather than the random effect, a statistical hypothesis testing were conducted for assessing the real effect of the application order.

For this purpose, the *Kruskal-Wallis* (KW) test was used. The KW test is a non-parametric method supporting a one-way analysis of variances by ranks. The KW test is used for comparing more than two non-related samples. Thus, the null

[1] http://alarcos.esi.uclm.es/per/mfernandez/

hypothesis is H_0: $\mu_1 = \mu_2 = \mu_n$, while the alternative hypothesis means that there is a significant difference between the means of sub-samples, i.e., H1: $\mu_1 \neq \mu_2 \neq \mu_n$. In this study, the different sub-samples were selected according to the five different configurations (order and previous actions). For example, the five samples of relevance (R) are in which R is applied at the beginning, is applied in second place (CR or GR), or is applied at the end (CGR or GCR). TABLE III provides the results of the KW test, whose inspection shows that the order (RQ1) and previous refactoring of all the categories (RQ2) affect the gain achieved at least for some of the measures. In case of relevance, the configuration affects to all the measures. In case of granularity, the order and previous refactoring affect to size, density and connectivity gain, but do not affect to separability. Finally, in case of completeness, the configuration only affects to density and connectivity. These results demonstrate that the application in an arbitrary order is not a good idea.

Having known there is a difference between different configurations, it is necessary (in order to complete the answer of research questions RQ1 and RQ2) to figure out which certain configuration is better than other in each category. Figure 1 graphically shows these variances. Regarding Relevance, the best choice was to apply it in the second place after granularity if the goal is to maximize size and separability. However, density and connectivity gains, which are always negative, are better if the relevance category is applied in second place after correctness refactoring. Concerning granularity, the best combination was to apply it at the beginning to achieve the greatest gain of size, density and connectivity. However, the best separability was achieved when granularity is applied at the end after correctness and relevance categories. Anyway, the differences of separability gains are negligible for every order (see Figure 1). Finally, with respect to completeness, most gains are unfortunately negative. Despite this fact, the best order in every case is to apply completeness in the second place after relevance.

After analyzing outgoing results, some rules to prioritize the application of refactoring categories can be derived so that research question can be fully answered. The first insight is that refactoring operators related to relevance should be applied in second place. Particularly, after granularity refactoring if the gain of size and separability are prioritized and after completeness if density and connectivity gain has to be maximized. The second rule is about granularity category, which should be applied in the first place. The third rule about completeness states that it should be applied in second place after relevance refactoring operators.

E. Validity Evaluation

This section presents the threats to the validity of this case study and possible actions to mitigate them. There are mainly three types of validity: internal, construct and external. As far as the *internal* validity is concerned, a sample of 40 business process models was retrieved from a two information systems, and it is therefore possible to obtain statistically representative results. Nevertheless, the study may be replicated by using more information systems, to attain a larger sample of business process models. Anyway, there are two decisive threats. The first one is related to the way in which business process models were retrieved by reverse engineering, i.e., through MARBLE. This supporting tool was used to obtain the business process models, could be a factor that affects the initial sample of business process models. Secondly, the set of refactoring operators included in IBUPROFEN as well as their categories is a threat to the generalization of the results. The replication of the study by using different refactoring operators and techniques may be a mean for mitigating these threats.

TABLE III. KRUSKAL-WALLIS TEST RESULTS

	Size		Density		Connectivity		Separability	
	χ^2	Sig.	χ^2	Sig.	χ^2	Sig.	χ^2	Sig.
Relevance	48.8	0.000	20.4	0.000	24.1	0.000	52.5	0.000
Granularity	24.6	0.000	21.7	0.000	25.8	0.000	1.6	0.801
Completeness	3.45	0.485	35.6	0.000	44.7	0.000	5.7	0.226

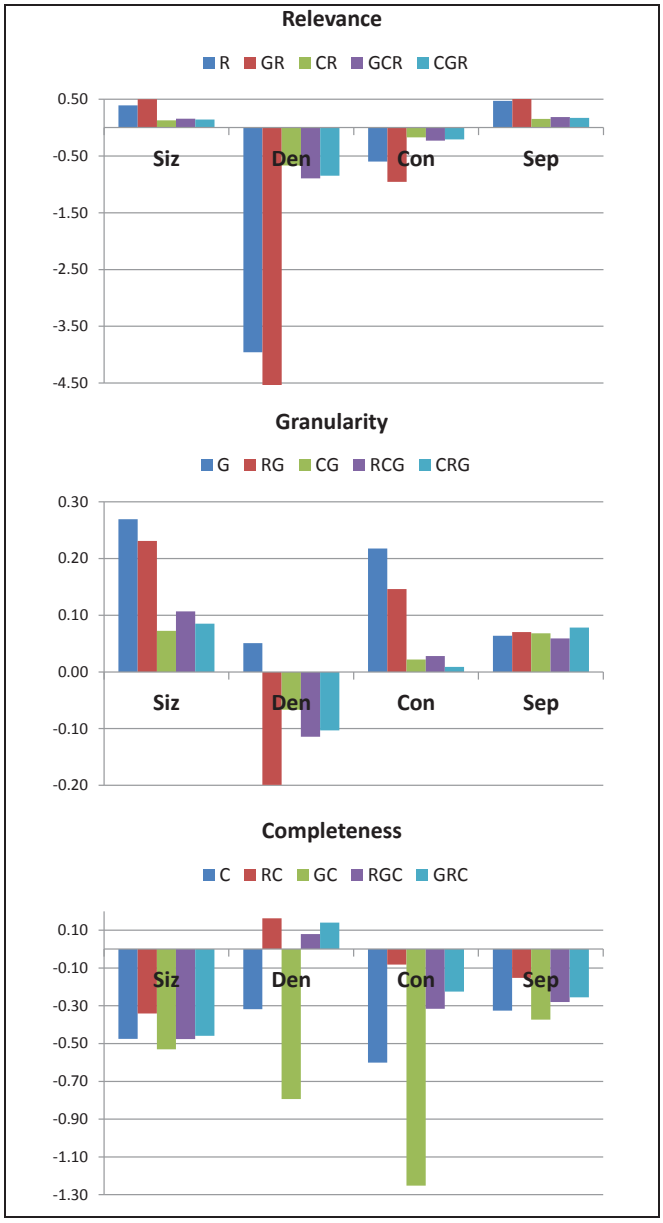


Figure 1. Behaviour of categories with different orders and previous actions

Moreover, with respect to the *construct* validity, the selected measures (size, density, connectivity and separability) were suitable for assessing the theoretical understandability of business process models. However, a more practical approach based on expert viewpoint could be used to assess the understandability of business process models. Finally, *external* validity is concerned with the generalization of the results. This study considers the whole population to be business process models retrieved by reverse engineering from legacy information systems. The results obtained can be strictly generalized to this population with the particularity that all the information systems under study are based on Java platform. This restriction is related to the mentioned supporting tools used in the study. This threat may be mitigated by replicating the study using systems implemented in different platforms.

V. Conclusions and Future Work

Business process model refactoring has proved to be a good mechanism for dealing with understandability problems and other faults. Unfortunately, most refactoring techniques only address business process models manually modeled and hardly ever consider reversing models semi-automatically retrieved from existing information systems. This paper precisely focuses on this kind of refactoring techniques by means of an empirical study that tries to assess how different configurations of refactoring affect the understandability gain. However, the understandability of business process model is difficult to be measured. On one hand, the understandability additionally depends on the people in charge of use, manage or evaluate such business process models, which is individually subjective. On the other hand, understandability of business process models that were previously refactored could vary due to the application of different refactoring operators. In fact, some operators might lead to a worse understandability. This study precisely attempts to establish links between different refactoring configurations (in terms of categories applied, i.e., relevance, completeness and granularity) and the understandability gain, which is measured with size, density, connectivity and separability of business process models. The study's results reflect that refactoring categories can be prioritized concerning the order in which to be applied as well as the previous refactoring actions so that the understandability gain can be optimized.

Acknowledgments

This work was supported by the FPU Spanish Program and the R&D projects PEGASO/MAGO (TIN2009-13718-C02-01) and GEODAS-BC (TIN2012-37493-C03-01).

References

[1] Alarcos Research Group. IBUPROFEN. 2012; Available from: http://marketplace.eclipse.org/node/423052.

[2] Canfora, G., M. Di Penta, and L. Cerulo, Achievements and challenges in software reverse engineering. Commun. ACM, 2011. 54(4): p. 142-151.

[3] Dijkman, R., B. Gfeller, J. Küster, and H. Völzer, Identifying refactoring opportunities in process model repositories. Information and Software Technology, 2011.

[4] Dijkman, R., M.L. Rosa, and H.A. Reijers, Managing large collections of business process models—Current techniques and challenges. Computers in Industry, 2012. 63(2): p. 91.

[5] Fernández-Ropero, M., R. Pérez-Castillo, I. Caballero, and M. Piattini, Quality-Driven Business Process Refactoring, International Conference on Business Information Systems (ICBIS 2012). 2012 p. 960-966.

[6] Fernández-Ropero, M., R. Pérez-Castillo, J.A. Cruz-Lemus, and M. Piattini, Assessing the Best-Order for Business Process Model Refactoring. 2013. p. 1400-1406.

[7] Fernández-Ropero, M., R. Pérez-Castillo, and M. Piattini, Refactoring Business Process Models - A Systematic Review, in 7th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2012). 2012, INSTICC: Wroclaw, Poland. p. 140-145.

[8] Gambini, M., M. La Rosa, S. Migliorini, and A. Ter Hofstede, Automated error correction of business process models. Business Process Management, 2011: p. 148-165.

[9] Heuvel, W.-J.v.d., Aligning Modern Business Processes and Legacy Systems: A Component-Based Perspective (Cooperative Information Systems). 2006: The MIT Press.

[10] Jeston, J., J. Nelis, and T. Davenport, Business Process Management: Practical Guidelines to Successful Implementations. 2nd ed. 2008, NV, USA: Butterworth-Heinemann (Elsevier Ltd.). 469.

[11] La Rosa, M., P. Wohed, J. Mendling, A.H.M. ter Hofstede, H.A. Reijers, and W. van der Aalst, Managing process model complexity via abstract syntax modifications. Industrial Informatics, IEEE Transactions on, 2011. 7(4): p. 614-629.

[12] Leopold, H., S. Smirnov, and J. Mendling, Refactoring of process model activity labels, in Proceedings of the Natural language processing and information systems, and 15th international conference on Applications of natural language to information systems. 2010, Springer-Verlag: Cardiff, UK. p. 268-276.

[13] Mendling, J., H.A. Reijers, and W.M.P. van der Aalst, Seven process modeling guidelines (7PMG). Information and Software Technology, 2010. 52(2): p. 127-136.

[14] OMG. Business Process Modeling Notation Specification 2.0. 2011; Available from: http://www.omg.org/spec/BPMN/2.0/PDF/.

[15] Pérez-Castillo, R., M. Fernández-Ropero, I. García Rodríguez de Guzmán, and M. Piattini, MARBLE. A Business Process Archeology Tool, in 27th IEEE International Conference on Software Maintenance (ICSM'11). 2011, IEEE Computer Society: Williamsburg, Virginia, USA. p. 578-581.

[16] Pérez-Castillo, R., M. Fernández-Ropero, I.G.-R.d. Guzmán, and M. Piattini, MARBLE. A Business Process Archeology Tool, in 27th IEEE International Conference on Software Maintenance (ICSM 2011). 2011: Williamsburg, VI. p. 578 - 581

[17] Pérez-Castillo, R., B. Weber, I. García Rodríguez de Guzmán, and M. Piattini, Generating Event Logs from Non-Process-Aware Systems Enabling Business Process Mining. Enterprise Information System Journal, 2011. 5(3): p. 301–335.

[18] Runeson, P. and M. Höst, Guidelines for Conducting and Reporting Case Study Research in Software Engineering. Empirical Softw. Eng., 2009. 14(2): p. 131-164.

[19] Sommerville, I., P. Sawyer, and S. Viller, Viewpoints for Requirements Elicitation: A Practical Approach, in Proceedings of the 3rd International Conference on Requirements Engineering: Putting Requirements Engineering to Practice. 1998, IEEE Computer Society. p. 74-81.

[20] van der Aalst, W., Process Mining: Overview and Opportunities. ACM Transactions on Management Information Systems (TMIS), 2012. 3(2): p. 7.

[21] Weber, B., M. Reichert, J. Mendling, and H.A. Reijers, Survey paper: Refactoring large process model repositories. Comput. Ind., 2011. 62(5): p. 467-486.

[22] Weske, M., Business Process Management: Concepts, Languages, Architectures. 2007, Leipzig, Germany: Springer-Verlag Berlin Heidelberg. 368.

[23] Yin, R.K., Case Study Research. Design and Methods. 3rd ed. 2003, London: Sage.

[24] Zou, Y. and M. Hung, An Approach for Extracting Workflows from E-Commerce Applications, in Proceedings of the Fourteenth International Conference on Program Comprehension. 2006, IEEE Computer Society. p. 127-136.

A multi-dimensional approach for analyzing software artifacts

Sébastien Adam, Ghizlane El Boussaidi
Department of Software and IT engineering
École de technologie supérieure
Montréal, Canada

***Abstract*—During a software project the development team deals with various artifacts such as requirements, models, design patterns, and procedures. Each artifact has its proper features and issues that may threaten the success of a project. This paper introduces an approach to promote a methodical and multi-dimensional analysis of the linkages among the artifacts' features and issues by using networks of weighted and semi-formal arguments. These networks of data may be used to produce quantitative information in different multi-dimensional views to ease the identification of critical artifacts and issues of a software project. The Template Method design pattern is used as an example of artifact to identify a coherent set of features and issues related to this pattern. This case study illustrates how the designers may use the proposed approach to manage software artifacts of their system.**

Keywords-software knowledge management, software artifacs, multi-dimension analysis, decision support systems

I. INTRODUCTION

During the development of a software system, the development teams deal with numerous artifacts such as requirements, models, design patterns, and procedures. Each artifact can be characterized using a set of features and it relates to a set of issues that are factors of influence that may threaten the success of a project. For instance, a design pattern [1] is a design artifact that is characterized by a rationale, a solution, plausible consequences, and trade-offs to be considered when used. Somehow, the artifacts constitute the assets that embody decisions and trade-offs applied during the project.

For the development teams to be efficient, the projects' artifacts must be managed and shared in an efficient fashion. Indeed, designers must evaluate how the most influential artifacts impact the capacity of the system to satisfy the stakeholders' needs. Insufficient details about these artifacts and their relationships may lead teams to inappropriate or suboptimal decisions. In particular, several approaches propose, to some extent, a process or a technique aiming at relating software artifacts (e.g., [3, 7, 8, 11, 13, 14, 15, 17]). These approaches usually focus on a subset of the artifacts involved in the development process and on some specific development perspective that is part of this process. However, there is a lack of works that support a multi-dimensional view and a methodical treatment of the artifacts and their related factors of influence, which include artifacts' features, issues, and arguments that influence the activities and dimensions of a software development project.

Designers' expertise and experiences remain the key elements in identifying the critical factors to their projects and the appropriate solutions to these factors. This is true at different stages of the software development process (e.g., analysis, design, or implementation) and for different projects' dimensions (e.g., budget, quality, and schedule). The accumulated knowledge related to software engineering should be addressed in an integrated and systematic manner to enable the development of decision support systems that: 1) relate the software artifacts to their factors of influence; 2) offer support to use the relevant artifacts and to appropriately solve their related issues; and 3) keep track of the adopted arguments and resolved issues.

In this paper, a multi-dimensional analysis approach to efficiently use and relate software artifacts is proposed. This is a two-phase approach for analyzing and identifying critical factors related to software artifacts for a given project. The first phase of the approach is the preparation phase, which aims at eliciting the artifacts' knowledge in the form of inter-related arguments. The second phase is the analysis phase where this knowledge is used to provide views that ease identifying critical factors to a project. As an example of artifact, the Template Method design pattern [1] and its related factors are presented. Then, these factors are analyzed and ranked for producing multi-dimensional views of the pattern that highlight critical factors. The approach have been applied in the context of an undergraduate course project for validating its usefulness especially for novice designers. The contributions of this paper are: 1) reusable specifications of artifacts based on a uniform argument format; 2) a systematic method for executing multi-dimensional analysis of artifacts; 3) a flexible method to transform networks of arguments to multi-dimensional views.

This paper is organized as follows. Section II gives an overview of the proposed approach. Section III introduces the Template Method that is used to illustrate the preparation and analysis phases of the proposed approach in sections IV and V, respectively. Section VI details a case study realized in the context of an undergraduate course. Finally, sections VII and VIII present some related works and conclude.

II. OVERVIEW OF THE MULTI-DIMENSIONAL ANALYSIS

Our approach aims at applying multi-dimensional analysis using a set of factors that characterize software artifacts and influence software engineering. In this paper, a factor may be a feature, an issue, an argument, an activity, or a dimension. Table I describes each of these factors.

TABLE I. FACTORS OF THE MULTI-DIMENSIONAL ANALYSIS

Factor	Description
Feature	Distinguishing characteristic of an artifact
Issue	Problem that emanates by using a feature
Argument	Reasoning about an issue or a solution
Activity	Set of cohesive tasks
Dimension	Either functions, quality, cost, schedule, or staff

A feature is a distinguishing artifact's characteristic that we can conceptually use for analyzing its impacts on a project. Features may be of various types (e.g., a property describing a characteristic promoted by the artifact or a role assigned to a person within the process of implementing the artifact). Each feature relates to some inherent issues. For example, implementing a behavior (feature) may require capabilities that are not mastered by the work team. Different artifacts may share similar features. Both artifacts and their features relate to activities (e.g., design, implementation) and dimensions (i.e., budget, quality) that are determinants of successful projects. Finally, arguments are what tie factors altogether. They encode the reasoning about how a set of factors relate to each other and influence a system.

Our approach is a two phases approach: a preparation phase that aims at eliciting the factors that describe the system's artifacts, followed by an analysis phase that aims at analyzing these factors in order to create multi-dimensional views that shall enable identifying important factors of a software project. Both phases are independent. The results of the preparation phase may be used for multiple executions of the analysis phase. Specifically the preparation phase is organized into three steps: 1) Eliciting features and related activities; 2) Eliciting issues and impacted dimensions; and 3) Eliciting arguments. The analysis phase is divided into three steps: 1) Selecting factors and building generic views for the artifacts under analysis; 2) Ranking factors according to the context of the project and generating contextual views; and 3) Identifying important factors of the project using the contextual views. The approach will be illustrated through the analysis of the Template Method (TM) design pattern as an example of a software artifact.

III. THE TEMPLATE METHOD DESIGN PATTERN

The Template Method (TM) design pattern [1] is used for providing reusability and extensibility of algorithms in object-oriented software. It aims to implement the skeleton of an algorithm in a base class, and calls primitive methods that subclasses override to provide concrete behavior. The base class interface declares the algorithm as a template method, which calls abstract primitive methods that represent the algorithm's variation points. The subclasses implement the primitives for specializing the algorithm. The template method may be declared final in order to ensure that it cannot be overridden. The primitive methods may be declared protected in order to ensure that they cannot be called by another algorithm. As a result, the algorithm's structure is written only once and indirectly specialized in subclasses, which reduces duplication of code and enforces class interface stability. Also, the template method allows the addition of instrumentation in the base class, and lightens users' duty since it is no longer required to call a primitive.

IV. PREPARATION PHASE

A. Eliciting features and related activities

Table II presents some of the distinguishing features we identified from the analysis of various TM descriptions as given in the literature. Each feature has a description and is classified under a specific type. Table III summarizes the feature types that relate to the TM pattern. We classified each feature type as part of the problem space or the solution space related to the TM pattern. We use these feature types as a mean to ease the elicitation of features and to constrain their possible interpretations.

The Rationale type regroups the features that provide reasoning about the problem. The rationale of the TM pattern summarizes what the design pattern does. The Property type regroups the drivers that define the problem. Clements *et al.* [2] define a property as additional information about entities and relations, such as names and attributes of quality [3]. Each pattern may promote or disadvantage one or more properties. For instance, the TM pattern promotes the object-oriented paradigm, and reusability and extensibility of software. Pattern descriptions fix a subset of elementary actions (Operationalization type) and behaviors (Behavior type) that shall lead to their appropriate implementations. The Convention type regroups de facto standards that define guidelines for the problem to be solved. The domain objects (Role type) and the structures of elements (Structure type) have roles in realizing the solution. A domain object may be a human, a device, or another software interacting with the system to execute some tasks. The situational factors (Situational Factor type) describe the environment and hypotheses that influence this artefact. For the TM pattern, the requirements imply the use of an object-oriented programming language to define an abstract class that subclass writers extend with their specific concrete classes.

TABLE II. DISTINGUISHING FEATURES OF THE TM

Feature			Act.
Id	**Type**	**Description**	
Ra1	Rationale	Define an algorithm, defer steps to subclasses	D
Pr1	Property	Object-oriented paradigm	AD
Pr2	Property	Reusability	AD
Pr3	Property	Extensibility	AD
Be1	Behavior	Template method calls primitive operations	DI
Op1	Operational.	Define an abstract base class	DI
Op4	Operational.	Define a template method	DI
Op5	Operational.	Define a concrete child class	DI
Op7	Operational.	Declare a final template method	DI
Op8	Operational.	Declare protected primitives operations	DI
Be2	Operational.	Hook operations do nothing by default	DI
St1	Structure	Abstract class	I
St2	Structure	Concrete class	I
St3	Structure	Object-oriented programming language	I
Ro1	Role	Subclass writers	M
SF1	Situational factor	Multiple kinds of primitive operations	ADIM
Co1	Convention	Naming convention	IM

It is important to address each artifact's feature during the development activities that cause most beneficial influences. An activity is a set of cohesive tasks intended to contribute to the achievement of a common goal. Table II classifies each feature of the TM pattern based on these criteria. We considered four important activities: architecting (**A**), designing (**D**), implementing (**I**), and managing (**M**). Architecting deals with the properties and high-level structures (e.g., paradigm or platform selection) that shape the software. Designing deals with the detailed structures (e.g., abstract class) and the requirements that refine the properties. Implementing usually deals with more specific features (e.g., final method). Managing deals with roles (e.g., subclass writers), situational factors, and conventions (e.g., naming convention) that constitute the organizational system.

B. Eliciting issues

We analyzed the TM pattern to identify issues that may hinder its usage. Table IV lists some issues and the features they relate to. Due to lack of space, we present only few of the numerous issues we identified. Each feature may solve or engender one or more issues. For example, the extensibility property (feature) may not be well defined for a module (issue). Also, to lighten subclass writers' responsibility, the template methods call the primitive operations (feature). However, the uncontrolled calls to primitive operations (issue) may cause problems. To address this issue, the pattern proposes to declare protected primitive operations (feature). Our approach proposes to use a semi-formal argument format to describe the issues.

C. Eliciting arguments and impacted dimensions

One important objective of the preparation phase is to elicit arguments that will be used during the analysis phase to estimate the impacts of each issue, which may differ depending on the context of use of an artifact. Argumentation is concerned with reasoning in the presence of imperfect knowledge by eliciting arguments for exploring issues rather than eradicating them [4]. In our approach, the argumentation is geared towards quantifying the impacts of the factors on software development dimensions and activities. We used the format presented in Table V (adapted from [5]) to describe the arguments. The claim is the conclusion of the argument and it exposes the relationships between a set of features. The claim is unique for each argument. The description provides the chain of reasoning that ties altogether the argument's parts. The reasons are related arguments that support the claim. The rebuttals are counter-arguments for the claim. The alleviations are arguments that affect the intensity of the argument. Reasons, rebuttals, and alleviations are connection points. The two last parts of the argument format specify the scope of the argument. It refers to activities and dimensions that are strengthened (+) or weakened (-) by the argument. The activities are inferred from the activities related to the features exposed in the argument's claim while the dimensions are inferred from the dimensions impacted by the issue that prompted the argument.

TABLE III. DISTINGUISHING FEATURE TYPES RELATED TO THE TM

Space	Feature type	Description
Problem	Convention	A de facto standard that define guidelines
	Rationale	A reasoning about the problem
	Situational factor	A factor of the context that is problematic
Solution	Behavior	An interaction among a set of elements
	Operationalization	An operation that is part of a module interface
	Property	An additional information about elements/relations
	Role	An allocation of responsibilities
	Structure	A set of elements and relationships
	Procedure	A domain object interacting with the system

TABLE IV. ISSUES RELATED TO THE TM FEATURES

Feature	Issue description
Co1	The naming convention is not well defined
Be1	The template method behavior is subject to change
Op1	The deferred steps are not well known
Op8	The hook operations are not well identified
Pr1	The object-oriented paradigm is not well mastered
Pr2	The reusability objectives are not well defined
Pr3	The extensibility objectives are not well defined
Ro1	The subclass writers do not discern which methods must be overridden
St3	The programming language is not well mastered

TABLE V. ARGUMENT FORMAT USED TO RELATE THE FACTORS

Claim: conclusion of the argument.
Description: description of the argument.
Reasons*:* evidences that support the claim.
Rebuttals*:* evidences that establish the falsity of the claim.
Alleviations*:* evidences that reduce the intensity of the claim.
Dimensions: dimensions impacted by the claim.
Activities: activities concerned by the claim.

The project's dimensions we consider in our approach are (adapted from [6]): Functions (**F**) – What is the estimated impact of the claim in terms of the capacity of the software to accomplish (+) or not (-) its functions? Quality (**Q**) – What is the estimated impact of the claim in terms of the capacity of the software to deliver (+) or not (-) previsioned quality? People (**P**) – What is the estimated impact of the claim in terms of the capacity of a human to accomplish (+) or not (-) his task? Budget (**B**) – What is the impact of the claim in terms of the number of budgeted resources saved (+) or invested (-) to resolve the issue? and Schedule (**S**) – What is the estimated impact of the claim in terms of the number of work hours saved (+) or invested (-) to resolve the issue?

Table VI presents some of the arguments we elicited to establish how each issue of the TM pattern impacts projects' dimensions (F, Q, P, B, S) and activities (M, A, D, I). For example, the argument (Arg1) predicts positive impacts on the functional dimension (F+) by declaring a final method. One reason is that a final method cannot be overridden. One rebuttal or reservation is that it is possible to hack the final mechanism (Arg7). The argument refers to features (i.e. feature OP7) that may concern both design (D) and implementation (I) activities. The prevision is not weighted during the elicitation step because the elicited arguments are not project-specific. They can be reused among projects with other situational factors. The arguments are weighted during the analysis phase where a specific project is analyzed.

TABLE VI. ARGUMENTS THAT RELATE TO THE TM

Id	Claim	Rea	Reb	All	Dim	Act
1	A final method cannot be overridden by subclasses		7		+ FQ	DI
2	Programming require skills for object-oriented	5, 6, 11, 12			- BPS	M
6	The low cohesion reduces the analysability of modules			9	- BPQS	ADI
7	The final mechanism is hackable				- FQ	DI
9	The class cohesion is proper for the team's				+ BPQS	DI
11	The low cohesion makes maintaining more tedious	5, 6			- BFPQS	ADIM
15	The template method is subject to change	22, 24, 25, 26,			- BQS	DI
22	The extensibility objectives are not well defined				- BQS	ADI
24	There are too many primitive methods				- Q	DI
25	The deferred steps are not well known	22			- BQS	DI
29	The hook operations are not well identified	22			- BFQS	DI

V. ANALYSIS PHASE

During the analysis phase, we use the artifact's factors that were elicited in the preparation phase to engender multi-dimensional views for assessing the factors' impact in different contexts. It is a three steps phase. The planning step aims at selecting factors and building generic views of networked arguments related to these factors. The execution step aims at ranking factors according to the specific context of the project and generating weighted views. These contextual views are used to identify critical factors to be addressed by designers.

A. Selecting factors and building generic views

Consider the TM pattern as our artifact under analysis. For the lack of space we will consider as factors only four activities (M, A, D, I) and five dimensions (F, Q, P, B, S) of a project, and only some of the arguments related to the TM. By selecting activities and dimensions we obtain a generic multi-dimensional view of the TM arguments that relate to the factors under analysis. Table VII presents the view obtained from the arguments described in Table VI.

B. Ranking arguments, activities, and dimensions

We use the absolute ranking (H: high, M: medium, L: low and X: not relevant) for prioritizing the factors. As a first step, a work team needs to estimate how much each activity and dimension is relevant for the project. The weighing of activities and dimensions may be different depending on the project's context and nature. These rankings are used for filtering the set of arguments that shall be further analyzed from the multi-dimensional view of Table VII. In addition, the values of the rankings are used for multiplying the weights of the arguments. As result, the arguments that relate to the most prioritized activities and dimensions will produce more remarkable values in the contextual (i.e., quantified) view.

TABLE VII. A MULTI-DIMENSIONAL VIEW OF THE TM ARGUMENTS

	B	F	P	Q	S
A	6, 11, 22	11	6, 11	6, 11, 22	6, 22
D	6, 9, 11, 15, 22, 25, 29	1, 11, 29	6, 9, 11	1, 6, 9, 11, 15, 22, 24, 25, 29	6, 9, 15, 22, 25, 29
I	6, 9, 11, 15, 22, 25, 29	1, 11, 29	6, 9, 11	1, 6, 9, 11, 15, 22, 24, 25, 29	6, 9, 15, 22, 25, 29
M	2, 11	11	2, 11	11	2

As a second step, the work team needs to estimate how much each argument is relevant for the project. The ranking of the arguments generates the concrete quantified views. As a result of this step, the arguments are now contextualized and their weights may be calculated. Each argument is potentially the root of a tree of arguments (i.e. arguments that are part of its reasons, rebuttals, and alleviations). Therefore, the weight of an argument is the sum of its rank (H, M, or L) and the ranks of its children divided by the number of nodes in the arguments tree. The computation of a concrete view will be illustrated in the case study (Section VI).

C. Identifying critical factors

Weighing activities, dimensions, and arguments generates contextual views that are used to identify critical factors to the project, which correspond to view's cells that have remarkable values. The view's cells are prioritized based on their values. Firstly, we identify the most prioritized cell (i.e., with a priority of 1). Our approach suggests reasoning further about the factors that relate to this cell in order to nullify or reduce its value. We make the assumption that taking actions to address these most influent factors shall produce the highest benefits. After these critical factors are addressed, their ranking is adjusted. The adjusted rankings will provide new priorities. The user shall iterate these steps (i.e., identifying flaws and taking actions accordingly) until he is satisfied with the values in the concrete views (i.e., specific threshold values are attained).

VI. APPLICATION OF THE TM MULTI-DIMENSIONAL ANALYSIS TO A CONCRETE CONTEXT

We analyzed the TM design pattern in different contexts of application. In this section, we detail our experiment of applying the approach in the context of an undergraduate course of object-oriented software design at ETS.

A. Context of application of the TM

During the software design course the students are asked to design, implement, and document their projects' decisions in teams within given deadlines and using the Java programming language. The project we analyzed in this experiment focuses on the design and implementation of a software framework that provides the skeleton of a dice game. The design of a software framework promotes extensibility and reuse. In this case, the framework aims to provide a set of classes to allow the software implementation of various dice games. In addition, the project requires extending these classes to implement the functionality of a concrete dice game. At least three patterns are used in this project: Iterator, Template Method (TM), and Strategy.

The resulting Dice Game Software Framework (DGSF) was required to be simple enough to be understood by junior programmers that have backgrounds only in procedural programming. Students' teams were divided into two groups: teams proceeding without the analysis method and teams using the method.

B. Execution and results of the analysis

The DGSF promotes extensibility and reusability of basic high-quality components. Therefore, the arguments related to the design activity and quality dimension shall generate more remarkable values in the multi-dimensional view. Table VIII presents a contextualization of the factors (i.e., activities, dimensions, and arguments) for the DGSF. We quantified the ranking as H=100, M=10, L=1, and X=0. Table VIII indicates that we consider designing and quality as the most important factors for the project. The budget factor is not relevant for the project and the arguments that only relate to this dimension have been removed from the analysis. As a result of the rankings given in Table VIII, the weight of the first argument (Arg1) will be multiplied by ten thousand (10000 = 100 * 100) in the view's cell that intersects the design activity (H) and quality dimension (H) (i.e., Arg1 is part of this cell as shown in Table VII) while its weight will be multiplied by one thousand (1000 = 100 * 10) in the view's cell that intersects the design activity (H) and functions dimension (M). A total impact value is then computed for each cell of the view by summing the multiplied weights of the arguments it contains. These values are then translated into priorities (1 is the highest priority).

TABLE VIII. RANKING FACTORS FOR THE DGSF

Activities' rankings for the analysis	
Architecting	M
Designing	H
Implementing	M
Managing	L
Dimensions' rankings for the analysis	
Budget	X
Functions	M
People	M
Quality	H
Schedule	M

Arguments' rankings for each iteration			
Arg.	**Iter1**	**Iter2**	**Iter3**
1	L	L	L
2	H	H	H
6	M	L	X
7	L	X	X
9	H	H	H
11	X	X	X
15	L	L	L
22	H	L	X
24	X	X	X
25	H	X	X
29	X	X	X

TABLE IX. CONCRETE VIEWS OF THE DGSF ARGUMENTS

Activity	**Dimension**			
Iteration 1	**F**	**P**	**Q**	**S**
A	10	11	3	8
D	6	5	1	2
I	13	12	4	7
M	16	15	9	14
Iteration 2				
A	7	6	1	5
D	14	13	16	12
I	11	10	15	9
M	8	4	2	3

In the context of the DGSF, most students executed at least three iterations (i.e., identifying and addressing critical factors) in the analysis phase. Table VIII displays the weight of some arguments for the three iterations. For example, the argument (Arg1: "A final method cannot be overridden by subclasses") was ranked as low; this means that Arg1 is of low importance for the project. The evaluation is made in terms of the capacity of the DGSF to accomplish its functionalities (+F) and deliver quality (+Q). Students considered that the final mechanism provides a functional capability to the DGSF through the Java programming language. In addition, this feature contributes to improve the robustness of the DGSF and force subclass writers to reuse the fundamental components. However, its global impact on the functional and quality dimension is of low importance. As a second example, the arguments (Arg11) and (Arg24) do not apply for the DGSF.

Table IX presents the resulting multi-dimensional views for the DGSF for the two first iterations. As expected, the first iteration generated a high priority value (i.e., 1) for the designing and quality factors. Therefore, students took actions for addressing the most prioritized arguments that relate to these specific factors. Then, they adjusted the arguments' ranks in order to reflect the updated context. As a result, the impact values in the view are recomputed. The second iteration generated a high value for the architecting and quality factors as shown in Table IX. The reader shall also remark that the sum of each line (respectively column) provides a global indication of how much a particular activity (respectively dimension) and its related factors contribute to the project (the less is the value, the more important is the factor).

We compared the design reports of both the students' teams who didn't use the analysis method and the ones who used it. It is worth noting two results: 1) teams who used the method had elicited more arguments for supporting the design decisions that shaped their DGSF; 2) the number of changes they made to their DGSF for implementing a concrete dice game application was significantly reduced compared to teams who didn't use the method.

VII. RELATED WORKS

Designers deal with multiple factors that conflict depending on their software project. Many organizations maintain significant software artifacts and knowledge in a database to help the document control, development, and maintenance activities. In particular, much knowledge and support for the designers is provided by the literature including patterns, tactics and quality models (e.g. [1, 2, 3, 7, 8, 9, 10, 11, 12]). To take full advantage of the accumulated knowledge, the designers need frameworks and tools not only to manage this knowledge but also to relate it to the decisions taken and artifacts produced during the design activity. In particular, when evolving a system it is important not to lose sight of the initial objectives that drove the initial decisions. However, most of models, methods, and tools provide limited views into this knowledge database [3, 7, 8].

Our method was influenced by the ATAM (Architecture Tradeoff Analysis Method) [3], which aims at discovering risks and reporting correlations between decisions, quality attributes, and business objectives. However, our method is distinct: 1) the ATAM is based on tactics for supporting tradeoffs analysis; our method is based on inter-related factors for supporting multi-dimensional analysis; 2) the ATAM uses formatted scenarios for correlating decisions and objectives; our method uses formatted arguments for correlating factors; 3) the ATAM uses ranking for prioritizing scenarios; our method uses ranking for creating multi-dimensional views on networks of factors.

Many approaches were proposed to support the design process [2, 3, 7, 14], but few approaches (e.g. [8]) support the designers in managing and keeping track of the accumulated knowledge during the design process. One of the most used approaches is the Attribute-Driven Design method (ADD) [3]. The focus of ADD is the process of architecting systems in order to satisfy a set of quality attributes and to manage tradeoffs between these attributes (quality dimension). Our approach can be used to analyze and keep track of the artifacts and knowledge produced by the ADD, but it also provides additional capacities for analyzing multiple dimensions and encoding decisions. In particular, many architectural styles and patterns have been described and cataloged [1, 2, 3] in the literature, but few approaches support the designers in selecting and using the appropriate patterns. We believe our approach can be used to describe the artifacts in a manner that may ease the selection of appropriate tactics and patterns and keep track of the selected artifacts as quantified design decisions.

Finally, our work is closely related to Ovaska *et al.*'s work [8]. They propose an approach to fully integrate quality requirements in the software design process. Their approach allows the architect to manage and track the quality attributes from the requirements specification to the architecture design. This approach focuses on finding styles and patterns using some quality attributes. While this is very useful, an architect still needs to keep track of the rational, objectives and other constraints that led the choice of these quality attributes. The management of the relationships between the decisions and the factors they influence is a challenge [11]. Our framework can be useful to manage these relationships into a structured view that relates in a finer-grained manner the artifacts of the problem space to the ones of the solution space, from the organizational goals to the specific system artifacts. We believe a multi-dimensional view is a valuable artifact for providing an integrated view of the knowledge.

VIII. CONCLUSION

In this paper, a multi-dimensional analysis approach for analyzing artifacts such as design patterns is presented. The approach describes artifacts using a set of factors of influence such as features, issues, arguments, activities, and dimensions. Relating these factors enabled the creation of multi-dimensional views that support designers in identifying the critical factors to their projects and addressing them.

The approach was used within a context of an undergraduate course project especially for analyzing and addressing the factors related the TM design pattern. As a proof of concept a prototype tool that students used for analysis has been developed. This experimentation proved that the multi-dimensional view eases design decision making and helps keeping track of these decisions.

This preliminary work produced evidences that the multi-dimensional analysis approach is a valuable step towards handling artifacts as an integrated set of factors of influence. The proposed approach can be customized to better support particular development process and systems' needs. In the near future the approach will be applied on other artifacts such as architectural patterns. In addition, the arguments' characteristics will be analyzed to define a set of guidelines that will support the process of eliciting arguments. One goal of this work is to contribute to building a reference model of factors linked formally and exploited by algorithms.

REFERENCE

[1] Gamma, E., Helm, R., Johnson, R., Vlissides, J., "Design Patterns: Elements of Reusable Object-Oriented Software", A.-W., B. (1995)

[2] Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Nord, R., Stafford, J., "Documenting Software Architectures – Views and Beyond", Addison Wesley, Boston (2003)

[3] Bass, L., Clements, P., Kazman, R., "Software Architecture in Practice", Addison Wesley, Boston (2003)

[4] Carbogim, D, Robertson, D, Lee, J, "Argument-based applications to knowledge engineering", *Knowl. Eng. Rev.* 15, 2 (J. 2000), 119-149.

[5] Carlos, C, Jarred, M, Sanjay, M, Iyad, R, Chris, R, Guillermo, S, Matthew, S, Gerard, V, Steven, W, "Towards an argument interchange format", *Knowl. Eng. Rev.* 21, 4 (Dec. 2006), 293-316.

[6] Karl E. Wiegers, "Standing on Principle," *Journal of the Quality Assurance Institute*, vol. 11, no. 3 (July 1997).

[7] Kim, S., Kim, D.K., Lu, L., Park, S., "Quality‐driven Architecture Development Using Architectural Tactics", Journal of Systems and Software 82, 1211‐1231 (2009)

[8] Ovaska, E., Evesti, A., Henttonen, K., Palviainen, M., Aho, P., "Knowledge Based Quality-driven Architecture Design and Evaluation", Journal of Info. and Soft. Tech. 52, 577-601 (2010)

[9] Fayad, M., Schmidt, D., Johnson, R., "Building Application Frameworks: Object-Oriented Foundations of Framework Design", Addison-Wesley, Boston (1999)

[10] Shahin, M., Liang, P., Khayyambashi, M.R., "Architectural Design Decision: Existing Models and Tools", In: WICSA/ECSA 2009, pp. 293-296. IEEE, Cambridge (2009)

[11] Khaled, L., "Achieving Goals through Architectural Design Decisions", Journal of Computer Science. 6, 1424--1429 (2010)

[12] Parizi, R.M., Ghani, A., "Architectural Knowledge Sharing (AKS) Approaches: a Survey Research", Journal of Theoretical and Applied Information Technology, 1224--1235 (2008)

[13] Scott, J., Kazman, R., "Realizing and Refining Architectural Tactics: Availability", Technical report, Software Engineering Institute (2009)

[14] Bachmann, F., Bass, L., Klein, M., "Deriving Architectural Tactics: a Step Toward Methodical Architectural Design", Technical report, Software Engineering Institute (2003)

[15] Mylopoulos, J., Chung, L., Nixon, B., "Representing and Using Nonfunctional Requirements: A Process-Oriented Approach", *IEEE Trans. on Sofware. Eng.*, Vol. 18 No. 6, June 1992, pp. 483-497.

[16] L. Chung, B. Nixon, E. Yu and J. Mylopoulos, "*Non-functional requirements in software engineering*" Kluwer Acad., Boston, 2000.

[17] D. Gross and E. Yu, "From Non-Functional Requirements to Design Through Patterns", *Requirements Eng. Journal* Vol. 6, 2001, 18-36.

[18] S. Toulmin, "The Uses of Argument", Cambridge, C. U. Press, 1958.

Semantic Conflicts Detection in Model-driven Engineering

Valéria Oliveira Costa[1,2], João M. B. Oliveira Junior[1], Leonardo Gresta Paulino Murta[2]

[1]Instituto Federal de Educação, Ciência e Tecnologia do Piauí, IFPI
Teresina, Brasil
valeria@ifpi.edu.br
joaomanoel@aluno.ifpi.edu.br

[2] Instituto de Computação
Universidade Federal Fluminense, UFF
Niterói, Brasil
leomurta@ic.uff.br

***Abstract*— An important challenge of Model Driven Version Control System (VCS) is to use conflict detection methods that are appropriate for models. Methods that analyze only the syntax of models can detect conflicts that do not exist in reality (false positives) and can fail to detect conflicts that do exist (false negatives). This paper presents a method to reduce the occurrence of both false positive and false negative conflicts. For this, the presented method provides an analyzer of the semantic equivalence between models. Our method verifies if the model versions are semantically equivalent, if one version semantically contains the other version, and if there are conflicts between versions.**

Keywords- model-based version control; semantic conflict detection

I. INTRODUCTION

With the advent of the Model-driven Engineering (MDE), which aims to facilitate the development of systems through the creation, manipulation, and maintenance of models, it became possible to direct the focus of the developers to design applications at higher abstraction levels [1]. Thus, a system can be constructed through the refinement of models that begins at the highest level of abstraction and goes toward the lower levels through the use of transformations [1].

In the context of MDE, during the development or evolution of a system, multiple versions of a model can be generated. Similarly to source code, this brings the necessity of model-driven Version Control Systems (VCS). A VCS helps the development team to manage the evolution of a software product through consistent maintenance of its many variants and revisions [2]. Therefore, a model-driven VCS manages model version. To do so, it compares models, detects and resolves conflicts, and makes the consistent merge of models. Among the existing Model driven VCS we can mention Odyssey-SCM [3], Smover [4], and Mirador [5].

One of the key concepts in the area of version control is the conflict. Conflict is a set of contradictory changes where at least one operation performed by the first developer does not agree with at least one operation performed by a subsequent developer [6]. A conflict is not desirable because it generates an additional effort for the developer [7] , which means rework. One might think that the time and effort needed to resolve a conflict could be used to continue the development or evolution of the system.

In this scenario, it may be noted that one important challenge of the model-driven VCS is the use of a conflict detection method that is appropriate and efficient to models. According to [8], in order to obtain success in a model merge process, it is necessary to understand not only the logical structure of the model, but also its semantics. According to [9], to resolve the conflicts, it is needed to identify the reasons of the conflict. This is especially difficult when only syntactic detection support is used.

In a modeling process, there may be situations where the same intention can be modeled in different ways. Thus, two developers working in parallel may use different strategies to model the same situation. A purely syntactic analyzer identifies this difference as conflicting. After a manual analysis, it can be verified that the conflict does not proceed, since the two representations are semantically equivalent. This type of conflict is false positive and reduces the efficacy of the conflict detection method, since the method should not report erroneous information to the developer.

One solution to reduce occurrences of false positive conflicts is to understand the semantics of the models. This understanding allows the identification of related syntactic conflict that is actually a semantic equivalence. Furthermore, a semantic analyzer also allows it to detect semantic conflicts. These conflicts occur when modifications in a given model element interfere in another model element even without explicit syntactic relationships among them. Semantic conflicts are more difficult to detect and, because of it, they generate false negatives conflicts. These, in turn, are conflicts that exist in reality, but unfortunately the conflict detection method cannot diagnose them.

This way, a good method of conflict detection should be able to identify semantic equivalences and not report them as

conflicts. This contributes to the reduction of false positive conflicts. Also, it must be able to detect semantic conflicts, thereby decreasing the number of undetected conflicts or false negatives conflicts.

To help solve the problems related above, this paper presents a semantic conflict detection method for models. The method focuses on the investigation of semantic equivalence of models in order to reduce false positive conflicts. It also uses the semantic understanding of the models to increase the coverage of the conflict detection method. The increased coverage decreases the likelihood of a semantic conflict undetected by the method (false positives).

The rest of the paper is organized as follows: section II presents important concepts about model-based version control; Section III explains the proposed method; Section IV shows an example of our method in action; Section V discusses the technologies used in the implementation of the prototype; Section VI presents some related works; and Section VII presents the conclusion and future work.

II. BACKGROUND

Consider a scenario where the development team uses an optimistic VCS. With this type of VCS, each developer can work in an individual copy separately [2], until they decide to socialize their copy with other developers. Initially, the original version of the project, called *base version*, is stored in the repository. Then, all team members download the *base version* and start working on it separately. During socialization, the developer´s version, called *developer version*, should be analyzed and compared with the *base version*, and with the latest version already committed into the repository, called *current version*. The Fig. 1 shows this scenario. The process that considers the information contained in the ancestral version for calculating differences between two versions is used in three-way merge [2].

Another concept used in this work is state-based merge [2]. In this type of merge, only the information contained in the *base version* and its revisions are considered [2]. In this type of merge, there are no records of the operations performed that facilitate the understanding of the transformation from one version to another. On the other hand, this type of merge is more realistic, as it imposes no restriction over the development environment. The complete state-based merge process is composed of four phases [9]: comparison, conflict detection, conflict resolution, and the merge itself.

III. SEMANTIC CONFLICTS DETECTION OF MODELS

This work is based on the before-mentioned state-based merge process. However, as it focuses only on the identification of semantic conflicts, it is restricted to the first two phases: comparison and conflict detection.

In the comparison phase, our method receives the three versions to be analyzed (*base*, *current* and *developer* versions). The versions are automatically transformed into a set of Prolog facts. Then, each Prolog version is analyzed in order to infer indirect relationships. This step is done by a Prolog set of rules that describes the semantic relationships of models according to

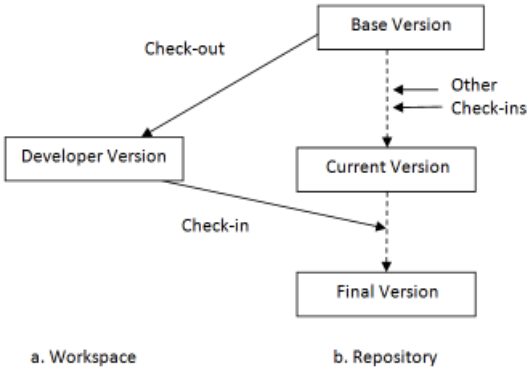


Figure 1. Development scenario

a specific metamodel. The versions are then compared to verify if they are semantically equivalent or if one version semantically contains the other. If the versions are not equivalent, and if the *current* version does not contain the *developer* version (or vice versa), then the conflict detection is initiated. In this phase, four sets are created considering the *base*, *current*, and *developer* versions: all model elements added to *base* version to compose *current* and *developer* versions and all model elements deleted from *base* version to compose *current* and *developer* versions. The special intersection among the sets of added and deleted model elements of different versions (*current* and *developer*) indicates conflicts.

The Fig. 2 provides an overview of the proposed method. In the first activity, called Translation, the *developer* version represents a version just produced by a member of the development team as a revision of *base* version. On the other hand, *current* version is the tip of the repository, created due to a previous commit performed by other team members. This way, *current* and *developer* versions were created in parallel, and both are revisions of *base* version. The goal of this activity is to transform these three versions into Prolog facts. Each fact refers to an existing element or relationship in the model. In the case of a relationship, there is involvement of a pair of elements in general. This way, the Translate activity result is a set of Prolog facts that represents all relevant syntactic information contained in the analyzed model.

The second activity, called Semantic Enrichment, is responsible for inferring new facts. To do so, this activity combines the previously generated facts with Metamodel-specific Rules. The Metamodel-specific Rules represent the semantics of relationships in a given metamodel. This way, they need to be set once and can be used for every model compliant to the metamodel. For instance, in the context of Use Case diagrams (UCD) of the UML metamodel, Tab. I shows the semantic rules used by our method. Such rules help on extracting semantics from the syntactic set of model elements represented as Prolog facts.

TABLE I. USE CASE DIAGRAM SEMANTIC RULES

Semantic rules	
#	Rule
1	$\forall a, b \in Actor, \forall c \in UseCase,$ $inheritance(a,b) \wedge association(b,c) \Rightarrow association(a,c)$
2	$\forall a \in Actor, \forall b, c \in Use\ Case,$ $inheritance(b,c) \wedge association(a,c) \Rightarrow association(a,b)$
3	$\forall a \in Actor, \forall b, c \in Use\ Case,$ $include(b,c) \wedge association(a,b) \Rightarrow association(a,c)$
4	$\forall a \in Actor, \forall b, c \in Use\ Case,$ $extend(b,c) \wedge association(a,c) \Rightarrow association(a,b)$

Next, the third activity, called Conflict Detection is performed. In this activity, the enriched Prolog facts of *current* and *developer* versions are analyzed and compared to *base* version. The analysis is done based on the three-way diff concept, forming two pairs: the first involving the diff between *base* and *current* versions and the second involving the diff between *base* and *developer* versions. For each pair, two sets of differences are computed: one that holds all added items (Add) and another that holds all deleted items (Del). These sets are computed according to (1) and (2):

$$Add_v = v \setminus base \quad (1)$$

$$Del_v = base \setminus v \quad (2)$$

Where $v \in \{current, developer\}$. After the computation of the additions and deletions sets, a conflict is detected if an element of the model appears simultaneously in the set of additions of the first pair and in the set of deletions of the second pair or vice versa. It is important to emphasize that to generate the set of semantic conflicts, the Metamodel-specific Rules were previously used during the Semantic Enrichment activity. Equation (3) denotes how the set of conflicts is formed:

$$Conflict_{v1,v2} = (Add_{v1} \cap^* Del_{v2}) \cup (Add_{v2} \cap^* Del_{v1}) \quad (3)$$

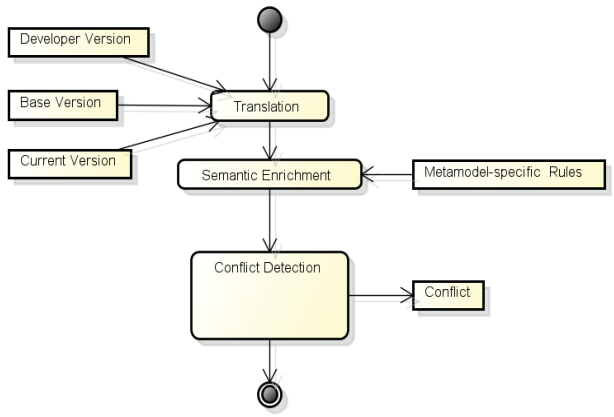


Figure 2. Our method overview

Where $v1$ and $v2$ can be any two variants in general, but in the specific case of this work, $v1$ = *current* and $v2$ = *developer*. Moreover, $\cap^*$ represents an especial intersection between sets that overloads the equality property to match not only identical elements. The matching between non-identical elements is possible because our method takes into account the syntactic rules of relationships described in the UCD metamodel. When a relationship is used, not only the relationship, but also the elements that compose it are verified. So, if an actor is deleted by $v1$, while $v2$ adds a relationship that uses the same actor (e.g. an association between this actor and a use case), a conflict is detected. This reasoning is analogous to the use of other relationships of the UCD metamodel.

The analysis of the $v1$, $v2$ and Conflict sets may lead to the following conclusions:

- If $v1 = v2$ then the versions are semantically equivalent. This means that the intentions of the developers were similar. In this case, there is no conflict and any one of the two versions can be chosen.
- If $(v1 \subset v2) \vee (v2 \subset v1)$ then one version semantically contains the other. In this case, there is also no occurrence of conflict and the intention of one developer entails the intention of the other. Since there is no divergence of intention, the most complete version should be chosen.
- If $(v1 \neq v2) \wedge \neg\big((v1 \subset v2) \vee (v2 \subset v1)\big) \wedge Conflict_{v1,v2} = \emptyset$ then the versions differ among themselves, one version semantically doesn´t contain the other but there is no semantic conflict. In this case, a syntactic merge suffices.
- If $Conflict_{v1,v2} \neq \emptyset$ then the versions semantically differ, one version semantically doesn´t contain the other and there are semantic conflicts. In this case, the Conflict set contains the syntactic facts that are implying semantic conflicts.

In order to have a better understanding of our method, section IV shows it in action.

IV. OUR METHOD IN ACTION

Consider the UCD of a bank control system as depicted in Fig. 3, where Fig. 3.a shows the *base* version, Fig. 3.b shows the *current* version and Fig 3.c shows the *developer* version. These last two versions are revision of the *base* version.

The three versions have their files submitted to Translation activity, as explained in the previous section. The results produced in this activity are shown in Tab II. In this table, each column shows the automatically generated facts for each of the presented versions.

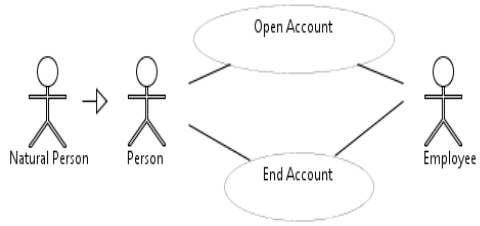
Then the Semantic Enrichment activity is performed through the application of previously defined semantic rules. These rules, written in Prolog, discover new facts created through the indirect relationships among model elements, as shown in Tab. III.

To illustrate the second activity of Fig. 2, consider the diagram presented in Fig. 3.a. Note that the Natural Person actor is indirectly associated with the Open Account use case. The association occurs through inheritance between Natural Person and Person actors. Considering the semantics of

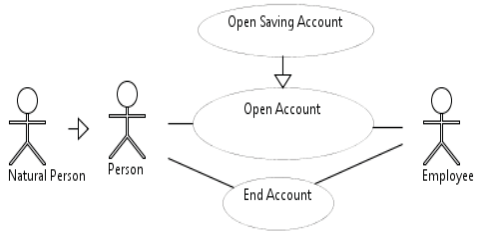
inheritance, it can be said that Natural Person is a Person. Furthermore, as Person is directly associated to the Open Account and End Account use cases, we can say that Natural Person is also associated with Open Account and End Account use cases, although the association is not directly shown in the model.

The discovery of the indirect relationships is a key factor in order to enable the detection of semantic conflicts in our method. The knowledge of those relationships allows us to verify that a change made to an element X generates conflict in an element Y, even if X and Y are not directly connected in the model.

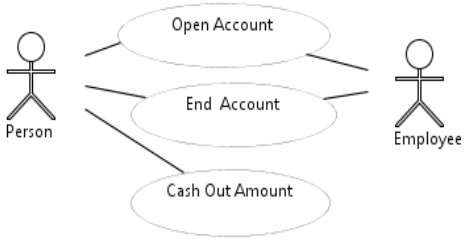
Tab. III shows the results of the second activity of Fig. 2 applied to the case shown in Fig. 3.



a. *base version*



b. *current version*



c. *developer version*

Figure 3. Model versions of a bank control system

The discovered facts are added to the files shown in Tab. II. Consider the *base* version column of Tab. III. The first Prolog fact of this column is association(natural_person, open_account). This fact can be visually inferred in Fig. 3.a where Natural Person actor is connected through inheritance with Person actor, which is associated with Open Account use case.

The third activity, Conflict Detection, computes the following sets:

$Del_{current} = \emptyset$

$Add_{current}$ = { usecase(open_saving_account),
inheritance(open_saving_account, open_account),
association(natural_person, open_saving_account),
association(person, open_saving_account),
association(employee, open_saving_account)}

$Del_{developer}$ = {actor(natural_person),
inheritance(natural_person, person),
association(natural_person, open_account),
association(natural_person, end_account)}

$Add_{developer}$ = { usecase(cash_out_amount),
association(person, cash_out_amount)}

Next, the intersection between the elements of the sets is calculated to detect occurrence of conflicts:

$Add_{current} \cap^* Del_{developer}$ = { actor(natural_person),
association(natural_person, open_saving_account) }

$Add_{developer} \cap^* Del_{current} = \emptyset$

$Conflict_{current, developer}$ = { actor(natural_person),
association(natural_person, open_saving_account)}

It can be observed that actor(natural_person) and association(natural_person, open_saving_account) operate over with natural_person actor. This intersection represents a conflict because an association requires the existence of both model elements in its association ends. According to Tab. II, natural_person is an actor. Moreover, according to $Del_{developer}$, this actor was deleted in *developer* version. Meanwhile, according to $Add_{current}$, a new association has been established in parallel with this actor in *current* version. This scenario shows that, in the same team, a developer is deleting an actor while another is expanded its responsibilities in the same system design. This situation indicates a semantic conflict between the versions. This type of conflict is not detected when only syntactic elements and direct relationships are analyzed, leading to false negatives.

As shown in the example, the presented method contributes to detect these conflicts by increasing the efficiency of the detect conflict method. Moreover, the computation of semantic equivalences identifies the use of different relationships that have the same meaning. This is

TABLE II. MODEL TO PROLOG TRANSLATION.

Contents of prolog files – activity 1		
***base* version**	***current* version**	***developer* version**
actor(person).	actor(person).	actor(person).
actor(natural_person).	actor(natural_person).	
actor(employee).	actor(employee).	actor(employee).
usecase(open_account).	usecase(open_account).	usecase(open_account).
usecase(end_account).	usecase(end_account).	usecase(end_account).
	usecase(open_saving_account).	
		usecase(cash_out_amount).
inheritance(natural_person, person).	inheritance(natural_person, person).	
	inheritance(open_saving_account, open_account).	
association(person, open_account).	association(person, open_account).	association(person, open_account).
association(person, end_account).	association(person, end_account).	association(person, end_account).
association(employee, open_account).	association(employee, open_account).	association(employee, open_account).
association(employee, end_account).	association(employee, end_account).	association(employee, end_account).
		association(person, cash_out_amount).

TABLE III. SEMANTIC ENRICHMENT OF PROLOG FACTS

Prolog facts added by semantic rules application		
***base* version**	***current* version**	***developer* version**
association(natural_ person,open_account).	association(natural_ person,open_account).	
association(natural_ person, end_account).	association(natural_ person, end_account).	
	association(natural_person, open_saving_account).	
	association(person, open_saving_account).	
	association(employee, open_saving_account).	

possible because semantic rules abstract the syntactic differences of relationships and extract the meaning of such relationships in the model as a whole. Thus, the method does not identify these differences as conflicts and reduces conflicts false positives reported to developer. This feature reduces the rework generated for the team.

V. PROTOTYPE IMPLEMENTATION

We are implementing a prototype of our method for UCD. Currently, the prototype entails activities related to comparison and conflict detection phases. We are testing these phases and we intend to extend our prototype in the near future to other UML diagrams.

For the purpose of testing our prototype, we adopt Papyrus[a] to design use case models. This tool provides an editing environment for EMF and UML models, among others. For each model, Papyrus generates two important files: a diagram interchange file, which contains the diagram information such as position of elements, and an XML Metadata Interchange (XMI) file, which contains the model elements themselves.

Each XMI file is submitted to Translation activity to be automatically transformed into a set of Prolog facts. The model´s transformation to Prolog facts is made using the OMG Model to Text (M2T) standard. The implementation of the transformation is based on Acceleo[b]. Acceleo is a generator that transforms models into code. It uses Model-driven Architecture (MDA) to transform a model into text. To perform the second activity, Semantic Enrichment, we adopt the TuProlog[c] library integrated with Java.

Our method was conceived to accommodate new types of diagrams and metamodels. As previously discussed, it is not restricted to proprietary model formats as input models are XMI files. The support for a new diagram or even a new metamodel requires three main tasks: writing a M2T transformation to generate Prolog facts according to the new diagram or metamodel, writing Prolog rules for the Semantic Enrichment phase and writing syntactic rules in Java to be used in Conflict Detection phase .

VI. RELATED WORK

In [10], a semantic conflict detection method is presented, named Smover. The approach is based on semantic views of interest and inspection strategies of elements that can be configured by the user. A semantic view maps a metamodel to another based on relevant aspects of the first. The output of the transformation is a model in conformity with the second that contains the aspects of interest. The conflicts found in the original metamodel are syntactic conflicts, and those found the mapped metamodel are semantic conflicts. Our method transforms the elements and relationships into Prolog facts and uses inference rules to help compare two models. Moreover, we use only one metamodel to detect conflicts.

Odyssey-VCS [11] is a Model-driven VCS that allows the use of fine granularity for version UML 2 models. The conflict detection is based in existence analysis of elements and processing of attributes and relationships. It considers both non containment and containment relationships. Our method focuses in semantic conflict detection to all elements and relationships of the UCD models.

[a] http://www.eclipse.org/papyrus/

[b] http://www.eclipse.org/acceleo/

[c] http://tuprolog.alice.unibo.it/

Gerth et al. [12] present a method to detect conflicts that takes into account the semantics of business process models. This method decomposes the process model into fragments and activities to make your comparison. Moreover, it creates add, delete and move operations for fragments and activities. The approach also provides a method to the resolution of conflicts and uses individual strategies to resolve different types of conflicts. The method uses change-stated merge. Our work presents a method to state-based approach to UCD and in the future other UML models. We also intend to automatically resolve detected conflicts.

Koegel et al. [13] provide an algorithm to compute conflicts on the operations that change the model. It also takes into account the serialization of the application of these operations. The conflicts are classified into hard and soft. The hard conflicts must be resolved by the user and the soft ones automatically resolved. However, different of our method it does not take into account the semantics of the models and is made for operation-based approach.

Mirador [5] uses a hybrid state and operation-based approach. The merge is based on operation and detects direct and indirect conflicts. Conflicts are detected by the before(a,b) predicate where an operation a must come before an operation b. The approach describes techniques for detection and resolution of conflicts based on decision tables. The users can customize the rules of the tables. These tables can take into account the semantics and to use their rules to detect false positives. The approach uses metametamodel Ecore extended to compute differences between versions. Our method uses inference rules of metamodel to help compute semantic conflicts. It considers not only the false positive conflict detection generated by similar situations but also false negative conflict detection.

VII. CONCLUSION

This paper presented a conflict detect method to MDE. The method expects three model versions as input (two variants with a common ancestry) and verifies: if the variants are semantically equivalent, if one variant semantically contains the other, and if there are semantic conflicts to be resolved. The process starts by transforming the models into Prolog facts. The Prolog facts are semantically enriched by means of metamodel-specific rules. Finally, semantic conflicts are discovered via three-way diff technique.

We also present an example that shows how changes in different elements can interfere with other elements, even if they are not directly connected. Due to the difficulty in identifying this type of conflict, the method helps on reducing the amount of false negatives conflict. Thus, it increases the efficacy of the conflict detection method as a whole.

Moreover, the detection of semantic equivalence decreases the amount of false positives conflicts reported to the developer, whereas purely syntactical analysis detects differences and reports them as conflicts. This feature also contributes to the improvement of conflict detection method because it reduces the rework of the team.

Currently, we are studying how to make the merge when there are no conflicts are detected. In the case of the equivalent models, the system must choose or suggest which model should be considered the merged version. To help on this suggestion, the traceability of indirect relationships should be considered. The traceability can indicate the best model designed.

As future work, we intend to support automatic conflict resolution and collaborative merge. At the phase of conflict resolution, heuristics may help on suggesting consistent solutions. Regarding collaborative merge, traceability can also figure as an important technique, visually guiding developers from semantic conflicts to the syntactic elements that triggered these conflicts.

We also intend to support additional UML diagrams and expand the method to work directly on the metametamodeling language, such as EMF and MOF, via the reflective API. This would allow processing any metamodel, requiring only the metamodel XMI file as input. Finally, we are planning to run some experimental studies with the proposed method.

VIII. REFERENCES

[1] A. Cicchetti, F. Ciccozzi, e T. Leveque, "On the concurrent Versioning of Metamodels and Models: Challenges and possible Solutions", 2011, p. 16–25.

[2] T. Mens, "A state-of-the-art survey on software merging", *Software Engineering, IEEE Transactions on*, vol. 28, nº 5, p. 449–462, 2002.

[3] L. Murta, H. Oliveira, C. Dantas, L. G. Lopes, e C. Werner, "Odyssey-SCM: An integrated software configuration management infrastructure for UML models", presented at the Science of Computer Programming, 2007, vol. 65, p. 249–274.

[4] K. Altmanninger, "Model Versioning – SMoVer", *Smover: Configurable & Semantically Enhanced Conflict Detection in Model Version*, 2011. [Online]. Available: http://smover.tk.uni-linz.ac.at/prototype.php. [Accessed: 20-dez-2011].

[5] S. Barrett, P. Chalin, e G. Butler, "Table-driven detection and resolution of operation-based merge conflicts with mirador", *Modelling Foundations and Applications*, p. 329–344, 2011.

[6] R. Conradi e B. Westfechtel, "Version Models for Software Configuration Management", *ACM Computing Surveys (CSUR)*, vol. 30, nº 2, p. 232–282, 1998.

[7] J. G. Prudêncio, L. Murta, C. Werner, e R. da S. V. Cepêda, "To lock, or not to lock: That is the question.", *Journal of Systems and Software*, vol. 85, nº 2, p. 277–289, 2012.

[8] K. Altmanninger e G. Kotsis, "Towards accurate conflict detection in a VCS for model artifacts: a comparison of two semantically enhanced approaches", 2009, p. 139–146.

[9] K. Altmanninger, M. Seidl, e M. Wimmer, "A survey on model versioning approaches", presented at the International Journal of Web Information Systems, 2009, vol. 5, p. 271–304.

[10] K. Altmanninger, W. Schwinger, e G. Kotsis, "Semantics for accurate conflict detection in smover specification detection and presentation by example", *IJEIS*, p. 68–84, 2010.

[11] L. Murta, C. Corrêa, J. G. Prudêncio, e C. Werner, "Towards Odyssey-VCS 2: Improvements over a UML-based Version Control System", presented at the ACM, Leipzig, Germany New York, USA, 2008, p. 25–30.

[12] C. Gerth, J. M. Küster, M. Luckey, e G. Engels, "Detection and resolution of conflicting change operations in version management of process models", *Software & Systems Modeling*, dez. 2011.

[13] M. Koegel, M. Herrmannsdoerfer,, e O. von Wesendonk, "Operation Base Conflict Detection", presented at the IWMCP10: International Workshop on Model Comparison in Practice, Malaga, Spain, 2010.

Automatic Generation of Semantic Web Services

Thiago P. da Silva, Thais Batista, Frederico Lopes
DIMAp
Universidade Federal do Rio Grande do Norte - UFRN
Natal, RN, Brazil
{thiagosilva.inf, thaisbatista, fred.lopes}@gmail.com

Flavia C. Delicato, Paulo F. Pires
DCC/IM
Universidade Federal do Rio de Janeiro - UFRJ
Rio de Janeiro, RJ, Brazil
{fdelicato, paulo.f.pires}@gmail.com

***Abstract*—Web services typically contain only syntactic information describing their interfaces. Due to the lack of semantic descriptions, service composition becomes a difficult task. To solve this problem, Web services can exploit the use of ontologies for the semantic definition of service's interface, thus facilitating the automation of discovering, publication, mediation, invocation, and composition of services. However, ontology languages, such as OWL-S, have constructs that are not easy to understand, even for Web developers, and the existing tools that support their use contains many details that make them difficult to manipulate. This paper presents a MDD tool called AutoWebS (Automatic Generation of Semantic Web Services) to develop OWL-S semantic Web services from UML models. AutoWebS offers an environment that provides many features required to model, implement, compile, and deploy semantic Web services.**

Keywords-Model Driven Development; OWL; UML; Web Service; Semantic Web

I. INTRODUCTION

The semantic Web services (SWS) [1] encompasses the semantic definition of services through the use of ontologies. Ontologies provide a semantic description of a Web service (WS) that is computationally interpretable by combining the concepts defined in the ontology with the elements of the WS syntactically described in WSDL [2]. The development of SWS is typically divided into two stages: (i) WS creation, and (ii) WS ontology creation. There are languages that allow the description of WS using ontologies, such as OWL-S (*Ontology Language for Web Services*) [3]. The ontology languages have different syntaxes, a very extensive vocabulary, and most of them are based on first order logic.

Existing tools that support the use of ontology languages do not offer mechanisms to abstract their syntaxes, thus hampering their use [4]. The adoption of semantic descriptions of WS is hindered by the limitations of the tools and the fact that creating an ontology is a difficult and time-consuming task [5]. In order to facilitate the use of SWS it is necessary to abstract specific details for each semantic description language.

A way to guide the development of tools that meet this purpose is to define the essential requirements for such tools. In this sense, [6] proposed some essential requirements for the development of tools to compose WS. These requirements can be adapted to guide the development of a high level tool that provides an easy way to create atomic SWS. The requirements should define the need of abstracting the underlying technologies used in the development of SWS. Moreover, it is necessary to automate the generation of code artifacts, since it is expected to abstract the languages involved in the creation of SWS. The requirements should also specify the need of integrating functionalities to allow the creation of SWS without the need of accessing external resources or tools. This is essential to reduce the development time, to avoid errors and possible conflicts that arise when using different tools/applications such as conflicts of languages versions.

In this context, Model Driven Development (MDD) [7] is useful to manage the inherent complexity in the use of ontologies to specify SWS. MDD is a software development approach that focuses on creating models instead of program code, allowing separation of concerns between specification and implementation. Thus, a MDD approach can abstract away the underlying technologies of WS through models.

In the MDD approach, UML profiles can be used to create models that provide a higher abstraction level of the underlying technologies of SWS. A UML profile consists of a collection of extensions that customize UML for a specific domain and, models created by UML profiles are valid UML models, which can be created using the same tools for UML modeling. UML and ontology specification languages have some overlaps and similarities. For example, both languages use classes, associations between classes, class properties, generalizations, and data types for structural representation of a software system. Such similarities make it possible to make some elements of ontology specification languages in elements of the UML model [8]. Other elements of the ontology specification languages that do not directly correspond to the primitive elements of UML can be represented by using UML profiles. Thus, UML profiles can be used for specifying SWS, since they are capable to represent ontologies and WS interface.

This paper proposes a UML profile for the SWS and presents AutoWebS (**Auto**matic Generation of Semantic **Web S**ervices), a MDD tool to create SWS. AutoWebS offers a graphical environment, which can be used to graphically represent OWL (*Web Ontology Language*) [9] ontologies as UML models, using the elements defined in a UML profile. Thus, it avoids that developers directly deal with the OWL language syntax by supporting the definition of SWS interface via UML models. This tool enables to: (i) specify the WS interface, i.e., model the inputs and outputs of each WS operation, (ii) perform semantic annotations, linking the inputs and outputs of operations with the elements of an ontology, (iii) automatically create the OWL-S file that contains the semantic

description of WS, (iv) automatically generate the skeleton code for the WS, (v) extend its functionality, for example, including support for another semantic description language. The tool is implemented as a plugin of the Eclipse platform and uses EMF (*Eclipse Modeling Framework*) for the specification of the OWL and OWL-S languages metamodels. AutoWebS uses the Axis2 middleware [10] for generating WS and it also uses syntax validators for OWL-S ontologies.

This paper is organized as follows. Section 2 presents the requirements for a tool to create SWS, an overview of AutoWebS, and a motivating example. Section 3 presents the implementation details. Section 4 presents the results of a controlled experiment that evaluates AutoWebS compared to an application suite composed by the OWL-S Editor [11] and the Axis2 Eclipse plugin. Section 5 presents related work. Section 6 contains the conclusions and future perspectives.

II. AUTOWEBS

This section presents the essential requirements for a tool to support the creation of SWS, details of the AutoWebS, and a motivating example.

A. Requirements

Before the development of AutoWebS, a set of essential requirements for a tool aimed at creating SWS have been established. The requirements are intended to specify the need of abstracting specific details and the syntax of semantic description languages used in the creation of the semantic description of WS. The considered requirements are: (i) **R0** To provide a mechanism for a user to model the SWS interface without the needing to have a deep knowledge about Web technologies; (ii) **R1** To perform the automatic generation of some WS source code; (iii) **R2** To perform the automatic generation of the WS semantic description in a semantic language; (iv) **R3** To allow the use of pre-existing ontologies to create semantic description of WS through the interconnection of the concepts defined in these ontologies with the elements of the WS; (v) **R4** To provide a development environment that integrates all the features required to create a SWS, without the need to use external tools or resources; (vi) **R5** To generate syntactically correct code artifacts. The generated code should be readable, executable, and meet the W3C specification for WS and SWS; (vii) **R6** To allow the manutenability e.g., allow the insertion of new features without the need of performing major structural modifications in the tool.

R0 requirement can be achieved using UML models to specify the interface of the SWS, and to abstract the underlying technologies used to develop SWS. **R1** and **R2** requirements are related to automating the generation of code artifacts. In this work we adopt the OWL language because this work is part of a wider project which uses OWL. **R3** requirement is concerned with the use of pre-existing ontologies to create semantic description of WS. This issue contributes to promote interoperability since it enables the use of existing ontologies that can be widely known and available on the Internet.

R4 requirement is essential to reduce the time of developing SWS and avoid possible conflicts that arise when different tools are used to build a SWS. For example, when using a tool to create the domain ontology that adopts a particular version of the ontology specification language that is not supported by other tool used to create the WS ontology. **R5** aims to ensure the correctness of code artifacts generated by the tool. **R6** is concerned with the insertion of new features or update of the current features, such as the upgrade to a newer version of the semantic language used to describe WS.

B. Tool Overview

AutoWebS provides an environment that integrates various functionalities used to create SWS (**R4** and **R6**). The use of AutoWebS, as shown in Figure 1, consists of three main activities: (a) import domain ontology, (b) design the WS, and (c) generate the OWL-S ontology and WS. The tool requires as input OWL ontologies (**R3**) and produces as outputs OWL-S ontologies (**R2**) and the source code of the WS in Java (**R1**).

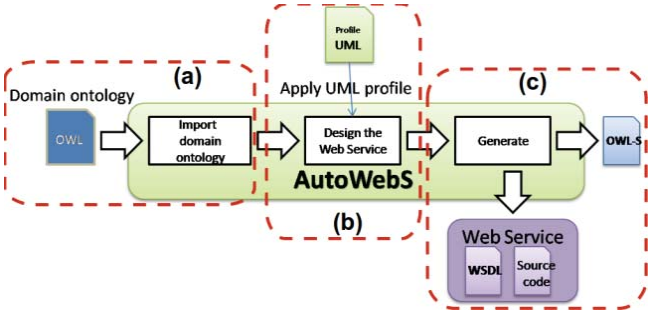


Figure 1. Overview of the AutoWebS

In the first activity, (a), the tool maps the elements of the OWL ontology to UML elements, and the result is a UML model (class diagram) that represents the OWL ontology. The UML profile customizes the model using stereotypes, constraints and tagged values, which are attached to UML model elements. They define the semantics of the UML model elements and make it possible to add additional information such as properties that define the *port* where are the WS, and its *endPoint*. In the (b) activity, the user works at the modeling level rather than on the direct manipulation of ontology languages. To create UML models that specify the SWS interface, the user relies on stereotypes, constraints and tagged values defined in the UML profile. From the perspective of the SWS designer, the tool resembles an UML class diagram editor (**R0**). In the environment provided by AutoWebS it is possible to create UML models that specify the SWS interface by using elements of the UML class diagram. In the (c) activity, the UML model that specifies the SWS interface is the input to a set of model-to-model and model-to-text transformations that automatically generate the semantic description of the WS in OWL-S ontology (**R2**) and build a WS project to the Eclipse IDE (**R1**). The created WS project contains: (i) the WSDL document, (ii) the descriptor of the WS, (iii) the classes that compose the SOAP communication infrastructure, and (iv) the Apache Ant script that automates the tasks of compiling and packaging the WS. To ensure that the semantic description of the WS is valid, the following validators are used (**R5**): the RDF validator; the OWL validator; and the OWL-S validator available in the OWL-S API.

C. Motivating Example

For the purpose of illustration of how to create a SWS, consider the WS *Barnes & Noble Price Finder* (http://www.mindswap.org/2004/owl-s/services.shtml) that returns the price of a book as advertised in Barnes and Nobles Web site given the ISBN Number. This WS has the following operation:

```
GetBNQuote (ISBN:String)Price:String;
```

There are several technologies that can be used to implement this WS and also different ways to structure its inputs and outputs. For example, we can represent the input Book as a class that contains the attributes *title* and *author*, or the input can only be a String. Abstract types are elements specified within WSDL documents used to characterize the inputs and outputs of the WS.

Figure 2 illustrates the declaration of the abstract types for the *Barnes & Noble Price Finder* WS. The *GetBNQuoteSoapIn* and *GetBNQuoteSoapOut* elements (lines 20 and 23) specify the request and response messages of the WS. Such messages are defined by *GetBNQuote* and *GetBNQuoteResponse Parts elements* (lines 3 and 10), which provides a logical description of the message through XML Schemas.

```
1  <wsdl:types>
2    <xs:schema elementFormDefault="qualified">
3      <xs:element name="GetBNQuote">
4        <xs:complexType>
5          <xs:sequence>
6            <xs:element minOccurs="0" maxOccurs="1"
   name="sISBN" type="xs:string"/>
7          </xs:sequence>
8        </xs:complexType>
9      </xs:element>
10     <xs:element name="GetBNQuoteResponse">
11       <xs:complexType>
12         <xs:sequence>
13           <xs:element minOccurs="0" maxOccurs="1"
   name="GetBNQuoteResult" type="xs:string"/>
14         </xs:sequence>
15       </xs:complexType>
16     </xs:element>
17   </xs:schema>
18 </wsdl:types>
19
20 <wsdl:message name="GetBNQuoteSoapIn">
21   <wsdl:part name="parameters" element="GetBNQuote"/>
22 </wsdl:message>
23 <wsdl:message name="GetBNQuoteSoapOut">
24   <wsdl:part name="parameters" element="GetBNQuoteResponse"/>
25 </wsdl:message>
```

Figure 2. WSDL Document

In order to semantically describe this WS, the concepts defined in *BibTex* (http://purl.org/net/nknouf/ns/bibtex) and *Concepts* OWL ontologies may be used. The *BibTex* ontology formally defines the concepts and their relationships in the domain of the reference management software for formatting lists of references. The *Concepts ontology* defines the concept of price. In the *BibTex ontology*, *Book* is an OWL class that formally defines the concept of a book and in the *Concepts ontology* there is the *Price* class that defines *price.*

The OWL-S ontology that semantically describes the WS needs to model the *GetBNQuote* operation as an entity that exchanges semantic data serialized in XML messages and associates the input and output with elements of the *BibText* and *Concepts* ontologies. The WS input should be associated with the *Book* class and the WS output should be associated with the *Price* class. The OWL-S ontology must also specify how the receiver can interpret the XML message and send it back into the semantic data and vice-versa. The transformations of the inputs and outputs are performed by XSL Transformations.

```
1 <owl:NamedIndividual rdf:about="bibtex:MyBook">
2   <rdf:type rdf:resource="bibtex:Book" />
3   <hasPrice rdf:datatype="&xsd;string">1234</hasPrice>
4   <hasYear rdf:datatype="&xsd;string">2012</hasYear>
5   <hasAbstract rdf:datatype="&xsd;string">Abstract</hasAbstract>
6   <hasAuthor rdf:datatype="&xsd;string">Author</hasAuthor>
7   <hasISBN rdf:datatype="&xsd;string">ISBN number</hasISBN>
8   <hasLocation rdf:datatype="&xsd;string">Location</hasLocation>
9   <hasPublisher rdf:datatype="&xsd;string">Publisher</hasPublisher>
10  <hasTitle rdf:datatype="&xsd;string">Title</hasTitle>
11 </owl:NamedIndividual>
```

Figure 3. Instance of the OWL class Book

The process of manually creating an OWL-S ontology is an error prone task that consumes time and effort. For example, the OWL *Book* class used to semantically describe the WS input has several properties, among them *hasPublisher*, *hasTitle*, *hasYear*, *humanCreator*, and *hasISBN*. The structure of the OWL *Book* class and consequently an instance of this class is quite different from the WS input (String). Figure 3 shows an instance of the OWL *Book* class, which may be a hypothetical input of the WS. This example shows some high-level activities to create a SWS, and the real need for tools to create SWS. The produced artifacts, such as the syntactic description of the WS (WSDL document) and the WS ontology are difficult to perform by hand.

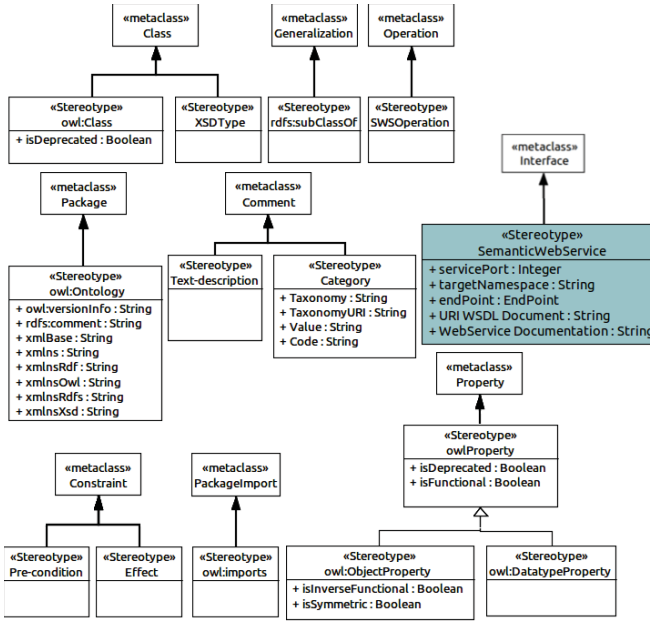


Figure 4. UML Profile

I. Implementation Details

This section presents the implementation details of AutoWebS. It is composed of: (i) an UML editor, (ii) a mechanism to import ontologies, and (iii) a mechanism for the automatic generation of SWS descriptions and some source code from a UML model (**R4**). AutoWebS uses the Papyrus UML graphical editor [12] that supports UML profiles and allows creating custom editors based on UML standards. The QVTo plugin is used to perform the following model-to-model

transformations: (i) from an OWL ontology (in OWL/XML format) to an EMF model (equivalent to the OWL metamodel), (ii) from an OWL model to a UML model; and (iii) from an UML model to an OWL-S model. The OWL-S document is created from OWL-S model using the code generator Acceleo. The source code of the WS is created from the UML model (**R1**) using *UML to Java Generator* plugin. AutoWebS extends the functionality of Axis2 to enable the generation of source code and also to allow the creation of WSs projects to the Eclipse platform (**R0**). It also offers some facilities for compiling, packaging, and deploying WS. All features offered by this component are accessible via buttons or menus in our tool. New functionalities can be integrated into AutoWebS with the insertion of new plugins (**R6**) since the infrastructure offered by Eclipse enables the development of modular applications as plugins. Furthermore, the MDD approach implemented by AutoWebS allows the insertion of a new semantic description language. For this purpose, it is necessary to create a metamodel to such language and to create the model-to-model and model-to-text transformations.

A. *UML Profile*

The UML profile (Figure 4) defined and implemented by AutoWebS has two main purposes: (i) to allow the partial representation of OWL ontologies as UML models, and (ii) to represent meta-information about the WS in UML models. We chose to propose a new UML profile for OWL because the AutoWebS represent only the necessary OWL elements to specify the SWS instead of the full OWL ontology. In this profile the *owl:ontology* stereotype is applied to the definition of an UML Package and corresponds to the Ontology OWL element. This stereotype defines the ontology domain and contains information about the version of the ontology, author, comments, and namespaces' declarations. An OWL ontology can import concepts from other ontologies. The *owl:imports* stereotype enables the relationship with other ontologies. The *owl:Class* stereotype represents a concept that has been modeled as an OWL class. The OWL properties (DatatypeProperty and ObjectProperty) are mapped to attributes of a UML class with the *owl:Class* stereotype. In these attributes are applied the *owl:DatatypeProperty* and *owl:ObjectProperty* stereotypes. The *rdfs:subClassOf* stereotype is applied to the UML Generalization metaclass and represents OWL classes generalizations.

The *Text-description* and *Category* stereotypes extend the UML *Comment* metaclass and enable the textual description of the SWS and the association with a category. The *Pre-condition* and *Effect* stereotypes extend the UML *Constraint* metaclass in order to specify constraints which must be satisfied for the WS to properly run. In OWL-S *preconditions* and *effects* elements are represented as logical formulas and can be expressed with languages whose standard encoding is XML or literal strings. In the current version of the AutoWebS, logical formulas are set as UML *Constraint*.

The *SemanticWebService* stereotype is applied to an UML interface, which defines the WS operations (modeled as methods). This stereotype has the following properties: *target-Namespace* that defines the namespace used in the WSDL document; *URI WSDL* that determines the WS URI; *Web-Service Documentation* that serves for documentation purposes; *servicePort* specifies the port on which the WS responds; and the *endPoint* determines the type of endpoint used for end-to-end communication. The methods defined in the interface stereotyped as *SemanticWebService*, which represent the operations of the WS are stereotyped as *SWSOperation*. The input and output of the operations can be defined as UML classes, UML primitive types or classes from the OWL ontology, which are represented in the UML model by *owl:Class* stereotype.

In our proposed MDD approach, the OWL ontology import process is implemented in two phases. In the first phase, a set of QVT transformations transforms an OWL ontology in an OWL model (according to the OWL metamodel). In the second phase, some elements of the OWL model are mapped to UML elements, so that at the end of the two phases the outcome is an UML model (class diagram), that can be used to model the interface of the SWS. In the resulting UML model, the UML profile provides additional meta-information inherent to the imported ontology and of the context of SWS. For example, the UML model includes properties that define the port on which the WS is listening, *endPoint*, and *namespaces*.

B. *MDD Aproach*

AutoWebS implements a MDD approach, illustrated in Figure 5, in order to meet the **R0**, **R1**, **R2**, and **R3** requirements. This strategy allows separation of concerns between the specification and implementation, and provides a high abstraction level of the OWL language. In our proposed MDD approach, a set of QVT transformations transforms an OWL ontology (or a set of ontologies) in an OWL model (according to the OWL metamodel), and some OWL elements of the OWL model are mapped to UML elements, producing an UML model (UML class diagram). This UML model need be manually extended to incorporate the WS definitions. From the UML model that describes the SWS interface, the tool automatically generate the WS project to Eclipse IDE (**R1**),

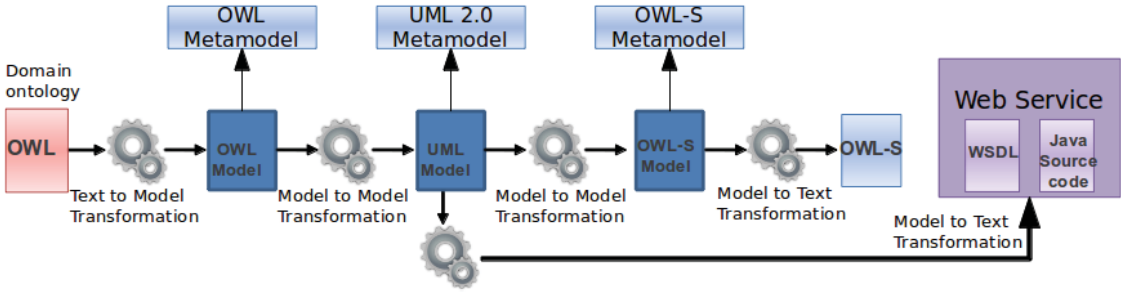


Figure 5. MDD Approach.

and the OWL-S document. AutoWebS exports the graphical representation of the UML model to an XMI document, and from the XMI document the artifacts are created. To create an OWL-S document, AutoWebS uses an OWL-S metamodel and a set of QVT transformation rules that are applied to the UML model to automatically generate the OWL-S model (according to the OWL-S metamodel). AutoWebS uses an Acceleo model-to-text transformation that transforming the OWL-S model into the OWL-S document. The source code in Java, which composes the WS project to Eclipse IDE, is generated from the UML model by the model-to-text transformation *"UML to Java"*. The WS operations represented through public methods in UML interface (stereotyped as *SemanticWebService*) are mapped to methods in Java Interface, and they are used to generate the source code of the WS (each public method represents an WS operation). AutoWebS calls the API Axis2 to perform the following tasks: (i) to create the WSDL document associated with the WS; (ii) to generate the source code of the WS, i.e. the code artifacts that make up the SOAP infrastructure of the WS; (iii) to create the *build.xml* Ant script. The main code artifacts generated are: i) the *services.xml* deployment descriptor, which contains the runtime configuration of the WS, ii) the *MessageReceiver*, which is responsible for the end-to-end communication, iii) *Skeletons* that implement the protocol used for the transmission of messages, and iv) Apache Ant *build.xml*. The *build.xml* file is used to automate the building process and packaging the WS. All artifacts are attached in a project for the Eclipse platform, so the user can develop the business rules of the WS. After the implementation of the WS business rules, the user can use the features offered by AutoWebS to build and to deploy the WS. The result of the compilation is a file with the *aar* extension containing that can be deployed in a Web Container that has installed the Axis2 runtime.

II. CREATING A SEMANTIC WEB SERVICE

This section shows how AutoWebS can be used to create SWS, demonstrating the process of creating the *Barnes & Noble Price Finder* SWS. The demonstration encompasses four steps: (i) to import the *Bibtex* and *Concepts* OWL ontologies; (ii) to model the SWS interface; (iii) to trigger the generation of the OWL-S ontology, and the source code of the WS; and (iv) to implement the WS business rules. Steps (i) and (iii) are automated by AutoWebS and the user only needs to manually perform the steps (ii) and (iv). Step (i) imports the *Bibtex* and *Concepts* OWL ontologies, which contains the *Book* class that can be mapped to a UML class. The OWL *Book* class is a subclass of *Entry* and contains the properties: *hasTitle*, *humanCreator*, *hasPublisher*, and *hasYear*. The OWL *Book* class is mapped to UML as a *packageElement* element with *uml:Class* type and *Book* name. The UML *Book* class is a generalization of the UML class *Entry*. The OWL property classes are mapped to the *ownedAttribute* XMI element, keeping their names. The values are mapped to the UML primitive types. The Step (ii) consists of adjusting the UML model. In this activity the user can insert or remove elements in the UML model. Figure 6 illustrates an UML model that contains the interface stereotyped with *SemanticWebService*. This UML model defines the *BNPrice* SWS. In this SWS the *GetBNQuote* operation receives as input a *Book* element of the *BibTex* ontology and returns its price as a *Price* element of the *Concepts* ontology. The Step (iii) models the SWS interface. This step consists of defines the interface of the WS and triggering the mechanism of automatic generation of OWL-S file and WS project to the Eclipse platform. In this step the transformations presented in Section 3.3 are performed. The Step (iv) consists in the implementation of the WS business rules. After implementing the WS business rules, the next step consists in using the functionality of AutoWebS to build and deploy the WS. The WS project is built using Apache Ant's buildfile *(build.xml*). The result of the building process is a file with the *aar* extension. To deploy this WS, AutoWebS makes a copy of the file with the *aar* extension into the Web Container that has installed the Axis2 runtime.

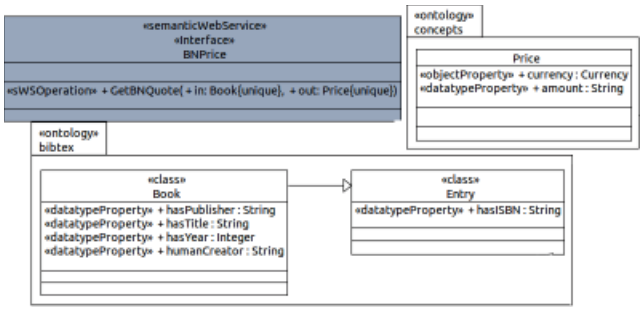


Figure 6. UML model to the semantic Web service BNPrice.

III. ANALYSIS

For purposes of analysis, we have conducted a controlled experiment that evaluated AutoWebS and an application suit comprised by the OWL-S Editor and the Axis2 Eclipse plugin, in the activities for the creation of SWS. The choice of OWL-S Editor and Axis2 Eclipse plugin was motivated by the fact that together they have some of the functionalities of AutoWebS and are used to create SWS. All the details of the experiment (hypotheses, methods and results) are available at www.consiste.dimap.ufrn.br/projetos/autowebs. We applied eight replicas of the experiment and created two SWS for each *semantic WS project*, each one created by a different tool. The results were analyzed with the Wilcoxon nonparametric statistical test [13]. Table I shows the times, in minutes, required for developing each SWS and the number of errors or inconsistencies in each OWL-S ontology.

TABLE I. RESULTS

	AutoWebS		***Application suite***	
	time	***error***	***time***	***error***
OilMonitor 1	12	0	35	1
OilMonitor 2	8	0	55	1
Book Finder	20	0	35	1
Zip Code Finder	5	0	21	1
LatLongFinder	8	0	25	1
Barnes & Nobles	5	0	20	1
BabelFish Translator	5	1	45	1
Currecy Converter	5	0	14	1

The WS ontologies created by AutoWebS do not have errors or inconsistencies and in all OWL-S ontologies created by the application suite, errors or inconsistencies were found.

Figure 7 illustrates the time consumed by the participants to develop each SWS. The development time using AutoWebS

was shorter than the time when using the application suite. Based on data contained in Table 1, we can conclude that for all SWS the values for the metrics "*time*" and "*errors*" for AutoWebS were smaller than or equal to the application suite.

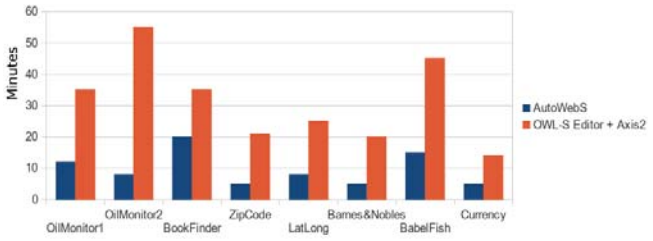


Figure 7. Time to develop each semantic Web service (SWS)

IV. Related Work

Some MDD approaches to create OWL-S ontology have been proposed in the literature. (i) ASSAM [14] is a tool that provides a graphical interface based on views. It uses a semi-automatic process where the user defines the mappings between concepts of an OWL and the *Message* elements of the WSDL document. This tool uses a learning algorithm that when a user defines the mappings, it presents some suggestions of the ontology classes that can be associated with the WSDL elements, and (ii) CODE [15] is a plugin for the Eclipse IDE. CODE provides a converter called *WSDL2OWL-S* to generate the semantic description of WSs in OWL-S. The tool has four editors that enable to edit the OWL-S sub-ontologies. The editors are based on forms that must be filled with data from the WS ontology. The aforementioned tools apply different approaches. Table 2 presents a comparison of them with AutoWebS, considering the requirements for a high-level abstraction tool for creating SWS (Section 2.1). "*Y*" means that the tool fully meets the correspondent requirement, "*N*" means that the tool does not meet the requirement, and "*P*" means that the tool partially meets the requirement.

TABLE II. Comparison between the tools

	Requirement						
	R0	***R1***	***R2***	***R3***	***R4***	***R5***	***R6***
(i)	N	N	Y	Y	N	P	N
(ii)	Y	Y	Y	N	P	P	Y
AutoWebS	Y	Y	Y	Y	Y	P	Y

As shown in Table 2, ASSAM does not fulfil the **R0**, since it requires an extensive knowledge about XML Schemas and OWL. Moreover, the manual association between ontology concepts and the WSDL elements may be difficult to be performed, since the number of possible combinations of mappings may be large. This tool is not integrated into a development environment and it does not provide functionality for creating the source code of the WS, and the WSDL document. Thus, ASSAM is not in compliance with **R1** and **R4**. The tool does not support the integration of new functionalities, therefore does not meet **R6**. The CODE tool has no mechanism to import the ontological concepts, therefore it does not meet **R3** and **R4**. Furthermore, the OWL-S ontology generated by the *WSDL2OWL-S* converter is incomplete and requires manual processing to supplement the *ServiceProfile* and *ServiceModel* sub-ontologies. For this reason, this tool partially meets the **R5** requirement. As CODE uses the Eclipse platform, it is possible to integrate new modules that provide new functionality (**R6)**. The *Java2WSDL* and *WSDL2OWL* generators provide some abstraction of the underlying technologies used to develop the SWS, thereby meeting **R0**.

V. Conclusions

This paper presented a MDD tool to make the development of SWS more intuitive and easy. The approach implemented by AutoWebS allows developers to focus their efforts on creating models instead on writing source code. Additionally, the fact that the models created from UML profiles are valid UML models and due to extensive use of UML as modeling language, this approach makes AutoWebS accessible to a wider audience. AutoWebS has some limitations that do not prevent its use. The current version of the algorithm for creating XSLT scripts is only able to create scripts to ontology concepts which have properties defined as primitive types or other concepts defined as OWL classes. For more complex cases, such as lists, our algorithm is unable to generate the XSLT script.

Acknowledgments

This work was partially supported by CNPq through the grant 485935/2011-2 for Thais Batista. Flavia Delicato and Paulo Pires are also partially supported by FAPERJ and CNPq (grants 311363/2011-3, 470586/2011-7, 310661/2012-9).

References

[1] McIlraith, S. A., Son, T. C., and Zeng, H. (2001). Semantic web services. IEEE Intelligent Systems. 16(2):46–53.

[2] Booth, D., et al.. (2007) Web services description language (WSDL) version 2.0 part 0: Primer. Tech. report, World Wide Web Consortium.

[3] Martin D, Burstein M, Hobbs J, et al. OWL-S: Semanic markup for Web Services. 2004. http://www.daml.org/services/owls/1.1/overview/.

[4] Brambilla, M., et al. (2007). Model-driven design and development of semantic web service applications. ACM Trans. Internet Technol., 8.

[5] Missikoff, M., et al. (2002). The usable ontology: An environment for building and assessing a domain ontology. In Proc. of the Int. Semantic Web Conf. (ISWC), volume 2342, p 39–53. Springer-Verlag.

[6] Chafle, G., et al. (2007). An integrated development environment for web service composition. In . IEEE Int. Conf. on Web Services (ICWS) 2007, pages 839 –847.

[7] Stahl, T. and Völter, M. (2006). Model-Driven Software Development: Technology, Engineering, Management. Wiley, Chichester, UK.

[8] Atkinson, C., et al. (2006). On the relationship of ontologies and models. In Proc. of the 2nd Int. Work. on Meta-Modelling, pages 47–60.

[9] Bechhofer, et al.: OWL - Web Ontology Language Reference, W3C Recommendation. 10 February 2004.

[10] Perera, S., et al. (2006). Axis2, middleware for next generation web services. In Proc. of the IEEE Int. Conference on Web Services, pages 833–840, Washington, DC, USA. IEEE Computer Society.

[11] Elenius, D., et al. (2005). The OWL-S editor - a development tool for semantic web services. In Proc. of the 2nd European Semantic Web Conf., p 78–92. Springer.

[12] Gérard, S., et al. (2007). Papyrus: A UML2 tool for domain-specific language modeling. In Model-Based Engineering of Embedded Real-Time Systems, p 361–368.

[13] Corder, G. and Foreman, D. (2009). Nonparametric Statistics for Non-Statisticians: A Step-By-Step Approach. Wiley.

[14] Heß, A., et al. (2004). ASSAM: A tool for semi-automatically annotating semantic web services. In 3rd Int. Semantic Web Conf., p 320–334. Springer.

[15] Srinivasan, N., et al. (2005). CODE: A Development Environment for OWL-S Web services. Tech. Report CMU-RI-TR-05-48, Robotics Institute, Carnegie Mellon University.

A Knowledge Modeling System for Semantic Analysis of Games Applied to Programming Education

Elanne Cristina Oliveira dos Santos[1,2]
Gleison Brito Batista[1]
[1] Instituto Federal de Educação, Ciência e Tecnologia do Piauí, IFPI
Teresina, Brasil
elannecristina.santos@ifpi.edu.br
brittobaptista93@gmail.com

Esteban Walter Gonzales Clua[2]
Instituto de Computação
[2] Universidade Federal de Fluminense, UFF
Niterói, Brasil
esteban@ic.uff.br

***Abstract* — Teaching and learning algorithms and programming is being an important challenge, not only in universities, but also in schools. With the purpose of making this process more efficient and enjoyable, we introduce Jplay providing structures and architectures for developing simple games. In this paper we introduce a new knowledge modeling system for semantic analysis of games applies to programming teaching within Jplay. The analyzer semantically interprets the programmer code and return detailed analysis of its correct functionalities. Our process consists of comparing the program with a model program, previously defined, searching for similar behaviors and structures between them. We developed a strategy for knowledge modeling based on pairs of similar classes and implemented a tool that enables the proposed analysis.**

Keywords- programming; JPlay; knowledge; classification; teaching; games

I. INTRODUCTION

Many studies point to the difficulty of teaching and learning the disciplines related to algorithms and programming, resulting in high dropout rates in computer courses [12][4]. The main reason for this negligence is the difficulty in learning abstract concepts of programming [13]. Trying to solve this issue, there are several proposals including application of new technological tools [2][10][16][3].

In this sense, the JPlay framework was proposed and developed for teaching programming [8][9]. Jplay is a framework developed in order to facilitate 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 programing necessary for a correct learning of algorithmic logic and does not introduce specific features of design patterns 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 helper objects that help to create 2D games using the Java language.

In this paper, we present a novel knowledge modeling system for semantic analysis of games applied to teaching programming using the JPlay framework. The analyzer 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 analyze the behaviors of a program for games using JPlay. The behaviors of the program can be obtained by the analyzer through the comparison with another program treated as model. 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 programmer, point out problems and suggest possible solutions. We developed a tool that enables the proposed analysis.

II. RELEATED WORK

Different works presents the modeling and representation of source code using Extensible Markup Language (XML) for various purposes.

The representation of source code in text format is widely used for encoding algorithms. However, for knowledge based system, the representation of programs needs a more abstract representation and converting it to meta-structures becomes an important issue [14].

The CodeMI [14] proposes a source code representation in XML format. The CodeMI aims to enable studies of the evolution of commercial software, through the use of data repository, where the representation of source code has a central role, preserving the characteristics necessary to conduct the studies of the evolution of software, but at the same time hide the details of its implementation.

Another representation of code as XML format is JavaML from Mamas and Kontogiannis [11]. In this work the authors generated XML files, one for each class of system, storing information on keys elements such as classes and their methods and attributes, but does not store information about the implementation strategy of the methods.

Badros et al. proposes the JavaML Badros [5] (or JavaML 1.0). This proposal also generates XML files, one for each class of system. In comparison with the representation of Mamas and Kontogiannis, in this work is possible to get information about the implementation of the methods, and there is an identifier (id) for each method and variable, with a reference to the identifier (idref) for every relationship that use these entities.

JavaML 2.0 of Ademar Aguiar [1] is based on JavaML of Badros and presents improvements that are mainly based on total preservation of the source code, which greatly increased the number of tags and the representation become more robust.

In relation to tutoring systems for programming, ProPat, is a sub-project of IBM Eclipse project for the construction of an Integrated Development Environment (IDE) for programming introduction courses and is oriented for C language. The ProPAT analyzes the student´s program, according to the standards specified by the teacher, and returns the results in Eclipse IDE [6].

Silva et al. [15] proposes a system for source analysis called Vertical Code Completion (VCC). The VCC is based on two phases: mining sequential patterns and querying these patterns. In the second phase, a source code is analyzed in order to find pieces corresponding to patterns generated in the first phase. The patterns are sorted by metrics and suggested to the user. The analysis of the code in the VCC is performed by the usage of an abstract syntax tree (AST) [17]. The restriction analysis by VCC is that it does not consider the conditional structures and the structures of exception handling, understand the source code that depends on these conditions as a simple sequential code.

None of the above representations was specifically developed to analyze semantically a Java source code represented in XML and generate results of this analysis. Our work differentiates from others by proposing a XML representation that makes possible the semantically analysis of a source code by applying a clustering algorithm and classification process of similar classes. Another important difference is that our proposal is based on a design pattern oriented to simple 2D game, according to the original purpose of JPlay [8][9], focusing the analysis at the behaviors between pairs of classes previously detached and classified.

III. JPLAY

A. JPlay Architecture

In Jplay architecture, we divide the JPlay diagram into three parts: the interaction between game and player, characters and output the game.

The classes responsible for interaction between game and player are Keyboard (it defines input data for the computer keyboard) and Mouse (it defines input data for computer mouse). The classes responsible for creating the characters of the game are Animation (it defines an animation. It must have a picture and their frames), Sprite (it extends the Animation class and contains methods that can make the image move across the screen) and Body (it 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 (it defines a window where all the game elements will be drawn), Time (it defines a time counter), Sound (it defines the sound that will be played in the game) and Collision (it's 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 collided in all classes, or by Collision static class).

B. JPlay Sequential Pattern

Our proposal is based on design patterns of sequence used in 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.

A typical sequence of activities of the JPlay framework is in Fig.1. In the sequence first a main method must be defined in the initialization of the program; then in the body of the main method objects will instantiated, one of these objects must contain a game loop (infinite loop). Finding a class that contains this loop means finding the loop execution of the program; then in the class that contains the game loop (infinite loop) objects are declared, at this point it is possible to check if all objects have been declared also have been instantiated; then objects declared as Sprites, Animations and GameImages should be drawn in the window of the game, at this point it is necessary to verify if all declared objects of these types were draw; then an object declared of Window should be updated in the game window; at this point it is necessary to verify if the Window object is updated. For the objects updated in the window, he should call the update method.

IV. MODELING AND ANALYSIS OF PAIRS OF SIMILAR CLASSES BETWEEN PROGRAMS

Since Badros [5] approach allows the preservation of the source code and our methods need a subsequent semantic analysis, we initially convert all classes from a developed program into a XML representation, based the proposed JavaML method. Due the increase of tag's representation from JavaML 2.0, we ignored this update for our proposal. CodeMI approach does not suit the goals of this work, because their representation is not fully preservative in relation to the structure of the source code, and consequently it hides a lot of information in an attempt to ensure the authoring protection of the program.

A. JavaML Representation

For this work we initially developed a parser based in the JavaML Representation. However, changes at the original JavaML representation were necessary in order to adapt the model for the purpose of this work.

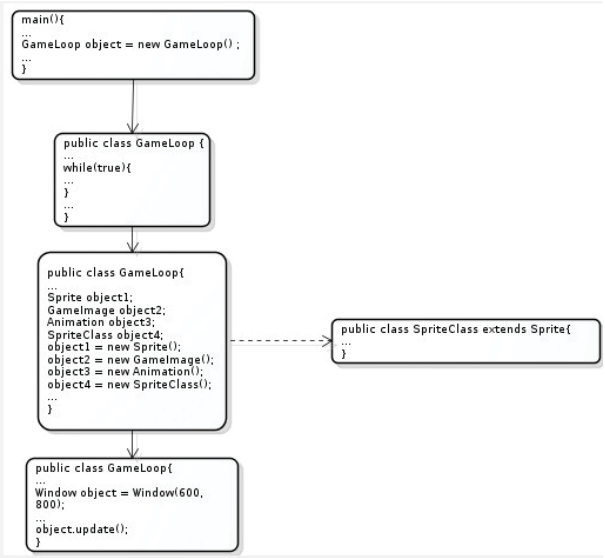


Figure 1 – JPlay sequential pattern

An additional maker was created with the name of the behavior. A behavior marked has assigned also its attribute <type>. In the case of an assignment command, the attribute <type> receives the value of the <assignment>; when identifying the conditional command, the attribute <type> receives the value of the <conditional>; when identifying the loop command, the attribute <type> receives the value of the <loop>.

It is important to note that the inclusion of the behavior marker does not delete or change almost none of the existing markers in the JavaML representation. The only exception concerns about the marker <do-while>, which was defined by the representation of Badros differently from the rest of the markers <while> and <for>. We chose, however, to maintain a standard for all markers <while>, <for> and <do-while>, then everyone gets the value <loop> in attribute <type> and after the attribute <kind> identifies the three types of repeating loops (<while>, <for> or <do-while>). The objective was to minimize the changes and keep the original representation, only adding a markup over the generated XML.

Following a example the marking <if> shows the original markings of representation and then immediately how the marking <behavior> was added in the code, the marking <if> is accompanied by a sub-markup <test> that identifies the test expression required for the conditional statement is performed and a sub-markup <true-case> that identifies what should be executed if the expression is true:

<if><test></test><true-case></true-case></if>

Then, we add the markup <behavior>:

<behavior id='' type='conditional'><test></test>

<true-case></true-case></behavior>

B. Grouping Pairs of Classes

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) [7]. DOM obeys the standard World Wide Web Consortium (W3C) and defines standard access to XML and HTML documents. The DOM interface defines objects and properties to all elements of these types of documents mentioned above, giving also methods to access them.

To group the pairs of classes we developed a matrix of objects, where the columns are the total number of classes from the model program and the lines are the total number of classes in the programmer code. The rules are analyzed and the results are stored in the structure of the matrix of objects. In the diagram of Fig. 2 it is possible to see the GeneralRules class, that implements the methods necessary to verify all the rules, the MatrixSimilarFirstPair class, that is responsible for the assignment of the similarity of pairs according to the specified rule for the first pair and is also responsible for verifying which pair has a higher similarity, the MatrixSimilarSecondPair class, that assigns the similarity pair value according to the rules specifies for the second pair and verifies which pair has higher similarity. In the diagram of the Fig. 2 each component of matrix is represented by an Element class.

The attributes of the Element class is stored in order to be used later as parameters of the program analysis. Below are the descriptions of these attributes:

- Weight of the similarity of each pair;
- Number of declared classes (from the same package) and number of declared classes (from imported packages) in the analyzed class in the programmer code and in the analyzed class in the model program;
- Number of instantiated classes (from the same package) and number of instantiated classes (from imported packages) in the analyzed class in the programmer code and in the analyzed class in the model program;
- Amount of classes (from the same package) and amount of classes (from imported packages) that call the draw method in the analyzed class in the programmer code and in the analyzed class in the model program;
- Amount of classes (from the same package) and amount of classes (from imported packages) that call the update method in the analyzed class in the programmer code and in the analyzed class in the model program;
- Results array, where each array element is a property that represents the result of a rule considered for that pair. The goal of this results array is to identify properties with a value of false as critical points. Each property has a boolean value and initially all values are set as true, indicating that they do not represent any critical points in the program. Each property

represents a rule that must be analyzed, then when a rule is true for a pair, a property receives value equal to true, and when a rule is false for a pair, a property receives value equal to false.

The properties are Main Similarity, Game Loop Similarity, Declaration Similarity, Initialization Similarity, Draw Similarity and Update Similarity. The use of properties is described in more detail in the section that deals the rules of the first and second pair of similar classes.

The pairs are compared using previously established rules based on standard Java projects with Jplay framework. After that, the pair of most similar classes between the programmer code and the model program is classified through our algorithm of similarity weight. During the process of grouping pairs of classes, attributes in Element class are filled according to the characteristics of the programs.

C. Classification of Pairs of Classes

After defining the pairs of classes, the comparisons between pairs are performed. The goal on this stage consists on classifying the pairs of classes that have higher similarity. The analysis of similarity between pairs of classes is based on rules previously established.

Each class in the model program is compared with all classes of the programmer code, and, according to the rules, weights are assigned to each compared pair.

At the end of each rule or set of rules applied, a pair of classes with the highest weight is classified. Through the result of comparisons between pairs, critical points of the programmer code are posted. Based on these critical points the analyzer can show the results of the analysis, in order to identify the problems found in the program and then proceed with suggestions of possible solutions.

D. Specific Rule of Classification for the first Pair of Similar Classes

In order to classify the first pair of similar classes we establish only one rule: the class must have the method main. Initially, the matrix is initialized with its similarity value, represented by the attribute weight, equal to 0 (zero) in all its elements.

The class which contains the method main will be increased by 1 (one) in its similarity value. When the analyzer finds two classes containing the main method, the similarity weight value of this pair will be equal to 2 and the Main Similarity property will continue as true. The rest will have similarity weight value equal to 1 or 0 and the Main Similarity property value equal to false, indicating a possible of a critical point this pair. The pair with the higher weight should be selected as the first pair of similar classes.

E. Specific Rules of Classification for the second Pair of Similar Classes

In order to classify the second pair of similar classes we established five rules:

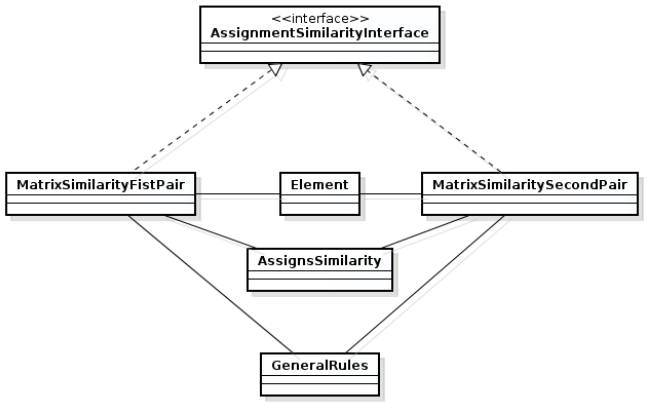


Figure 2 - Class diagram of the Semantic Analyzer

- Rule 1: If, for each class in the pair, there is a game loop, then the weight value is increased by 10 and the value of the results array at the Game Loop Similarity property will continue as true, otherwise the value of the Game Loop Similarity property is set to false.We emphasize that the rule 1, which deals with the existence of an game loop in the class, is the most important at this stage, so it has the highest weight. The game loop can be characterized by a search through the DOM interface with a tag called <behavior> of the type <while> which contain a boolean expression, a value equal to true or even a game ending condition.
- Rule 2: if the number of declared classes in the analyzed class of the programmer code is equal to the number of declared classes in the analyzed class of the model program, then the weight value is increased by 1 and the value of the results array at Declaration Similarity property continues as true, otherwise the value of the Declaration Similarity property is set to false, indicating a possibility of a critical point for this pair.
- Rule 3: if the number of classes declared in the analyzed class by programmer code is the same amount of classes that were instantiated of the program, then the weight value is increased by 1 and the value of the results array at Instantiation Similarity property continues as true, otherwise the value of the Instantiation Similarity will be set as false, indicating a possible critical point in these classes pair.
- Rule 4: if the number of objects by programmer code that call the draw method is equal to the number of objects that inherit from Sprite class, Gameimage class or Animation class in this program, then the weight value is increased by 1 and the value of the result array at Draw Similarity property is equal to true, otherwise it is false.
- Rule 5: if the number of objects by programmer code that calls the update method is equal to the number of

objects that inherit from Window class in this program, then the weight value is increased by 1 (one) and the value of the result array at Update Similarity property is equal to true, otherwise it is false.

F. General Classification Rule from the third Pair of Similar Classes on

At this stage we dispose the previously classes selected at the first and second pair of similar classes. For the third pair of similar classes, the rules are general for all other pairs and are based on behaviors of the game.

Initially it is checked if the pair of classes inherits from the same super class. If this happens, its weight will be incremented. For each class, lists are created with information about the behavior of the variables. The lists are as follows:

a) List of variables, formed by elements that contain variable type attribute and amount of variables of this type attribute;

b) List of variables behaviors: For each variable there is a list of behaviors formed by the behavior identifier, behavior type, expression and action.

Initially, the lists of variables of classes are compared and the pair of classes that contain the most similar lists will be chosen for the second comparison. After that, the lists of variable´s behavior of the classes pair are compared. This comparison is based on the similarity of the 3 behavior elements:

a) Type: the value can be equal to conditional, loop or assignment. If they are similar, the weight is incremented by 1.

b) Expression: the description of the conditional expression from which depends on the behavior. If the expressions are similar, the weight is incremented by 1.

c) Action: the description of the action will be executed if the expression is true. If the actions are similar, the weight is incremented by 1.

During this process, the results of these comparisons are stored in attributes of the objects matrix in order to be used later in the processing of the analysis.

V. RESULTS

In this paper we presented a semantic analyzer for program developed using the Jplay and a knowledge modeling of simple game projects based on JPlay framework. To semantically analyze a program, the developer must select in a repository another program which will be considered as a model program, as shown Fig. 3. In the example in Fig. 3, we compare the programmer code called *Jogo* with the model program called *Pong*.

The *Jogo* program contains the classes *Main*, *Bola*, *Jogo* and *Barra*; the *Pong* program contains the classes *Pong*, *Ball*, *Bar*, *GamePlay* and *Pong*.

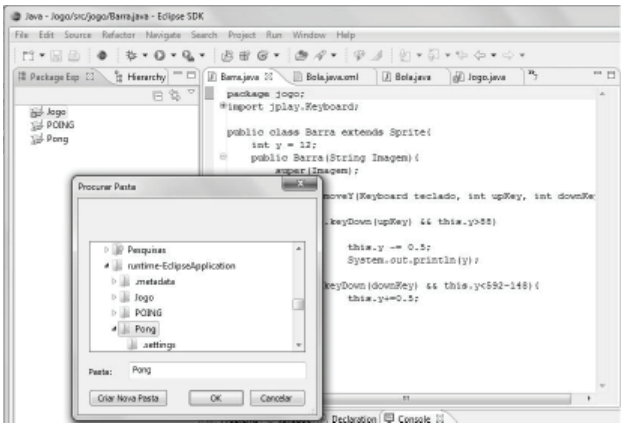


Figure 3. Choosing the model program at semantic analyzer plugin

The pairs of classes are compared and results are stored in the objects matrix. The goal of the analyzer is to interpret semantically a Java program that uses Jplay and return results to the programmer. The repository has available only XML files that represent the model program.

Initially, in the matrix, for each pair we assume that:

- The weight of similarity is equal to 0;
- The results array is entirely filled with true value.

After that, the analyzer will identify the first pair of similar classes and will look for classes that contain the main method in the two programs. This classification generates values for results matrix, as Tab. I. The matrix shows the pair [Main.java, Pong.java] with the highest similarity value equals to 2 and values of remaining pairs equal to 0 or 1.

After the analyzer to identify the first pair of similar classes, it will look for the second pair of classes with the highest weight of similarity according to the rules previously presented. This classification generates values for a new matrix as Tab. II. The matrix shows the pair [Jogo.java, Gameplay.java] with the highest similarity value equal to 16.

Through the definition of similar classes for the first and second pairs, it is possible to return some semantic analyzer results. Fig. 4 shows an example, where the analyzer tells to the programmer that probably the object *bola*, that is an object of the Sprite class (from the JPlay package), was not drawn on the game window, being this a semantic requirement based on the model program.

TABLE I. TABLE WEIGHT MATRIX FOR FIRST PAIR OF SIMILAR CLASSES

Programmer code classes	**Model program classes**			
	Ball.java	***Bar.java***	***Gameplay.java***	***Pong.java***
Barra.java	0	0	0	1
Bola.java	0	0	0	1
Jogo.java	0	0	0	1
Main.java	1	1	1	2

TABLE II. TABLE WEIGHT MATRIX FOR SECOND PAIR OF SIMILAR CLASSES

Programmer code classes	Model program classes			
	Ball.java	***Bar.java***	***Gameplay.java***	***Pong.java***
Barra.java	5	4	8	4
Bola.java	5	3	7	3
Jogo.java	13	11	16	12
Main.java	4	3	8	5

VI. CONCLUSION

This paper presents a knowledge modeling system and a novel semantic analyzer for programs developed using Jplay. Although our implementations and tests are related to this 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 programmer.

Our proposal brings significant contributions to researchers working in the field of programming and software engineering, specifically in relation to the knowledge modeling, having as a main contribution, an architecture for grouping and classification of similar classes to perform semantic analysis of programs based on XML representation. The proposal also contributes in the sense that we introduce a tool able to interpret semantically code built by programmers, returning results, pointing out problems and suggesting solutions.

As future work, we intend to use data mining and classification methods for improving the efficiency of the algorithm.

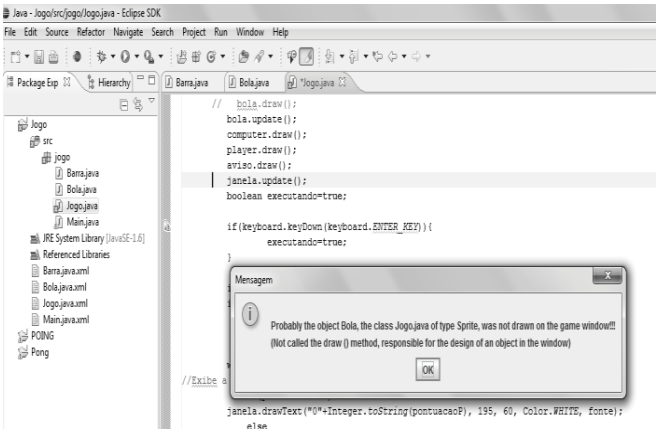


Figure 4 - The analyzer returns to the programmer telling that the object "bola" was not drawn on the game window

REFERENCES

[1] Aguiar A., David G., Badros G.L., "JavaML 2.0: Enriching the Markup Language for Java Source Code", in XML: Aplicações e Tecnologias Associadas (XATA'2004), Porto, Portugual, February 2004.

[2] Allen, E., Cartwright, R. and Stoler, B. Drjava: a lightweight pedagogic environment for java. SIGCSE Bull., 34(1):137-141.

[3] 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.

[4] Barbosa, Leonidas da Silva; 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.

[5] Badros G.J., "JavaML: A Markup Language for Java Source Code", In Proceedings of the 9th Int. Conf. On the World Wide Web (WWW9), Amsterdam, Netherlands, May 2000.

[6] Delgado, K. V. "Diagnóstico baseado em modelos num sistema inteligente para programação com padrões pedagógicos". Master's dissertation, Institute of Mathematics and Statistics. 2005.

[7] DOM, available in http://www.w3.org/DOM/. Accessed in November 2012.

[8] 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.

[9] Jplay, available in http://www.ic.uff.br/jplay/. Accessed in April 2012.

[10] Kolling, M., Quig, B., Pattern, 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.

[11] Mamas, E. and Kontogiannis, C., "Towards Portable Source Code Representatitions Using XML", in Proc, of the 7th Working Conf. on Reverse Engineering (WCRE'00), Brisbane, Australia, pp. 172-18, November 2000.

[12] Rapkiewicz, C. E. et al. (2006). "Estratégicas pedagógicas no ensino de algoritmos e programação associadas ao uso de jogos educacionais", http://ww2.deinfo.ufrpe.br:8080/licomp/Members/jeanemelo/plonelocalfolderng.2006-04-10.7475913377/PEP/Aula8/EnsinoJogos.pdf.

[13] Santos, N.S.R.S.; C.E. Rapkiewicz. 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.

[14] Santos, O.J.P. and Barros, O. M., "CodeMI – Source Code as XMI. Uma Representação de Código-fonte para Coleta de Métricas", in SBSI. 2009.

[15] Silva, L.L.N.D.; Plastino, T.N.D.O.A.;Gresta Paulino Murta, L., "Vertical Code Completion: Going Beyond the Current Ctrl+Space," Software Components Architectures and Reuse (SBCARS), 2012 Sixth Brazilian Symposium on , vol., no., pp.81,90, 23-28 Sept. 2012.

[16] Traetteberg, H. and Aalberg, T. Jexercise: a specification-based and test-driven exercise support plugin for eclipse. In eclipse "06: Proeedings of the 2006 OOPSLA workshop on eclipse technology eXchange, pages 70-74, New York, NY, USA. ACM Press. 2006.

[17] W. Holz, R. Premraj, T.Zimmermann e A. Zeller. Predicting software metrics at design time. 9th International conference on Product-Focused Software Process Improvement. Rome, pp. 34 – 44, June 2008.

Representing Chains of Custody Along a Forensic Process: A Case Study on Kruse Model

Tamer Fares Gayed, Hakim Lounis
Dépt. d'Informatique
Université du Québec à Montréal
Case postale 8888, 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
Case postale 8888, succursale Centre-ville, Montréal QC
H3C 3P8, Montréal, Canada
bari.moncef@uqam.ca

Abstract – Chains of Custody (*CoCs*) are tangible documents accompanying the evidences along the forensic process. They play vital role in the forensics investigation by demonstrating the road map of Who exactly, When, Where, Why, What, and How came into contact with the evidences in order to prosecute them in a court of law. In the cyber forensic, the evidences have digital nature; the fact that can make them easily altered and loses their values. With the advent of digital age, and the poor awareness of juries to understand digital evidences, *CoC* documents need to undergo a radical transformation from paper to electronic data (*e-CoC*) readable, discoverable, understandable, and consumable by people and computers. The semantic web is a fertile land to represent and manage the tangible *CoCs* because it uses web principles known as Linked Data Principles (LDP) which provides useful information in RDF upon URI resolution. These principles are used to publish data publicly on the web and provide a standard framework that allows such data to be shared, and consumed in a machine readable format. This paper provides a case study applied on a cyber forensic model, such as Kruse model, explaining how these principles are applied to represent chains of custody for this model, and how the *e-CoC*s are customized to be used on a small scale using public key infrastructure approach (*PKI*), where the role players and juries are the only people who are authorized to respectively publish and consume this *e-CoC*.

Keywords - Chains of Custody, Knowledge Representation; Provenance Vocabularies, Semantic Web, Linked Data Principles, Public Key Infrastructure, Kruse Model

I. INTRODUCTION

One of the most essential parts of the digital investigation process is the chain of custody (*CoC*). *CoC* is a chronological document accompanying digital evidences in order to avoid later tampering allegations. *CoC* records everything that happens to the evidence: who handled it, where, why, what, when, and how it was handled (known as 5Ws and 1H). There are three main reasons that motivate the transformation of the *CoCs* from tangible to electronic form:

- *Motivation 1*: cyber forensics is a daily growing field that requires the accommodation on the continuous changes of digital technologies as well as its tangible documents (i.e. concurrency with the knowledge management). Thus, tangible *CoCs* and all their contents (victim information and forensics information) must also undergo a radical transformation from paper to machine readable format in order to accommodate this continuous evolution.
- *Motivation 2*: judges' awareness and understanding the digital evidences are not enough to evaluate and take the proper decision about the digital evidence. Juries need to know more concerning the evidences in hand. One of the proposed solutions is to organize a syllabus and training program to educate the juries the field of Information and Communication Technology (ICT) [4]. The authors argue against this solution direction, because it will not be an easy task to teach juries with their juridical positions, the different concepts of ICT. The authors propose a solution offering the ability to the juries to navigate, discover (dereference) and execute different queries on the represented information.
- *Motivation 3: CoC*s play vital role in the investigation process that's why it must be maintained and managed throughout the investigation process in order to preserve its integrity, especially when the evidence has digital nature. However, if the *CoC* is not well maintained and the suspect was guilty, a lawyer/defense can argue that the *CoC* was not properly established and casting doubt on the damning of the acquired evidence. A security mechanism should be integrated with the represented data to keep its integrity and limit and control its access to only the authorized people.

Today, the semantic web is the web of data which is not just concentrated on the interrelation between web documents but also between the raw data within these documents. This data interrelation is based on four aspects known as LDP. In 2006, Berners Lee outlined set of rules/principles that are well structured for publishing data on the web. These principles explain that the data (content/resources) should be related one another just as documents are already [9][10]:

- Use URI as names for things and they are used as globally unique identification mechanism [7].
- Use Hyper Text Transfer Protocol (HTTP) as universal access mechanism so that people can look up those names [8].

- When someone looks up a URI, provide useful information using the standards (Resource Description Framework, SPARQL).
- Include RDF statements that link to other URIs so that they can discover related things (i.e. people locations, or abstract concepts).

Publishing data using this structured way can facilitate its consumption and help the consumer of this data to take the proper decision.

This paper resumes the works provided in [1][3]. The works in [1] launched the idea of exploiting LDP to represent the tangible *CoC*. The work in [3] provides a framework and lists 8 advantages for representing the *CoC* using LDP. Both works discussed how to apply this framework to one only forensic phase. However, this framework can be applied to a complete forensic process such as the Kruse model. Security approach such as Public Key Infrastructure (*PKI*) is also integrated to this framework to ensure the identity and the authentication of each role player participated in the forensics process. Current work explains in a simplified way how this framework can be used to produce *e-CoC*s for the phases of Kruse model using manually generated RDF code. However, the PKI approach provided in this paper is electronically presented and will be implemented in later publications. This work opens the door for cyber forensics experts to represent their forensics information in a structured way in order to be later consumed in an easy and understandable way by the juries in a court of law. Using the PKI approach to the Linked Data opens the door to a new era of research called the Linked Closed Data (LCD) instead of using the LDP only for the LOD.

The organization of this paper is as follows: next section discusses the state of the art of the semantic web, and the web of data. Section 3 explains the processes performed in Kruse model, and specifies different terms describing such processes in order to be later represented using LDP. Section 4 provides the solution framework and how the *e-CoCs* are represented by the role players, consumed by the juries in the court of law, and how the represented data is controlled and accessed by only those actors. Finally, the last section concludes and summarizes this work.

The related works in this paper are not presented in a separate section. However, they are mentioned in detail in [1], and through different references along the paper, especially in the explanation for each layer.

II. State Of The Art: Semantic Web and The Web of Data

Semantic web is an extension of the current web (i.e. from document to data) [11][12], designed to represent information in a machine readable format by introducing RDF model to describe the meaning of data and allows them to be shared on the web in a flexible way.

The classical way for publishing documents on the web is just naming these documents using URI and hypertext links. With the same analogy, entities and contents (data) within documents can be linked between each others using typed linked and with the same principles used by the web (i.e. web aspects). This is called the web of data. The Linking Open Data (LOD) project is the most visible project using this technology stack (URLs, HTTP, and RDF) and converts existing open license data on the web into RDF according to the LDP [9][10].

The LOD project created a shift in the community of research and development of the semantic web. Instead the concern was on the ontologies for their own sake and semantic, it becomes on the web aspects Ontologies are used then to foster and serve the semantic interoperability between parts that want to exchange such data. These are known as lightweight ontologies [13][20] that use the full advantages of semantic web technologies, minimum Ontology Web Language (OWL) [2] constructs, and reuse existing RDF vocabularies wherever possible.

RDF consists of three slots called triples: resource, property, and object. Also, resources are entities retrieved from the web (e.g., persons, places, web documents, pictures, abstract concepts, etc,). RDF resources are represented by uniform resource identifiers (URIs), of which URLs are a subset. Resources have properties (attributes) that admit a certain range of values or that are attached to another resource. The object can be a literal value or a resource.

While RDF provides the model and syntax for describing resources, it does not define the meaning of those resources and does not impose any interpretation on the kinds of resources involved in a statement beyond the roles of subject, predicate and object. That's where other technologies such as RDF Schema (RDFS) come in. The latter is a way of imposing a simple ontology on the RDF framework by introducing a system of simple types. RDFS enriches the basic RDF model, by providing a vocabulary for RDF. It develops classes for both resources and properties. However, RDFS is limited to a subclass hierarchy and a property hierarchy with domain and range definitions of these properties [6].

III. Kruse Model

Investigation models in the cyber forensics are numerous. Different cyber forensics models are proposed for the digital investigation process [14][15][16][17][18]. This paper uses the Kruse model [22] to illustrate how the LDP can be used to represent the *e-CoCs*, because it encompasses the three essential steps required by any cyber forensics investigation. The three phases of the Kruse model are: acquisition of the evidence, authentication of the recovered evidence, and analysis of the evidence.

Most works provided in the forensics models globalize the 5 Ws and 1H questions once over the whole forensics process. However, these questions must be asked separately over each phase of the forensics process, since each question for a given phase is different from the other.

Furthermore, each phase in a forensics model is accomplished by a role player responsible to prepare the *CoC* that answers the 5Ws and 1H questions for the forensics phase in hand. Figure-1 shows the role player participating in each phase of the Kruse model.

Figure 1. The Use Case diagram for Kruse Model

Before representing a *CoC* for a forensics phase, the terms that describe each phase should be specified in order to be defined later using the LDP. These specifications will be well achieved by the experts/analyzers of the cyber forensics domain (i.e. understanding of different processes and algorithms throughout the investigation tasks, and the resources such as forensics tools, role player).

TABLE I. THE *CoC*S OF THE KRUSE MODEL

Model	Phase	CoC Questions	Question Subject	Terms
Kruse	Acquisition	Who	The Role Player	First Responder
		What	Recovering and copying of data	File_allocation
		Why	Acquiring of Information for investigation	Evidence
		Where	The place that the acquisition task took place	Media / location
		When	The time that the authentication task took place	Date
		How	Procedures used to recovery and copy suspected data	Algorithm
	Authentication	Who	The Role Player	Investigator
		What	Verify the integrity of the acquired data	Evidence
		Why	Ensure the completeness & integrity of data	Hash
		Where	The place that the authentication task took place	Location
		When	The time that this task took place	Date
		How	Procedures used to perform the checking task	Algorithm
	Analysis	Who	The Role Player	Investigator
		What	Analysis of evidence type	Evidence
		Why	determine the significance and probative value of evidence through the logs verifications	Logs
		Where	The place that the analysis task took place	Media/location
		When	The time the analysis task took place	Date
		How	Procedures used for the analysis task	Algorithm

Acquisition: it is the operation of acquiring the evidence from suspect storage devices (e.g. hard disk, flash memory, digital camera). It starts by saving the state of the digital system under question so that it can be later analyzed. First responder is the role player of this activity. He is responsible to preserve the exact state that it was found [14]. Actually, the forensics analysis is not done directly on the suspect's device but on a copy instead. Thus, after preserving the scene' state, the role player performs two tasks: *recovering* and *copying*. Before copying the digital data from the suspected storage device to a trusted device, the deleted contents should be restored first. Later, copying the data from the suspect's device to another device (trusted) is performed to prevent tampering and alteration of the suspect's data on the digital device.

Authentication: it is the process of ensuring that the acquired evidence has not been altered and kept its integrity since the time it was extracted, to the time it was transmitted, and stored by an authorized source [19]. Any change to the evidence will render the evidence inadmissible in the court. Investigators authenticate the digital media by generating a checksum (Hash) of it contents (i.e. using the MD5, SHA, and CRC algorithms). Checksum is like an electronic fingerprint in that it is almost impossible for two digital media with different data to have the same checksums. The main aim behind this task is showing that the checksums of the seized media (suspected) and the trusted (image) are identical.

Analysis: this is the last and most time consuming step in this model. In this phase, the investigator tries to uncover the wrongdoing of the crime by examining the acquired data such as files and directories in order to identify pieces of evidence and determine their significance and probative value, and drawing conclusion based on the evidence found. In [5] the author defined the 3 major types of evidence that should be considered in the analysis phase:

- Inculpatory evidence: evidence that supports a given theory
- Exculpatory evidence: evidence that contradicts a given theory
- Evidence of tampering: evidence that is used to tamper the system to avoid the correct identification

Analysis of evidences must be accomplished without tainting the integration of the data.

IV. SOLUTION FRAMEWORK

The solution framework is imported from the work in [3]. Each forensics phase has its own *CoC*. In each phase, the player role is responsible to prepare and create the *CoC* of the phase in which he worked in. Each player role constructs his *CoC* using a web form that allows the player to import different resources (i.e. from the victim and forensics part) or create new triples using well predefined/custom vocabularies (see figure-2).

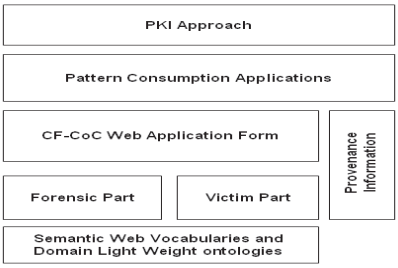


Figure 2. *Cyber Forensic-CoC* Framework [3]

The results will be a set of interrelated triples describing all phases in the forensics process. These triples are consumed by the juries in the court of law using different applications consumption patterns on the semantic web. Along this scenario, provenance dimension (metadata) is also integrated with the forensics data to answer all questions related to the origins of this data. The published data and its consumption will be published and consumed by the authorized people who are allowed to work on the current cybercrime case. PKI is used to ensure from the identities and authorization of each role player. Next sub sections describe each phase in details:

Figure-3 : Definition of first responder and responded term (Class and Property) [35]

A. *Semantic web vocabularies and domain light weight ontologies*

This framework uses two types of terms. The build-in terms that are predefined by the semantic web (e.g Friend-of-a-friend-FOAF, Dublin Core-DC, RDFS, OWL) and property vocabularies constructed to describe certain domain when the existing terms are not sufficient. The terms specified in table-1 are created using the lightweight ontologies [13][20] (e.g. RDF, RDFS, OWL, FOAF). Figure-3 shows an example of how a *CoC*' term was defined using lightweight ontologies. The "first-responder" term is defined as a property term (*rdf: type* and *owl#objectProperty*) and its range (*rdfs:range*) is the First-responder class (*rdf: type* and *rdf-schema#class)* which is a subclass (*rdf-schema#subClassOf*) of the Person class (*foaf:person*). This term is also the inverse (*owl:inverseof*) of the "responded" verb which has the domain (*rdfs:domain*) of First-responder class and range (*rdfs:range*) of the Acquisition class.

B. *Victim and Forensics Parts*

Once the vocabularies needed to describe all *CoCs* terms are created, the forensics part can use them to describe and represent their information. The paper assumes that the victim already has a published data describing their profiles and resources. The role player integrates the victim data with the forensics information by the support of different vocabularies (built-in and property) to construct and represent the *CoC* of each phase (figure-4). The victim and forensics parts are able to publish their information when they own a unique domain/namespace minted by unique URL. This URL is used to describe different resources using extended hash(#) URIs e.g. http://cyberforensics-coc.com/vocab/authentication#MD5 and to make such resources dereferenceable (i.e. HTTP clients can look up the URIs using the HTTP protocol and retrieve a description about the resources that are identified by these URIs). For example, the MD5 can be described by the "Hash" property term specified in table-1 (authentication phase). Also, the Hash term can be defined using lightweight ontology.

Furthermore, the integration of the forensics data resulted from the investigation process (i.e. analysis phase) can be realized using the Advanced Forensic Format (AFF4). It is an open format for the storage and processing of digital evidences [21]. The great advantage of this format is to represent different forensics metadata in the form of RDF triples (subject, predicate, and value), where the subject is the Unified Resource Name (URN) of the object the statement is made about, and the predicate (e.g. datelogin, datelogout, evidenceid, affiliation, etc,) can be any arbitrary attribute which can be used to store any object in the AFF4 universe.

Figure-4 provides the *e-CoC* of the authentication phase: What: evidence (RDF), Why: Hash (custom), Where: location (AFF4), When: date (DC), and How: MD5 (custom) are all integrated together in a unified framework answering the six questions of this phase. The big difference between the terms defined in the AFF4 format and the customs terms is that the latter are URI resources that can be dereferenced while the former are set of literals that are terminals. The higher the number of dereferenceable terms, the more the data provided to juries are descriptive.

C. *Cyber Forensic-CoC Web Application Form*

This section provides the design principles that should exist on a web application to generate complete *CoC* RDF triples and connect them in order to be later consumed and crawled by juries. The web application is designed to:

- Import resources from the forensics parts (e.g. role players profiles, results of forensics investigation)
- Import resources from the victim part (departments namse, employees names, machines IDs, etc,)
- Create and describe resources by the support of :
  - Existing terms imported from well established vocabularies.
  - New terms imported from custom vocabulary create to describe the *CoC* for each forensics phase.
  - Adding provenance metadata to the forensics data (i.e. provenance vocabularies are used to improve the origin of the data imported)

Two languages can be used to generate the RDF models from the data entered in the web form application: script language or mapping language. In both cases, the data is posted first in a relational database and queried or mapped later. So, for instance, the PHP script language can generate linked data in RDF/XML format by the help of the ARC library for working with RDF in PHP [23].

Figure-4 : *e-CoC* for Authentication phase (NG_{Auth}) [35]

On the other hand, the D2RQ mapping language [24][25] is used to map database content into RDF vocabularies and OWL ontologies and it allows the RDF data to be browsed and searched [9].

D. Pattern Consumption Applications

As mentioned in the last section, the first way to publish linked data on the web is to make URIs that identifies data items dereferenceable into RDF descriptions. Three main patterns can be used by juries to consume this information about the *CoC* published by role players: browsing, searching, and querying. Browsing is like the traditional web browsers that allow users to navigate between HTML pages. Same idea applied for linked data, but the browsing is performed through the navigation over different resources by following RDF links and downloads them from a separate URL (e.g. RDF browsers such as Disco, Tabulator, or OpenLink Browser). A custom semantic specialized for the juries can be easily created [26].

RDF Crawlers are developed to crawl linked data from the web by following RDF links. Crawling linked data is a search using a keyword related to the item in which juries are interested (e.g. SWSE and Swoogle). Juries can also perform extra search filtering using query agents. This type of searching is performed when SPARQL endpoints are installed, allowing expressive queries to be asked against the dataset. Furthermore, a void vocabulary (vocabulary of interlinked datasets) [27] contains a set of instructions that enables the discovery and usage of linked datasets through dereferenceable HTTP URIs (navigation) or SPARQL endpoints (searching), using SPARQL protocol (*void:sparqlendpoint*) or URI protocol.

E. Provenance Metadata

CoC data source should include provenance metadata together with the forensics data. Such metadata can be used to give the juries data clarity about the provenance, completeness, and timeliness of the forensics information and to strength the provenance dimension for the published data.

Provenance information can be integrated within the forensics data using three different methods. The first method is using provenance vocabularies of the semantic web. The second one is to use open provenance model [28], and the last method is by exploiting Named Graph (NG) for RDF triples, to add provenance metadata about each group of triples [32].

The most widely method is to combine the provenance metadata with named graph. A widely deployed provenance vocabulary is the Dublin Core [31]. This vocabulary contains different predicates that can provide extra information related to the forensics data like the *dc:creator*, *dc:publisher,* and *dc:date*.

The idea of the named graph is to take a set of RDF triples (i.e. Graph) and name this graph with a URI reference. Figure-5 provides an abstract diagram depicting the grouping of triples and naming them to a graph with the integration of provenance metadata (e.g. DC). The NG_{Auth} in figure-5 is the *e-CoC* of the authentication phase provided in figure-4. Each phase contains inner and outer links that relate all *CoCs* to each others.

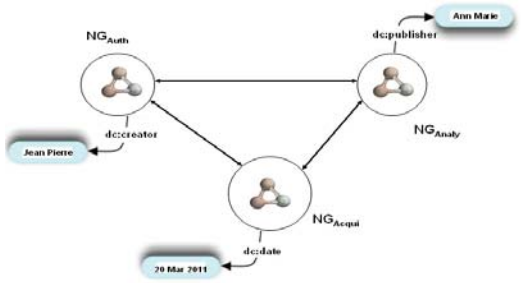


Figure-5: Named graph for Kruse Model

It is useful for juries to navigate using applications and access provenance metadata related to certain set of triples and get more description about them (e.g. LDspider [34] allows crawled data to be stored in an RDF store using the named graphs data model). As the SPARQL is widely used for querying RDF data, juries can also use the SPARQL query single or sets of named graphs and access the provenance information related to different graph.

F. PKI Approach

Provenance metadata are not enough to ensure that the published data belong to the right. PKI approach allows juries to ensure from the identity of role players

participated in the forensics investigation. Each player in the forensics process should have his own certificate which contains information about his identity and his digital signature. A digital certificate alone can never be proof of anyone's identity. A third trusted party is needed to confirm and sign the validity and authority of each player certificate. This party is then called certification authority (CA).

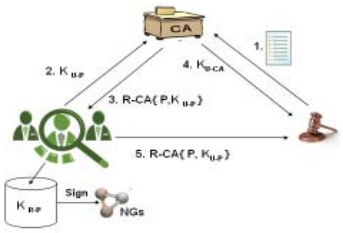


Figure-6 Application of PKI in *CoC* Representation

Any certificate contains pieces of information about the identity of the certificate owner (role player) such as owner's distinguished name, owner's public key, and information about the CA (issuer of certification) such as CA's signature of that certificate, and general information about the expiry date and the issue date of the certificate.

Figure 6 shows in a sequence of steps, how the juries verify the identity, and how the PKI certifications are applied in this context:

1. Juries send a list of players who are supposed to work on the current cyber crime case.
2. The role player generates a public-private key pair ($\{K_{U\text{-}P}, K_{R\text{-}P}\}$), where P is all information identifying the player, R is private, and U is public. Players store the private key in a secure storage to keep its integrity and confidentiality, and then send the public key $K_{U\text{-}P}$ to the CA.
3. The player's public key and its identifying information P are signed by the authority using its ($\{K_{R\text{-}CA}\}$) private key. The resulting data structure is back to the role player, R-CA $\{P, K_{U\text{-}P}\}$ is called the public key certificate of the role player, and the authority is called a public key certification authority.
4. Juries obtain the authority's public key $\{K_{U\text{-}CA}\}$
5. Each player creating a *CoC* must authenticate himself to the juries by signing his RDF graph NG using his private key R-P{NG} (i.e. all triples describing a phase are assembled in one graph called NG_{Auth}, NG_{Acq}, and NG_{Analy}). Later, before the court session, each player sends the certification R-CA $\{P, K_{U\text{-}P}\}$ to juries accompanied with the signed graph R-P{NG}[33].

The main idea behind this scenario is based on the public key cryptography, where the senders (role player and CA) make signature using their private key and the jury verifies these signature using their public key. Applying PKI to the LDP emerge a new era of research for the Linked Closed Data, where access restrictions to the URLs are applied. A new work will be published to solve the compromise between the URLs access restriction (i.e., especially the external links) and their consumption using the different patterns interface.

I. Conclusion

This paper discussed the idea of representing the tangible *CoC* using the LDP through the Kruse model. This paper provides design options to construct CF-*CoC* system. It answered the three motivations provided in section I:

1. The *CF-CoC* framework is based on the RDF model which enables the encoding and reuse of structured metadata, so it can be exchanged between role players, juries, and machine without the loss of meaning. Also, the vocabularies (predefined, property) accompanying this model enrich the represented resources with semantic interpretation that are useful to take proper decision about the case in hand (motivation 1).
2. The web aspects used by the LDP allow the creation of dereferenceable resources which allow the juries to navigate between resources, retrieve additional information about the represented data, and execute different queries using SPARQL language (motivation 2).
3. The PKI mechanism is used to secure the RDF model by signing the RDF graph associated to each phase, and limit the access only to the authorized people (motivation 3).

This paper opens the door to construct a system allowing role players to transform the tangible *CoC*s to *e-CoC*s, and helping juries to easily consume and understand the different results of investigation provided to them even their technical knowledge are not sufficient about the field of ICT [4]. The validation of a system based on the RDF is to check the syntax of RDF generated from this system. This type of validation will be held after constructing all the modules of the CF-*CoC* framework. However, the paper uses a manually generated RDF graph to show that the LDP are very suitable for representing the *CoCs* of the forensic process.

References

[1] T. F. Gayed, H. Lounis, and M. Bari, "Computer Forensics: Toward the Construction of Electronic Chain of Custody on the Semantic Web," . SEKE 2012 pp: 406-411

[2] F. v. Harmelen. OWL Web Ontology Language Overview - W3C Recommendation. http://www.w3.org/TR/2004/REC-owl-features-20040210/, 2004.

[3] T. F. Gayed, H. Lounis, and M. Bari, "Cyber Forensics: Representing and Managing the Tangible Chain of Custody using the linked data principles," The Fifth International Conference on Advanced Cognitive Technologies and Applications 2013, Valencia, Spain

[4] G. C. Kessler, "Judges' Awareness, Understanding and Application of Digital Evidence," Phd Thesis in computer technology in Education, Graduate school of computer and information sciences, Nova Southeastern University, 2010

[5] B. Carrier and B D, “Carrier, Defining Digital Forensic Examination and Analysis Tool Using Abstraction Layers,”. IJDE 1(4) (2003)

[6] Linked data : Evolving the web into a global data space, http://linkeddatabook.com/editiopns/1.0/

[7] T. Berners-Lee, R. Fielding, and L. Masinter, “RFC 2396 – Uniform Resource identifieres (URI): Generic Syntac. http://www.isi.edu/in-notes/rfc2396”, August 1998

[8] Roy fielding. Hypertext transfer protocol – http/1.1 request for comments

[9] M. Campbell and S. MacNeill, “The semantic web, Linked and Open Data, A Briefing paper,” June 2010, JISC CETIS

[10] L. Berners-Lee, T: Design issues: Linked data. Retrieved Mar. 19, 2010, from http://www.w3.org/DesignIssues/LinkedData.html

[11] T. Berners-Lee, J. Hendler, and O. Lassilia, “The semantic web. *Scientific American*,” 284(5):34–44, Mai 2001. http://dx.doi.org/10.1038/scientificamerican0501-34DOI: 10.1038/scientificamerican0501-34

[12] C. Bizer, T. Heath, and T. Berners-Lee, “Linked data - the story so far. Int. J. Semantic Web,” Inf. Syst., 5(3):1–22, 2009. http://dx.doi.org/10.4018/jswis.2009081901DOI: 10.4018/jswis.2009081901

[13] F. Giunchiglia and I. Zaihrayeu, “University of Trento, Italy. Technical Report,” October 2007

[14] Technical Working Group for Electronic Crime Scene Investigation, Electronic Crime Scene Investigation : A Guide for first responders, United States Department of Justine, 2001

[15] E. Casey, “Digital evidence and computer Crime,” Elsevier Academic Press, 2004

[16] S. O. Ciardhuain, “ An extended model of cybercrime investigations,” International Journal of digital Evidence, Vol. 3, 2004

[17] Y. Yusoff, R. Ismail and Z. Hassan, “common phases of computer forensics investigation models”. International Journal of computer science and information technology (IJCSIT), Vol 3, No 3, June 2011.

[18] M. Köhn, J. Eloff and M. Olivier, "UML Modeling of Digital Forensic Process Models (DFPMs)," *Proceedings of the ISSA 2008 Innovative Minds Conference*, Johannesburg, South Africa, July 2008 (Published electronically)

[19] S. Vanstone, P. Van Oorschot,, & A. Menezes, Handbook of Applied Cryptography“, CRC Press, 1997

[20] B. Glimm, A. Hogan, M. Krötzsch, and A. Polleres, “OWL: Yet to arrive on the Web of Data?,” April 2012, Lyon, France abs/1202.0984 (2012)

[21] M. cohen, S. Garinkel, and B. Schatz “Extending the advanced forensic format to accommodate multiple data sources,” logical evidence, arbitrary information and forensic workflow. Digital Investigation”, 2009. S57-S.

[22] W. Kruse and J. Heiser, “Computer Forensics: Incident Response Essentials. Addison Wesley,” 2002

[23] http://arc.semsol.org/

[24] http://sites.wiwiss.fu-berlin.de/sihl/bizer/d2r-server/index.html

[25] http://d2rq.org/d2rq-language

[26] Dennis Quan, David R. Karger. “ How to make a semantic web browser” May 2004, ACM, New York, USA

[27] Latest version provided http://www.w3.org/TR/void/

[28] http://purl.org/net/opmv/ns

[29] O. Hartig, J. Zhao, “Publishing and Consuming Provenance Metadata on the Web of Linked Data,” IPAW 2010: 78-90

[30] O. Hartig, “Provenance Information in the Web of Data,” LDOW 2009

[31] http://dublincore.org/documents/dcmi-terms/

[32] J. J. Carroll, C. Bizer, P. Hayes, and P. Stickler, “Named graphs, provenance and trust,” In WWW '05: Proceedings of the 14th international conference on World Wide Web, pages 613{622, New York, NY, USA, 2005. ACM Press. (doi: http://doi.acm.org/10.1145/1060745.1060835).

[33] J. J. Carroll. Signing RDF Graphs. In 2nd ISWC, volume 2870 of LNCS. Springer, 2003.

[34] R. Isele, A. Harth, J. Umbrich, and C. Bizer. Ldspider “An open-source crawling framework for the web of linked data.,” In ISWC 2010 Posters & Demonstrations Trach: Collected Abstracts Vol-658,2010

[35] Figures generated using : http://www.w3.org/RDF/Validator/

ARGUMENTATION UNDERSTOOD AS PROGRAM SYNTHESIS

Ashwag Omar Marghraby and Dave Robertson
Centre for Intelligent Systems and their Applications
School of Informatics, The University of Edinburgh,
Edinburgh, UK
A.O.Maghraby@sms.ed.ac.uk

***Abstract*— A gap exists between the specification of argument/argumentation norms and their implementation via protocols in multi-agent systems. This paper describes a new approach to support and automate the synthesis (code generation) of multi-agent argumentation protocols. The idea is to automatically transform a high-level description of an argumentation dialogue game provided by an argumentation user to a final multi-agent executable protocol based on reusable and parameterized patterns. This brings pattern based program synthesis into multi-agent programming in a new way.**

***Index Terms*— Program Synthesis; Automated Synthesis; Dialogue game; Argumentation.**

I. INTRODUCTION

Generally, the argumentation community views arguments in a formal way, where a Multi-Agent Systems (MAS) argumentation dialogue game [1] is defined using abstract specification languages such as the Argument Interchange Format [2] (AIF is a generic speciation language for argument structure) which abstracts the game from its technical details. Although arguments may be specified precisely using abstract specification languages, this is not a practical way to implement MAS.

In practice, abstract specification languages are not executable languages and in the software engineering community, in order to build complete multi-agent argumentation systems, there is a need to produce concrete implementations in which these abstract specifications are realized via protocols coordinating agent behavior. This creates a gap between standard argument specification and implementation of MAS protocols.

So far in the software engineering community, there are two ways of addressing this issue. The first way is to ignore the abstract specification representation of the dialogue game and implement (re-build) the MAS from scratch. The second way is through the automated program synthesis [3] (in which an automated program synthesis tool transforms a high-level specification into an executable code). Designing an automated program synthesis method involves three steps: (1) selecting or creating a new abstract language (which acts as a high-level language) that allows particular types of argumentation to be defined; (2) selecting a MAS protocol implementation language (which acts as an operational or low-level language) that is used for specifying the message-passing behavior of MAS interaction protocols; (3) selecting an approach to automated program synthesis and developing a system that is able to interpret the abstract language to produced MAS protocol implementation language.

This paper attempts to solve the engineering problems of MAS argumentation systems by using a combination of automated program synthesis (code generation) and verification methods. More precisely, this paper proposes a means of moving rapidly from dialogue game argument specification to protocol implementation using an automated synthesis method.

The paper is organized as follows. Section II represents the argumentation dialogue game abstract language (the first step toward automation synthesis of MAS protocol). Section III presents the MAS protocol implementation language (the second step toward automation synthesis of MAS protocol). Section IV illustrates the automated program synthesis approach (the third step toward automation synthesis of MAS protocol). Section V presents a verification approach which is used to evaluate the approach. Section VI contains the conclusion.

II. ABSTRACT LANGUAGE

The first step toward the automation of the MAS protocol is the selection or creating of new abstract language to represent argumentation dialogue game rules. The existing abstract specification languages in the argumentation community (e.g. AIF [2]) enable users to structure arguments using diagrammatic linkage between natural language sentences. However, these languages do not capture some concepts that are needed to support the interchange of arguments between agents (e.g. the sequence of argument or message). For that reason, this research created a new, intermediate recursive visual high-level language called Dialogue Interaction Diagram (DID) to represent, in an abstract way, the multi-agent dialogue game interaction protocol rules between two agents by allowing the designer to specify the permitted messages and their relationship to each other.

DID is a recursive visual language which restricts agents' moves to: (1) Unique-moves: agents can make just one move before the turn-taking shifts, and agents can reply just once to the other agent's move; (2) Immediate-reply moves: the turn-taking between agents switches after each move, moving from one level to the next level, and each agent must reply to the move of the previous agent.

This restriction is quite strict but it still allows us to include a large class of argumentation systems in our synthesizer. For instance, all argumentation systems that can be described as dialogue games. In general, we can synthesize arguments that can be described as a sequence of recursive steps (each of which involves turn taking between the pair of agents) terminating in a base case.

DID has nine interaction protocol rules concepts [4]: (1) Locutions rules: represent the set of permitted messages; (2) Commitment rules: define the propositional commitments made by each participant with each message during the dialogue; (3) Structural rules: define legal messages in terms of the available messages that a participant can select to follow on from the previous message; (4) Turn-Taking: specifies the next player; (5) Starting rules: define the conditions to begin the dialogue; (6) Termination rules: define the conditions to end the dialogue; (7) Pre-condition rules: define the conditions under which the message will be achieved; (8) Post-condition rules: define the conditions which must always be true just after the locution utterance.; (9) Sender and receiver agents' roles: a set of functions that an agent can use to interact with another agent. Each role identifies the messages that an agent can send or receive.

Space limitation prevent a discussion in depth of how these are represented. Instead, in the next subsection, we move to a summary of the (formal) visual notation we use to capture these concepts.

DID ELEMENTS

The basic element of every DID is a message (locution) icon. A message icon (as shown in Fig.1) is simply a rectangle divided into three sections. The topmost section contains the name of the message. The left section contains sender agent attributes (Role name, Role arguments, and Agent ID), and the right section contains receiver agent attributes (Role name, Role arguments, and Agent ID). A rhombus shape represents conditions which apply to each message; when connected to the left section it represents sender agent pre-conditions and when connected to the right section it represents receiver agent post-conditions. Dotted rectangles represent the message type: Starting (can be used to open a dialogue game), Termination (can be used to terminate the dialogue game), and Recursive locution (can be used to remain in the dialogue game). A DID is created by linking the message icons together. The links between the message icons represent reply relations between arguments. Please see papers [5,6] for more detail about the DID elements (syntax) and examples.

III. PROTOCOL IMPLEMENTATION LANGUAGE

The second step toward the automation synthesis of the MAS protocol is the selection of a MAS protocol implementation language. To support formal analysis and verification, we choose as our protocol implementation language the LCC [7]. LCC is a declarative [8], process [9] calculus-based, executable specification language for choreography [10] which is based on logic programming and is used for specifying the message-passing behavior of MAS

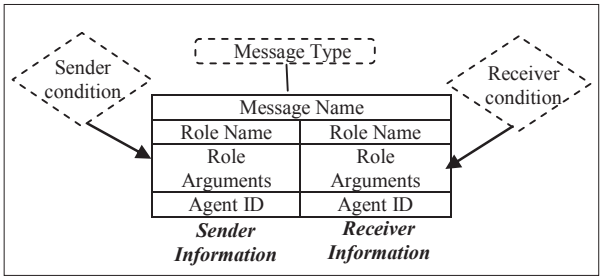


Fig.1. Locution icon

interaction protocols. See paper [7] for more information about the abstract syntax of LCC.

IV. AUTOMATED PROGRAM SYNTHESIS METHOD

The third step toward the automation synthesis of the MAS protocol is the selection of an approach to automated program synthesis as well as the development of a system that is able to interpret the abstract language as a MAS protocol implementation language. One approach used on order to automate program synthesis is a structured synthesis approach (pattern-based). The main idea of the structured synthesis approach is to capture information required for the transformations as reusable and parameterized patterns. These patterns can be used to apply the transformations through an automated program synthesis tool. Patterns [11], are generic, recurring solutions to common problems, defined in the target language (LCC in this paper) and capture the overall structure of a task, having been extensively studied within the software engineering community.

In the following subsections, we summarize some protocol patterns (LCC-Argument patterns) that can be embedded in the automated synthesis tools and used with DID to support the MAS protocol development activity. The reason for introducing protocol design patterns in argumentation is that by re-using them it is possible to reduce the effort of building dialogue game argumentation agent protocols.

A. LCC- Argument patterns

The patterns described in this paper are called LCC-Argument patterns. These patterns are similar to object-oriented design patterns (in the software engineering community). The only difference between them is the structure of an LCC-Argument pattern which is described by using the notions of roles instead of the notions of classes and objects. We use the notation of the roles since our protocol language is LCC which is not considered to be an object-oriented language and uses roles (instead of classes and objects) to describe MAS protocols.

LCC-Argument patterns capture the different relationships and interactions between LCC agents' roles. These patterns provide a common LCC argument code for developing protocols and their components along with defining how two or more agents can interact with each other. They are generic solutions to the common LCC argumentation protocol development problem that recur repeatedly across protocols and can be adapted to generate specific protocols. Maghraby

[11] describes these patterns in detail. To expedite our argument, we will not repeat these here. Instead we will describe the uses of two LCC-Argument patterns which describe the MAS interaction protocol between two agents:

1) *Starter pattern* : This pattern is used to start the dialogue between two agents (*A1* and *A2*). It is composed of two roles: sender role, *RoleOneA1*, and receiver role, *RoleOneA2*.The general idea of this pattern is that the agent with role *RoleOneA1* sends a starting message to the agent playing role *RoleOneA2* and then both agents change their roles (see Fig.2).
2) *Termination-Intermediate Pattern*: This pattern is used to send/receive a message(s) to terminate the dialogue game (interaction between agents *A1* and *A2*) or to change agents' roles**.** It is composed of two roles: sender role, *RoleOneA1*, and receiver role, *RoleOneA2*. The general idea of this pattern is that the agent with role *RoleOneA1* sends:
   a) a termination message (TM), to the agent playing role *RoleOneA2*, to terminate the dialogue game;
   b) or sends a recursive message (RM) (step b.1), to the agent playing role *RoleOneA2*, and then both agents change their roles (step b.2) (see Fig.3).

   The sender agent in this pattern can send one or more message to the receiver agent. This pattern handles this by using refinement (rewriting) methods [12].

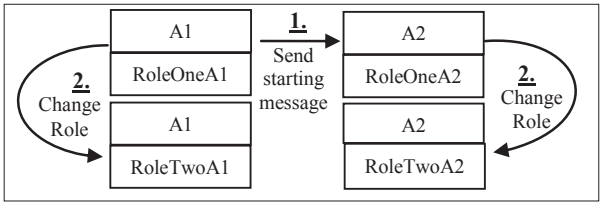


Fig. 2. Starter Pattern

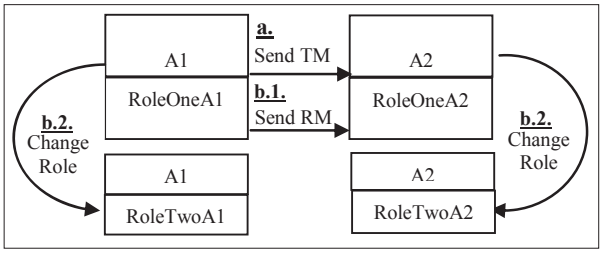


Fig. 3. Termination-Intermediate Pattern

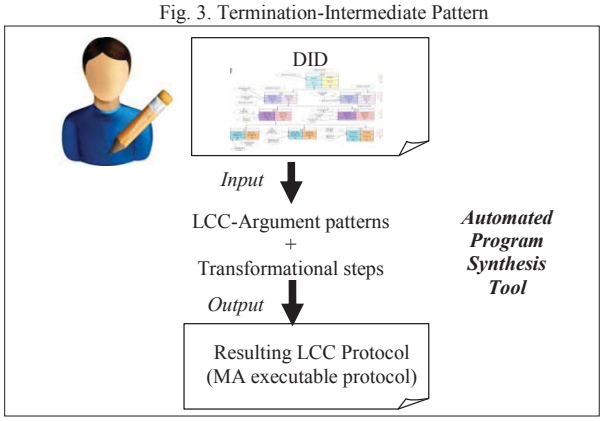


Fig. 4. Automated Program Synthesis Method

1. Input (DID, LCC-Argument patterns)
2. Select&Save Icon= one DID locution icon **(Step1)**
3. Select&Save Pattern= one pattern from the LCC-Argument patterns library **(Step2)**
4. If (Pattern has rewriting methods) then **(Step3)**
5. If (level has one message icon) then
6. Select&Save RewriteMethod=*Rewrite 1*
7. If (level has more than one message icon) then
8. Select&Save RewriteMethod=*Rewrite 2*
9. Match (Icon,Pattern,RewriteMethod) **(Step4)**
10. Go To line 2 **(Step5)**
11. End matching all level in the DID with the corresponding patterns
12. Output LCC protocol

Fig.5. Two Agents' Protocol Automated Synthesis Algorithm

B. Automated Program Synthesis Steps

This section illustrates the outline of our method for bridging the gaps between argument specification and implementation of MAS protocol languages as depicted in Fig. 4.

The user draws a DID diagram of the dialogue game. Then, the automated program synthesis tool transforms this diagram into an LCC MAS interaction protocol (an executable protocol) by applying predefined and automated transformational steps to the DID diagram. More precisely, the tool matches DID diagram icons with LCC-Argument patterns (parameterized patterns) to create a LCC MAS interaction protocol.

Patterns capture the overall structure of the interactions between agents and are defined in the LCC language. A DID diagram (specified by the user) specifies information required for fitting and customizing the pattern for a specific dialogue game. Transformational steps use the DID diagram to fit the patterns and generate the LCC MAS interaction protocol. We shall now describe the automated transformation steps in further detail.

AUTOMATED TRANSFORMATION STEPS FOR GENERATING AN AGENT PROTOCOL BETWEEN TWO AGENTS

The automated synthesis process of the two agents' protocol consists of five steps (The two agents' protocol automated synthesis algorithm is illustrated in Fig.5).

AUTOMATED TRANSFORMATION STEPS FOR GENERATING AN AGENT PROTOCOL BETWEEN N-AGENTS

The DID notation is general across all dialogue games but is limited to two agents. Our design patterns extend to N-agents but only accommodating those protocols that fit the patterns, so we lose the DID generality. However, we can reclaim some generality for patterns in which parts of the protocol are dialogues. Our idea is to consider the dialogue among N-agents as a dialogue between two agents by dividing agents into groups composed of two agents under certain

conditions. Our automated synthesis method uses an LCC-argumentation broadcasting pattern [11] to divide agents into groups composed of two agents and then it follows the general automated synthesis process of two agents protocol to generate the LCC protocol for DID for two agents, which allows the selected pairs of each group to communicate with each other.

V. Verification Method

Automated protocol synthesis (pattern-based synthesis) is complex. It requires many steps (e.g. profound knowledge of agent protocols, understanding of dialogue games and LCC language) and large amounts of time to define a correct set of patterns and adding new patterns risks introducing errors into the synthesizer. Therefore, this paper also presents a verification method based on Standard functional programming language (SML) [13] and Coloured Petri Net (CPNs) [14] to verify the semantics of the DID specification against the semantics of the synthesized LCC protocol. Space limitations prevent us from giving details of this but we will sketch the main elements here.

Given the DID and the LCC MAS interaction protocol, our verification tool could answer the question: Does the LCC specification satisfy the DID behavior properties? To answer this question, the tool performs the following tasks (see Fig.6): (1) Given the DID as an input, the automated verification tool extracts the DID properties using SML specification transformational steps; (2) Given the LCC MAS interaction protocol as an input, the automated verification tool transforms the LCC protocol into an equivalent CPNXML file using a set of transformational rules. The generated CPNXML file can then be used to construct the state space. From the state space the automated verification tool extracts the behavioral properties of the LCC protocol; (3) The tool compares the DID properties and the behavioral properties of the LCC protocol by using CPN SML functions. A positive (negative) result indicates that a specific property is satisfied (unsatisfied). For more details please see [5].

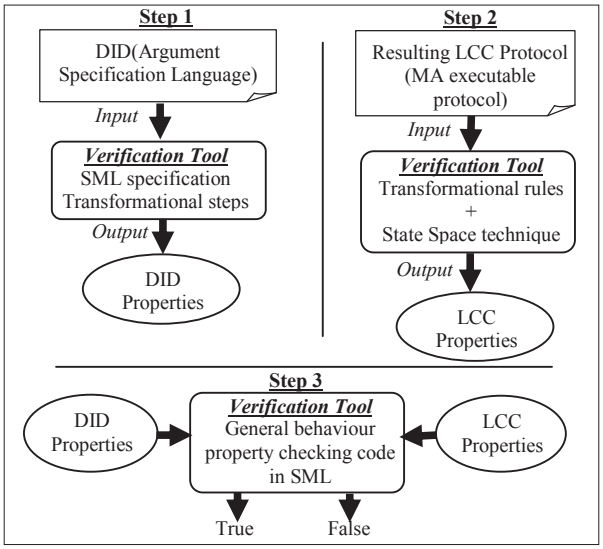


Fig.6. Verification Process

VI. Conclusion

This paper describes an approach to bridging the gap between dialogue game argument specification and multi-agent implementation using DID as an example of a dialogue game argumentation language and LCC as an example of a multi-agent implementation (coordination) language. The proposed approach is based on a new specification dialogue game argument language (DID) and reusable, parameterized transformation patterns.

Using our approach reduces the time in moving from abstract specification to implementation and gives greater assurance for the semantic equivalence between the DID specification and the resulting LCC protocol (which is executable via LCC interpreters).

Although the resulting synthesis and verification system is not an industry-strength specification tool, it demonstrates how automated synthesis methods can connect argumentation to MAS interaction protocols in a process language. Potentially, this could allow developers of argumentation systems to use specification languages to which they are accustomed (in our case DID) to generate systems capable of direct implementation on open infrastructures (in our case LCC).

References

[1] D. Walton, "What is reasoning? what is an argument?," the journal philosophy**,** vol.87, pp. 399-419, 1990.

[2] C. Chesnevar et al., "Towards an argument interchange format". The Knowledge Engineering Review*,* vol.21, pp.293–316,2007.

[3] S. Gulwani, "Dimensions in program synthesis," in PPDP, 2010.

[4] H. Prakken, "Formal systems for persuasion dialogue," The Knowledge Engineering Review, vol.21, pp.163-188, 2006.

[5] A Maghraby, "Automatic Agent Protocol Generation from Argumentation" in 13th European Agent Systems Summer School, Girona, Catalonia (Spain), 2011.

[6] A Maghraby, D. Robertson, A. Grando and M. Rovatsosr, "Automated Deployment of Argumentation Protocols," in Fourth International Conference on Computational Models of Argumen, 2012.

[7] D. Robertson, "Multi-agent coordination as distributed logic programming". in 20th International Conference, 2004.

[8] J. Lloyd, ″Practical advantages of declarative programming," in Joint Conference on Declarative Programming, GULP-PRODE'94, 1994.

[9] J. Baeten, "A Brief history of process algebra," Theoretical Computer Science, vol. 335, pp. 131-146, 23 May 2005.

[10] R. Dijkman and M. Dumas, "Service-oriented design: A multi-viewpoint approach," International Journal of Cooperative Information Systems, vol.13, pp.337-378, 2004.

[11] A. Maghraby, "LCC argument patterns," School of Informatics, Edinburgh university, 2011. Unpublished. Available from: http://homepages.inf.ed.ac.uk/s0961321/index.html

[12] C. Morgan, Programming from specifications, 2nd ed., Prentice Hall International (UK) Ltd., 1994.

[13] R. MILNER, M. TOFTE, R. HARPER and D. MACQUEEN, The definition of Standard ML, revised ed., Cambridge, 1997.

[14] K. Jensen, L. Kristensen and L. Wells, "Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems," International Journal on Software Tools for Technology Transfer (STTT), vol. 9, pp. 213–254, 2007.

Virtual Medical Board: A Distributed Bayesian Agent Based Approach

Animesh Dutta
Dept of IT
NIT Durgapur
India
animesh.dutta@it.nitdgp.ac.in

Sudipta Acharya
Dept of IT
NIT Durgapur
India
sudiptaacharya.2012@gmail.com

Aneesh Krishna
Dept. of Computing
Curtin University of Technology
Western Australia
a.krishna@curtin.edu.au

Swapan Bhattacharya
Dept of CSE
NIT Surathkal
India
director@nitk.ac.in

***Abstract*—Distributed Decision Making has become of increasing importance to get solution of different real life problems. Application of agent and multi agent system in this Distributed Decision Support System is an evolving paradigm. One of such real life problem is medical board formation. But always formation of a medical board with a group of expert physicians may not be always possible due to lack of infrastructure, availability, time etc. In these situations the role of multi agent based distributed decision making can comes into play. In this paper we develop a Virtual Medical Board System in which a number of software agents (expert agents) act as a group of expert physicians with knowledge base(KB), reasoning capability. They coordinate with each other to diagnose a patient.**

***Index Terms*—Vitual Medical Board, Bayesian Network Of Bayesian Agent, Multi agent system, Expert agents, Coordination Ontology, Distributed decision support system.**

I. Introduction

Proper medical treatment starts with proper medical diagnosis. Accordingly, doctors are trained to look for certain medical conditions when specific symptoms are presented by patients. When those symptoms are missed and a condition goes undiagnosed, the potential consequences can be fatal. While diagnosing a patient, sometimes it happens that the doctor is not able to reach any final conclusion regarding the disease and its treatment plan by himself, then he may need to consult with his fellow colleagues or some expert in that field. The group formed by these expert physicians is called Medical Board. Members of Medical Board collectively discuss with each other to diagnose a patient properly. Medical board formation is costly method and it needs a developed infrastructure in the hospital. That is why it is impossible to form medical Board to diagnose a patient in rural hospital which have weak infrastructure. So to provide a sophisticated medical facility like medical board in rural areas, a virtual medical board can be formed.

An agent [1] , [2] is a computer system or software that can act autonomously in any environment to achieve a goal. Multi-agent systems (MAS) [1] , [2] are computational systems in which two or more agents interact or work together to perform a set of tasks or to satisfy a set of goals.

Multi agent system (MAS) based decision support system [3] is a system where number of software agents take a decision of a given problem collectively. In these situations, each agent plays role of a human entity in human-based group discussion methodology.

Multi agent based distributed decision support system is the key idea in the formation of virtual medical board. Here number of software agents act as group of expert physicians forming medical board. In our system we represent the knowledge base(KB) of expert agent in the form of Bayesian network.

A Bayesian network or Bayes network or belief network [4] is a probabilistic graphical model. As medical diagnosis is a probabilistic method so in our system we use Bayesian network to represent the knowledge base of each expert agent by specifing probabilistic relationships between diseases and symptoms.

Ontology [5] is the most suitable representation of domain knowledge because concepts, relationships and their categorizations in a real world can be represented with them. With the concept of Ontology, we can say that coordination Ontology is the domain specific Ontology which contains some rules and methodologies about how to take an ultimate decision by a number of decision maker by resolving different obstacles/conflicts during group discussion.

II. Scope of Work

There have been a lot of contributions in the area of agent based decision support system in Medical Diagnosis. But most of them are centralized-knowledge based or single diagnostic agent based in nature. That is depending on single medical ontology of a diagnostic agent diagnosis of a patient is done[[6], [7], [9], [10], [13], [14]]. There is no human oriented medical board like group discussion mechanism to diagnose a patient. Some of the papers are on distributed decision support system. But here a main problem is divided into number of subproblem[[8], [11], [12]], where each agent deals with different subpart of the main problem. Here it is not proposed how to take a final decision if number of agents are dealing with the same problem. In our work we propose a system which is virtualization of medical board. Here number of software agents with knowledge base act as a group of expert physician. Where each physician handles same patient with same symptoms, diagnoses the patient independently, take diagnosis decision, communicate with each other to share their

decisions. Finally ultimate diagnosis decision is taken with the help of a coordination Ontology.

III. System model

A. System Architecture

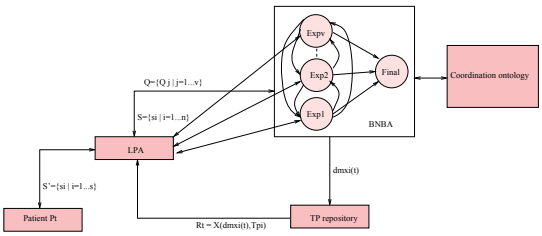


Fig. 1: Architecture of the proposed System

Figure 1 represents the brief architecture of our proposed system. Here S' is the set of symptoms of patient P_t. LPA adds some signs in the set S', and new set S is the set of sign and symptoms of that patient, so $n \geq s$. Local Physician Agent(LPA) sends this set S to the discussion module of the system. We represent it by 'Bayesian network of Bayesian agent with coordination ontology'. From the module final diagnosis result $dmx_i(t)$ is chosen which is sent to Treatment plan repository. From which corresponding treatment plan Tp_i is chosen and report $R(t)$ is generated. Report $R(t)$ is send back to LPA.

Formally system architecture can be defined by a set of tuple,

VBMS = $\{HA, BNBA, CO, Tprep\}$

where

- HA : Human agent. There are two types of human agent. 1. Patient 2. Local physician Agent(LPA)
- $BNBA$: Bayesian network of Bayesian Agent(BNBA) is part of discussion module of the system. It is a graph which can be represented by a set of tuple, $BNBA = \{V, E, P\}$ where,
  - V is set of variables in the BNBA which act as nodes in a graph. Two types of variables are there, 1.NV: Normal variable having number of mutual exclusive states.
    2.BAV: In the system each software expert agent also can be denoted as Bayesian Agent variable having number of mutual exclusive states. These variables or expert agents are called Bayesian Agent variable because each variable itself consists a knowledge base (KB) which is in the form of Bayesian Network. So Bayesian network of each Bayesian Agent can be represented by a set of tuple, $BN = \{V', E', P'\}$ where,
    - V' : Set of variables in the Bayesian Network of each Bayesian Agent which act as nodes of the graph. In our application there are two type of variables, 1. Ds: Set of disease variables. 2. Sy: Set of symptom variables.
      so we can say, $V' = (Ds \bigcup Sy)$
    - E' : Set of directed edges which represents the causal relationships between variable V'.
    - P' : Joint probability distribution over variable V'. It is defined as,
      $P(V'_1, V'_2....V'_w) = \Pi_{i=1}^{w}\ p(V'_i|\text{parents}(V'_i))$
  - E: Set of directed edges which represent causal relationships between variables V.
  - P: Joint probability distribution over variable V. It is defined as,
    $P(V_1, V_2....V_x) = \Pi_{i=1}^{x}\ p(V_i|\text{parents}(V_i))$
- CO: Coordination Ontology is a part of discussion module. It contains set of rules which are used to get final decision among a number of alternative decisions. The detail of coordination ontology can be found in our earlier paper [15].
- $Tprep$: Treatment plan repository is defined over medical domain M. Here for each disease treatment plan is defined. i.e. disease to treatment plan is a One-to-one mapping. It can be defined by these function,
  $TPkb$: $D \longrightarrow TP$
  where D = { d_i |i=1....n } is a set containing all treatable diseases.
  and TP = { tp_i |i=1....n } is set of corresponding treatment plan. Function X is used to generate a report R,
  $X(d_i, TP_i) = R$. This report is sent back to LPA.

B. Architecture of BNBA

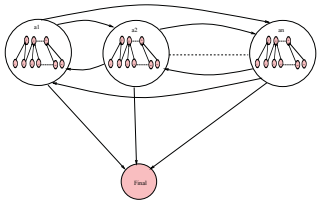


Fig. 2: Architecture of Bayesian Network of Bayesian Agent (BNBA)

Figure 2 represents the Bayesian network formed by Bayesian agents. where each Bayesian agents holds one Bayesian network. so number of mutually exclusive states of each Bayesian variables depends on number of different diseases in the Bayesian network of that agent. Number of mutual exclusive states of variable Final is union of total number of different diseases of all expert agents

C. Algorithm to construct Conditional Probability table if 'Final' variable in BNBA

As in $BNBA$ 'Final'node is conditionally dependent on all Bayesian Agent variables so to take decision among a number of alternative decision Conditional Probability Table (CPT) should be formed. This CPT should be formed by abiding rules of Coordination Ontology.

CPT_CONSTRUCT(m,count,Dc{})
If (m/2) $\leq$ count $\leq$ m then
If $\forall$ $d_i \in$ Dc Conflict = False then
{ **Make entry $\forall$ $d_i \in$ Dc in CPT of Final variable in BNBA as follows,**
p(d_i/Dc{}) = 1 and rest entries of the CPT = 0 }
ElseIf $\forall$ $d_i \in$ Dc Conflict = True then
Grouping is done
If Groups are asymmetric then
{ **Make entry $\forall$ $d_i \in$ Dc in CPT of Final variable in BNBA as follows,**
p(d_i/Dc{}) = { The Cardinality of the group in which di belongs $\times$ (1 / Count) }
}
Else
{ **Make entry $\forall$ $d_i \in$ Dc in CPT of Final variable in BNBA**
as follows,
p(d_i/Dc{}) = { Value of the maximum trust value of the group in which
d_i **belongs $\times$ (1 / summation of maximum trust value from each group)** }
}
EndIf
EndIf
ElseIf Count $<$ (m/2) then
new (m - Count) number of expert agents are chosen and with the total m number of expert agent again call function CPT_CONSTRUCT(m,count,Dc{})
EndIf

Where m = Number of expert agent participating in discussion. count = Number of expert agent able to take decision among m expert agents. Dc{} = array of decisions of 'count'number of expert agents.

IV. Problem formulation

Let in the system the set of expert agents A = { a_i |i = 1....m } is a team with a common objective. Let Sm is a global set of all possible sign symptoms of a patient. Now suppose by LPA a set of symptoms is taken from a patient P_t and the set of those sign and symptoms formed is S = { s_i |i=1...n } LPA sends this set S to all expert agents $a_i \in A$.
Their exists a logical network defined by a graph $G = (Ag, Ed)$ where A is agent corresponds to vertices and $e \in Ed$ corresponds to communication link. Depending on the received symptoms each expert agent arised some questionnaire. Let Q_j = { q_i |i=1...k } is a set of questionnaire arised by expert agent a_j. According to questionnaire LPA finds out other sign and symptoms of the patient P_t. Let 'α ' is a function responsible for the selection of sign and symptoms depending upon q_i. α (q_i)= s where $s \subset Sm$. similarly, $\alpha(Q_j)$ = S_j = { s_i |i=1...l } where $S_j \subset Sm$ and $S_j = \emptyset$ can be true. LPA sends this S_j to agent a_j. All expert agent independently maps all these processed data on their KB(represented in the form of Bayesian network) which contains (s, v) ordered pair of symptoms and their possible value/ranges. A function β is defined over (s, v) to find out probable disease.
β ($\sum_{i=1}^{u}(s_i,v_i)$) = $d_j(t)$ where $n \leq u$ (As after getting answer of questionaries some expert agents can get more than number of symptoms from patient, so these symptoms are added with n and let now set of symptom is u)
Where $d_j(t)$ is the decision of expert agent a_j at time t. β returns information regarding disease on the basis of the following relations.
F: $2^{s \times v} \longrightarrow$ D (many to one relation), Where D is the universal set of all possible disease of patient.
so D = {d_i |i=1...w } Now all expert agent exchanges their decisions with other expert agents and according to other's decision they update their knowledge about the diagnosis using equation 1, If expert agent $a_i \in A$ take diagnosis decision $d_i(t)$ at time t, where $d_i(t) \in$ D Then agent a_i 's probability distribution over D at time 't'is,
$p_i(d_i(t), t)$ = Belief of the agent a_i that disease $d_i(t)$ is the probable disease at time t. After message passing and sharing their decisions with other experts if agent a_i gets $d_i(t)$ is the decided disease of agent a_j at time t, Agent a_i's new belief at time twill be,
$p_i(d_i(t'), t')$ = $p_i(d_i(t), t)$ + ...(1)
Where ≥ 0 and ($1 \geq p_i(d_i(t), t) \quad \geq 0$),

According to equation (1) belief of each expert agent is updated for each disease $d_i \in D$. After belief updating again decision making process is restarted and each expert agent takes decision independently. Let those decision can to be represented in the form of set Dc = { $d_i(t)$ |i= 1....o and o $\leq$ m }. Now coordination ontology is used to find out final decision among all alternative decisions of set Ds. A function CON is defined is CON(Dc) $\longrightarrow$ $d(t)$ where $d(t)$ is the final decision at t^{th} time. The belief value of final decision is $p(d(t), t)$.

The performance of the team or the utility of the final decision will be affected if summation of divergence between each agent′s belief and final decision belief increases.
As utility of the final decision $\propto$ total divergence between decisions of expert agents.
The divergence between the belief of final decision and a_i agents decision $d_i(t)$ at time t is , $|p_i(d_i(t), t)$ - $p(d(t), t)$ |= Δ_i(D, $p_i(d_i(t), t)$)
The bigger the above value mean higher divergence. The cost of $\Delta_i(\bullet)$ divergence to an agent a_i at particular time is
C(a_i, $\Delta_i(\bullet)$) $\longrightarrow$ R
Where R is the real number. Thus the overall optimization function to minimize is,

$$\sum_{a_i \in A} \int_{t=0}^{T} \mathbf{C((}a_i\mathbf{,}\ \Delta_i\mathbf{(\bullet))\ dt}$$

To minimize this Optimization function again expert agents communicates with each other by message passing to share their decision and update their belief. The discussion and updating of their belief process goes iteratively until optimization function get minimized. After getting the final decision with maximum utility let that is $dmax_i(t)$ $\in$D,

where i may vary from 1 to w. This final diagnosis result is sent to Treatment plan repository to generate report.

V. Case Study

We have done a case study on a medical domain of different types of fever. In the domain 7 types of diseases are there. Those are, D = { Urinary tract infection, Typhoid, Brucellosis, Lobar Pneumonia, Malaria, Kala-azar, Diseased liver }
and 14 types of symptoms of these diseases. Those are,
Sm= { Headache(A), Body pain(B), Joint pain(C), Vomiting(D), Chills(E), Poor appetite(F), Loose bowels(G), Nausea(H), Urine problem(I), Abdominal pain(J), Diarrhea(K), Nose bleeds(L), Cough(M), Skin problem(N) }
In bracket we represent the abbreviations of corresponding symptom. e.g. Headache is denoted by A.
There are 8 expert agents we choose to participate in group discussion.
Each expert agent′s KB is represented in the form of Bayesian network formed by upper specified diseases and symptoms. KB may be different for different experts depending on their knowledge.

Now suppose from a patient LPA takes symptoms and sends them as a set S = {I(9), J(6), E(9), H(6), D(9), L(6), O(6), M(2), C(6)}. In bracket severity value of every symptom (high(9), medium(6), low(2)) is given.
After a number of rounds of discussion final decision about the diagnosed disease is found as Lobar pneumonia. Simulations of discussion is shown in Figure 3 and 4. In figure 3 x axis represents number of rounds of discussion after which experts get negotiated in same decision. We can see at 5^{th} round of discussion the optimization function value is 0, i.e. it gets minimized.

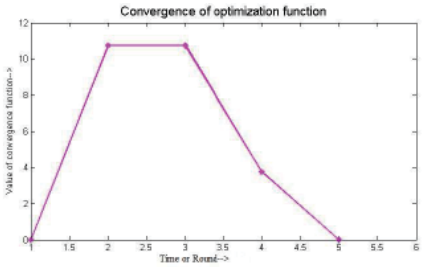


Fig. 3: Convergence of optimization function

In figure 4 it is shown that after time or round value 4, all experts belief value converges. that is their decision about the diagnosis become same.

VI. Conclusion

In this paper, we propose a multi agent based group discussion mechanism to form a virtual medical board like system. Here each agent acts as an expert physician, consisting of a knowledge base represented in the form of bayesian network. All expert physicians independently diagnose a patient, communicate with each other with a common vocabulary i.e.

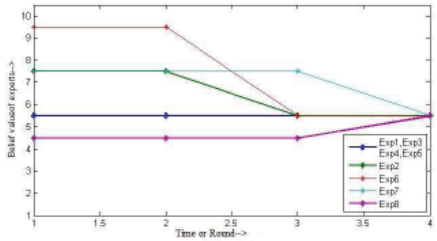


Fig. 4: Convergence of beliefs of all expert agents

Coordination Ontology to agree upon in a common decision. We also propose a optimization function, minimization of which increase the final decision utility.

References

[1] G. Weiss, Ed., "Multiagent systems: a modern approach to distributed artificial intelligence", MIT Press (1999)
[2] M. J. Wooldridge, "Introduction to Multiagent Systems", John Wiley Sons, Inc(2001)
[3] H Lee, M A Buckland and J W Shepherdson, "A multi-agent system to support location based group decision making in mobile teams", In BT Technology Journal, Vol 21 No 1, January 2003
[4] K. K. Breitman and J. C. S. do Prado Leite, "Ontology as a Requirements Engineering Product", In 11th IEEE International Requirements Engineering Conference (RE03),pages 309−319, Sep. 2003.
[5] A. Aguilera, E. Herrera and A. Subero, "MEDICAL COORDINATION WORK BASED IN AGENTS", In The 3rd International Symposium on Biomedical Engineering (ISBME 2008)
[6] I. G. Czibula, G. S. Cojocar and A. M. Guran, "IMASC - An Intelligent MultiAgent System for Clinical Decision Support", In 2008 First International Conference on Complexity and Intelligence of the Artificial and Natural Complex Systems. Medical Applications of the Complex Systems. Biomedical Computing.
[7] Y. Jiang, J. Hu and D. Lin, "Decision Making of Networked Multiagent Systems for Interaction Structures", In IEEE transactions on systems, man, and cybernetics − Part A: Systems and Human, VOL. 41, NO. 6, November. 2011.
[8] M. Indiramma and Dr K. R. Anandakumar,"Collaborative Decision-making in Multi-agent Systems for GIS Application", In Proceedings of the International MultiConference of Engineers and Computer Scientists 2008 Vol I IMECS 2008, 19-21 March, 2008, Hong Kong.
[9] M. Morge and P. Beaunea, "Negotiation Support System based on a Multi-agent System specificity and preference relations on arguments", In Proceedings of the 2004 ACM symposium on Applied computing Pages 474 - 478 ,ACM New York, NY, USA 2004.
[10] M. Morge, "Distributed Decision Making",Textbook, Part of the Course on MultiAgent Oriented Programming,2nd December 2008
[11] A. Filippoupolitis and E. GelenbeA, "Distributed Decision Support System for Building Evacuation", In Proceeding HSI'09 the 2nd conference on Human System Interactions Pages 320−327.
[12] V. K. Mago1, M. S. Devi, and R. Mehta, "Decision Making System Based on Bayesian Network for an Agent Diagnosing Child Care Diseases", In D. Riao (Ed.): K4CARE 2007, LNAI 4924, pp. 127136, 2008. Springer-Verlag Berlin Heidelberg 2008
[13] C. Smaili, C. Rose and F. Charpillet, "Using Dynamic Bayesian Networks for a Decision Support System Application to the Monitoring of Patients Treated by Hemodialysis", In Proceeding ICTAI '05 Proceedings of the 17th IEEE International Conference on Tools with Artificial Intelligence Pages 594 - 598
[14] W. Premchaiswadi, N. Jongsawat and W. Romsaiyud, "Bayesian network inference with qualitative expert knowledge for group decision making", In Intelligent Systems (IS), 2010 5th IEEE International Conference 2010
[15] S. Acharya, A. Dutta, "Coordination ontology for multi agent based distributed decision making", In proceddings of Parallel Distributed and Grid Computing (PDGC), 2012 2nd IEEE International Conference on 6-8Dec, .2012 vol., no., pp.508- 514.doi: 10.1109/PDGC.2012.6449873

Software Quality Assurance Ontology

from Development to Evaluation

Nada Bajnaid
Faculty of Computing, King Abdulaziz University
Jeddah, Saudi Arabia
nbajnaid@kau.edu.sa

Rachid Benlamri
Dept. of Software Engineering, Lakehead University
Thunder Pay, Canada
rbenlamr@lakeheadu.ca

Nada Bajnaid, AlgirdasPakstas, Shahram Salekzamankhani
School of Computing, London Metropolitan University
London, United Kingdom
(nab0378, a.pakstas, s.salekzamankhani) @londonmet.ac.uk

Abstract- Even though, Software Quality Assurance (SQA) becomes one of the most important objectives of software development and maintenance activities, yet there is no consensus among most of the domain terminolog, concepts and techniques. Inconsistency and terminology conflicts appear between standards even within the same organization. There was an effort by different bodies to improve consistency and coherency among standards. To contribute to this effort, a Software Quality Assurance ontology that represents both domain and operational knowledge of SQA is proposed in this paper. SWEBOK and international standards are used as the main sources to extract the ontology concepts and realtionships. Different evaluation approaches were conducted to validate the quality and usefulness of the proposed ontology .

Keywords-component; Software Quality Assurance, Ontology Engineering, e-learning, context-awareness.

I. INTRODUCTION

Software is a key element of modern computing systems. People are increasingly relying on software and demanding higher quality products than ever before. Studies show that software companies can make more money through increased customer satisfaction and improved product quality [1]. Some authors have defined the entire discipline of Software Engineering (SE) as the production of quality software [2]. Although, Software Quality Assurance (SQA) becomes one of the most important objectives of software development and maintenance activities, yet there is no consensus among most of the domain terminology, concepts and techniques. Hence, meaning of terms may inter-relate or overlap.

In software engineering, people with different necessities and viewpoints need to communicate and share knowledge during all the stages of the software life cycle. Information sharing helps to prevent inconsistency among teams that are geographically dispersed and are participating in the software development process.

Using ontology to model the SE knowledge shorts the development time, improves productivity, decreases cost, and increases product quality. Ontologies provide better understanding of the required changes and the system to be maintained [3; 4; 5]. Software engineering domain ontologies are very useful in developing high quality, reusable software by providing an unambiguous terminology that can be shared through the development processes. Ontologies also help in eliminating ambiguity, increasing consistency and integrating distinct user viewpoints [6; 7; 8; 9].

Standardization plays an important role in software engineering by providing organizations with agreed and well organized practices that assists the users of software development methods in their work. Despite the efforts in research and international standardization, inconsistency and terminology conflicts appear between standards even within the same organization. Besides, there is still no single standard which embraces the whole Software Quality Assurance (SQA) knowledge. There was an effort by different bodies to develop Software Engineering standards followed by the forming of the ISO/IEC Joint Technical Committee 1 (JTC1) workgroup in order to guarantee consistency and coherency among standards. The IEEE Computer Society and the ISOJTC1-SC7 agreed to harmonize terminology among their standards.

Aiming to contribute to the harmonization of SQA standards and research proposals, this article presents an ontology model that identifies and represents the SQA knowledge, both conceptual and operational. The rest of the paper is organized as follows: section 2 introduces the SQA ontology development and conceptualization activities. Section 3 evaluates the developed ontology showing case study that demonstrates the deployment of the developed ontology. Finally, section 5 summarizes the main findings of this study and suggests further research work.

II. FROM STANDARDS TO ONTOLOGY MODEL

There is no single standard which embraces the whole SQA knowledge. Various standards and proposals have used different terminologies and vocabularies. With a goal to develop a consistent terminology for software quality assurance, different ISO and IEEE standards were used in the ontology conceptualization activity while the Software Engineering Body of Knowledge (SWEBOK) [10] remains the important and primary source for developing the SQA ontology that includes both domain and operational knowledge. Table 1 shows part of the glossary of terms extracted from the previous sources aided by domain experts.

TABLE I. GLOSSARY OF TERMS OF THE SQA DOMAIN ONTOLOGY

Term	Super-concept	Definition	Source
Process	SQAConcept	A set of activities that can be recognized as implementation of practices for specific purpose A set of interrelated actions and activities performed to achieve a specified set of products, results, or services.	Adapted from CMMI v1.2 PMBOK 2008
Project	owl:Thing	A temporary endeavour undertaken to create a unique *product*, *service*, or *result*.	PMBOK 2008
Deliverable	SQAConcept	Any unique and verifiable product, result, or capability to perform a service that must be produced to complete a process, phase, or project.	PMBOK 2008
Attribute	SQAConcept	A measurable physical or abstract property of an entity.	IEEE 00100
Requirement	owl:Thing	Need or expectation that is stated, generally implied or obligatory	ANSI/ISO/ASQ Q9000-2000
Functional Requirement	Requirement	A requirement that specifies a function that a system or system component must be able to perform.	IEEE 610-12
Resource	SQAConcept	Any capability that must be scheduled, assigned, or controlled by the underlying implementation to assure no conflicting usage by processes.	IEEE 00100
Technique	Resource	A defined systematic procedure employed by a human resource to perform an activity to produce a product or result or deliver a service, and that may employ one or more tool.	PMBOK 2008
Measurement	SQAConcept	The determination of the magnitude or amount of a quantity by comparison (direct or indirect) with the prototype standards of the system of units employed.	IEEE 00100
Measurement Metric	SQAConcept	A quantitative measure of the degree to which a system, component, or process possesses a given attribute.	IEEE 610-12

For any quality product, measures associated with its attributes should collectively reflect likely user satisfaction with the use of the product and therefore the product entire quality [11].

Measurement plays an important part in software development. It can be used to indicate the quality of the product being developed [12]. According to Pressman's categorization of software metrics, quality metrics, which measure how the customer requirements are fulfilled, indicate how closely software conforms to explicit and implicit customer requirements. In this study, software measurements and metrics are at the heart of the SQA ontology design. All aspects of SQA measurement and metric as described in the ISO/IEC 9126 standard are reflected in the proposed SQA ontology design.

A complete conceptual model of the SQA ontology is illustrated in Figure 1 [13]. The figure shows the main SQA concepts as OWL classes where the arrows represent relationships (OWL object properties) between domain classes (the head of the arrow) and range classes (the tail of the arrow) where the name on the line depicts the name of the relationship. The individuals are modeled as 'objects' or literals in the rectangular boxes (see Figure 2). The *is-a* property relates an SQA concepts with its instances (OWL individuals). In the model, *Process* and *Measurement* are concepts (classes) while *Use Cases* and *Test Coverage* are instances of the classes *Technique* and *Measurement-Metric* respectively. Here we have followed some of the RDF graph notation for describing tuples.

Protégé 3.4.6 was used as the ontology editing and knowledge acquisition tool. Jambalaya tab, a protégé plug-in is used for ontology visualization. A class hierarchy of the software quality domain ontology as displayed by the Jambalaya tab of Protégé is shown in Figure 2 where blue arrows represents SQA main classes and red arrows represents individuals of a class. Due to space limitation not all individuals of all classes are shown.

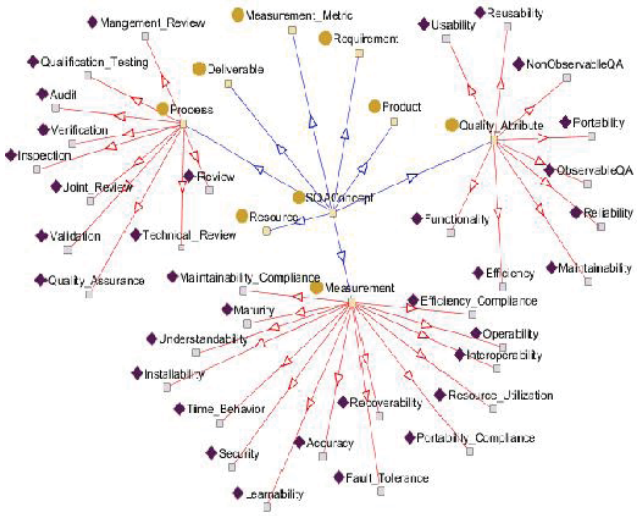


Figure 1. Class Hierarchy of the SQA Ontology

III. EVALUATING THE SQA ONTOLOGY

Before publishing an ontology or building a software application that relies on ontologies, the ontology contents (its concepts definitions, taxonomy and axioms) need to be evaluated. Ontology evaluation consists of assessing its quality, thus improving its users' confidence and promoting its reusability. A survey on ontology evaluation methods and tools can be found in [14].

According to Gómez-Pérez [14] ontology evaluation requires:

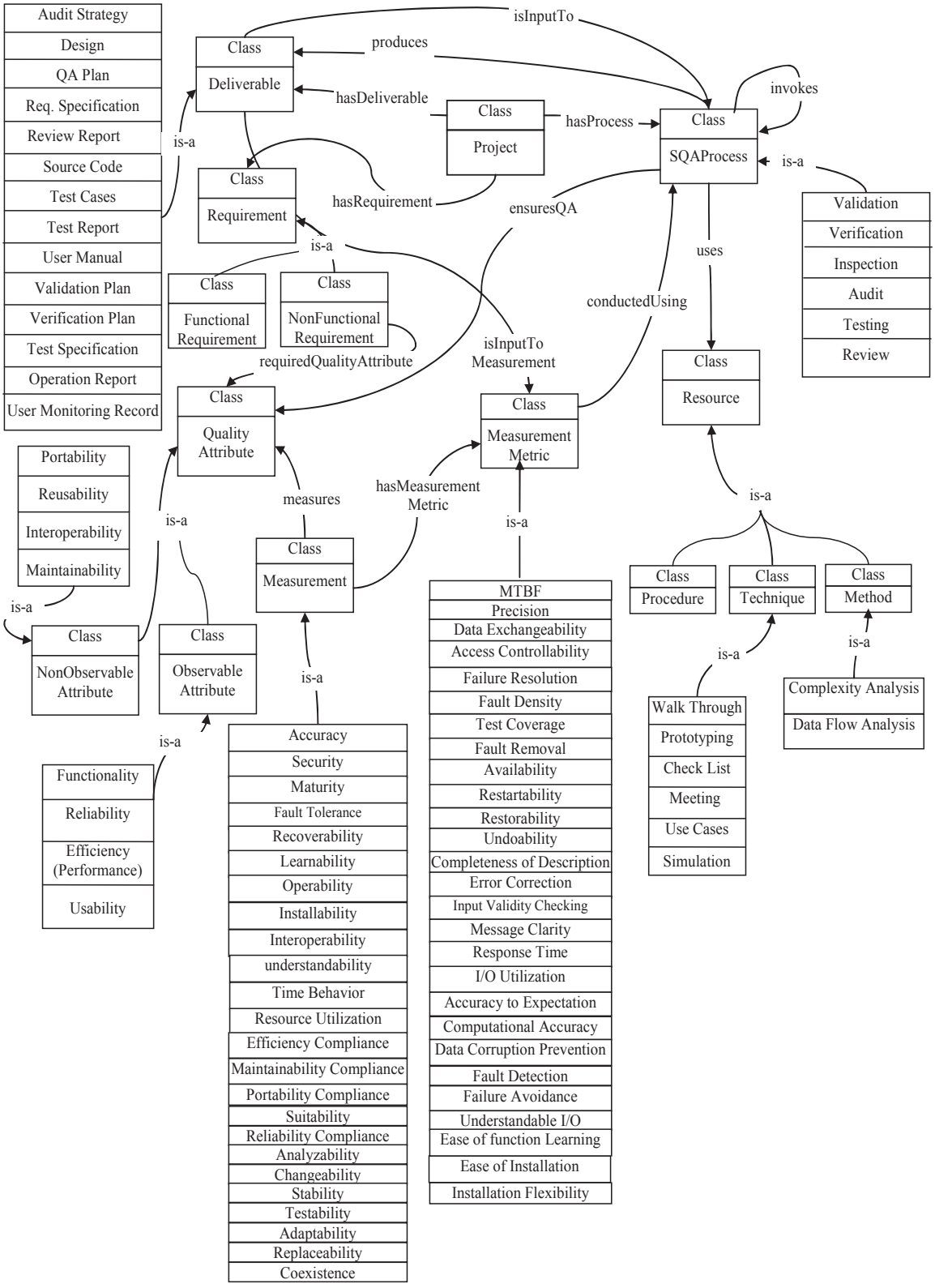


Figure 2. The SQA Conceptual Model

- ***Verification*** which refers to building the ontology correctly;
- ***Validation*** which refers to whether the ontology definitions really model the domain for which the ontology was created. Ontology validation ensures that the correct ontology was built. The goal is to show that the world model is compliant with the formal model;
- ***Assessment*** which focuses on judging the ontology from users' points of view (human judgment).

Many attributes were used to develop the above-mentioned ontology assessments. The most used attributes are:

- ***Completeness:*** all knowledge that is expected to be in the ontology is either explicitly stated in it or can be inferred.
- ***Consistency:*** refers to whether or not a contradictory knowledge can be inferred from a valid input definition.
- ***Conciseness:*** ensures that the ontology is free from any unnecessary, useless, or redundant definitions.
- ***Expandability:*** refers to the ability to add new definitions without altering the already stated semantic.

A. *Ontology Verification*

During implementation, the developed ontology was verified for consistency. We have used the Protégé consistency checker tool to automatically validate the consistency and conciseness of the developed ontology. The tool uses red color highlighting to signal inconsistency. Only inconsistent classes will be displayed by the tool. Fig. 3 shows the result generated by Protégé and the Racer Pro reasoner for the consistency checking where no inconsistence classes are listed.

Syntax checking is performed by Protégé OWL plugin which generates OWL statements during creation of the ontology using the Graphical User Interface. The plugin ensures that the generated OWL statements adhere to the rules of the OWL language. In addition, the Jambalaya tab (Protégé visualization plugin), enables a view of the graph representation of the ontology to ensure the ontology is consistent with the conceptual model (see Figure 1). Fig. 4 illustrates the SQA ontology top level concepts that adhere with the conceptual model.

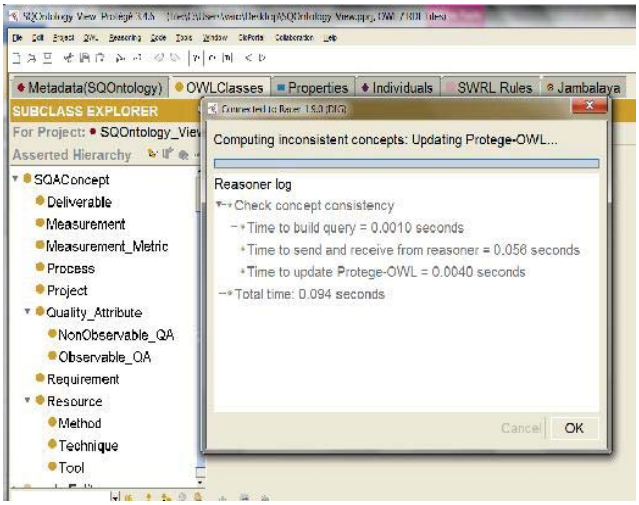


Figure 3. Racer Pro Consistency Checking Result

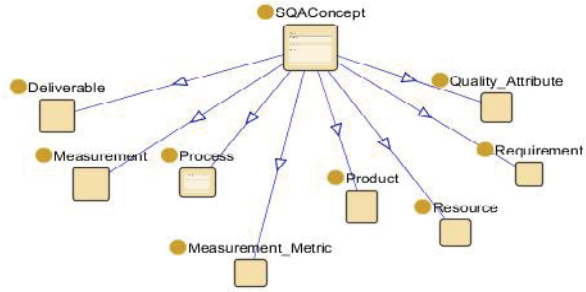


Figure 4. The Top Level of the SQA Ontology

B. *Assessing the Quality of the SQA Ontology*

Ontology assessment was conducted by judging the ontology content from the SE specialists' point of view. A survey based on an assessment questionnaire was developed for experts to check whether the model matches the purpose it was built for.

1) Assessment Questionaire Design

The ontology assessment questionnaire consists of four parts:

Part I contains closed questions about the respondent, such as their experience in the SQA and ontology domains, their involvement in teaching the SQA domain and their opinion about the usefulness of using ontologies in teaching SQA.

Part II consists of 7 closed questions with a scale of 1-5, where 5 = strongly agree and 1 = strongly disagree, to validate the following criteria [16; 17; 18; 19]:

Completeness: the model covers major concepts of the domain;

Structure: the taxonomy and relationships are represented correctly in the model;

Clarity: the model is free from unnecessary and redundant concepts;

Consistency: the model is free from explicitly or implicitly contradictory knowledge;

Expandability: new knowledge can be added to the model without altering the existing semantic.

Parts III and **IV** consist of open questions about the respondent suggestions of non-relevant concepts to be removed from the model and missing concepts to be added to the model respectively.

2) Statistical Analysis and Results

The conceptual model, shown in Figure 1, with a hyperlink to the questionnaire has also been sent to SE domain specialists to participate in the survey. Collecting responses was a challenge step due to the limited number of experts in the SE domain and in SQA in particular. It took more than 7 months to get 16 responses only where 11 of them rating themselves as experts. The results of the survey are presented below where an enhanced version of the ontology is being developed to reflect the main suggestions from the questionnaire:

Completeness: Majority of the participants (86.7%) agreed that the ontology developed in this research covers major concepts of the SQA domain. 15.4% of them strongly agree and none of the respondents disagree with the completeness of the ontology. However, an important suggestion to add *Testing* related concepts (black and white box, system and unit test…etc.) was made. Though, the current ontology is not heavily focused on testing techniques, it is worth investigating this ontology aspect in future developments. Another suggestion was made to add concepts such as *Software type*, *Software life cycle model*, *Architecture*, *Configuration management,* however, we strongly believe that these are not SQA concepts. Nevertheless, these concepts can be added to the ontology if the latter is to be mapped to other SE areas or to an upper-level SE ontology.

Structure: A reasonable majority of the respondents (66.7%) agreed with the ontology taxonomy as is, with no real disagreements. There were few remarks such as having *Design* comes after *Review Report* in the list of instances of the class *Deliverable,* which we consider semantically insignificant.

Clarity: This criteria obtained a borderline score, just around the mean (3.13). However, we believe that this a reasonably good result due to the large number of overlapped and redundant SQA terms in available proposals and sources of SQA knowledge. It was noted that most reported disagreements were related to the confusion between Measurements and Metrics. A significant suggestion that will be adopted in the enhanced version is to use the terms *Quality_Characteristic* and *Sub-characteristic* instead of *Quality_Attribute* and *Measurements* respectively. We can also replace the term *Measurement_Metric* with the term *Measure* as per the latest quality standard ISO/IEC 25010 [20].

Consistency: A reasonable majority of the responses (66.7) agreed that the developed ontology is consistent where 30% of them strongly agreed on its consistency. Ontology consistency was verified using the Protégé consistency checker plugin.

Expandability: A good ontology is assumed to cover necessary concepts of the domain and structure them in a way that adding evolving concepts would not affect the existing structure. A satisfactory result was obtained for this criterion as the majority (80%) agreed on the expandability of the developed ontology. Suggestions to include agile terminology with new quality measurements and metrics (as in ISO/IEC 25010) will be considered as extensions in the enhanced version of the ontology.

Although, there is no such a single ontology that can unanimously represent any knowledge area, especially for an evolving domain like SQA, the survey shows a high level of agreement around the major assessment criteria. This is despite the fact that each participant responds based on their own view, background and context.

Participants' responses to Part II of the assessment questionnaire are illustrated in Figure 5. Responses of participants who are considered to be expert in the field and those with average expertise are represented in figures 6 and 7 respectively. The experts' evaluation is significant and very positive.

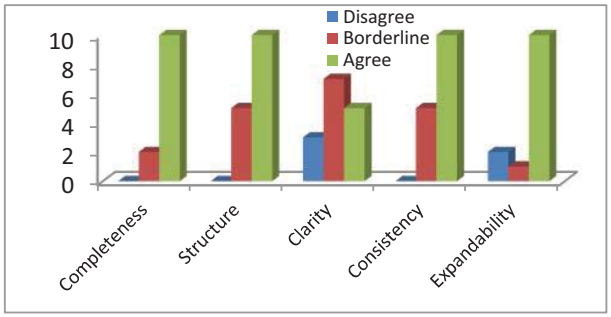


Figure 5. Participants' Assessments of the SQA Ontology

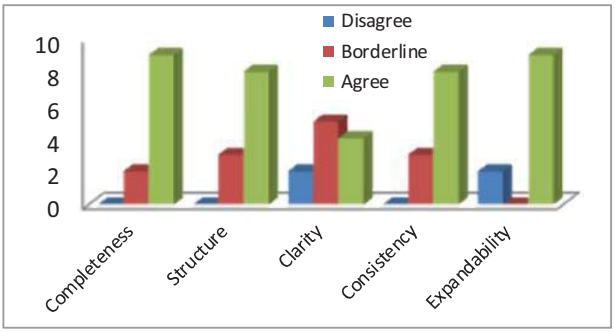


Figure 6. Experts' Assessments to the SQA Ontology

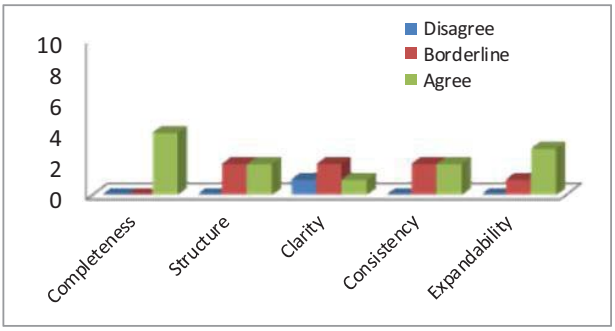


Figure 7. Assessments of Participants with Average Experience in the Domain

C. *Validation Using Application-Based Evaluation: SQAES Case Study*

Application-based (or task-based) evaluations offer a useful framework for measuring practical results of ontology deployment such as responses provided by the system and the ease of use of the query component [18]. A proof of concept prototype consisting of an SQA E-Learning System (SQAES) has been designed and implemented [21]. The prototype system aims at guiding software developers through the necessary QA practices by providing resources that deal with SQA related aspects of the software process in hand and hence improves product quality. This is achieved by sensing the developer's current activity and suggesting relevant learning objects LOs (e.g. recommendations for good practices, example code, and graphical description of a related methodology/process) that deal with all SQA aspects related to the current software

process. In this study, our context model is divided into a higher level operational global ontology and domain specific ontology (SQA ontology). The global ontology is a unified ontology that models the software developer's profile and context. The system is able to determine the leaner's current software development context and infer related SQA knowledge by invoking the appropriate reasoning mechanisms. To achieve this goal, the system's global ontology is augmented with reasoning rules encoded using the Semantic Web Rule Language (SWRL) [13].

Figure 8 shows a combined view of screen shots presented by the system when the learner query about "*Precision*" metric. Besides the learning resources associated with the queried concept, the system provides the learner with suggested SQA related topics for further investigation. Unnecessary and overwhelmed information is prevented using ontology axioms [21].

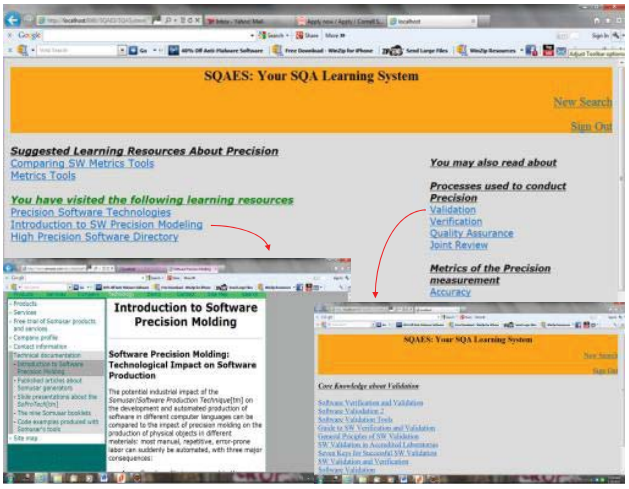


Figure 8. SQAES Responses to Learner's Query

IV. CONCLUSION

Software Quality Assurance ontology provides consistent terminology that may support communications between people and software agents. In this paper an SQA ontology that represents both domain and operational knowledge was developed. The common vocabulary and relationships modeled in the developed ontology is an attempt to resolve the problem of inconsistency among current standards and proposals. The ontology has been assessed and different evaluation approaches were conducted to validate and assess the SQA ontology. Future plans include the enhancement of the proposed ontology to account applicable comments provided by participants of the ontology assessment questionnaire.

ACKNOWLEDGMENT

The authors would like to thanks reviewers for their assessments, comments, and suggestions that helped improve the current work.

REFERENCES

[1] Boehm, B., Chulani, S., Verner, J., Wong, B., (2009). "Seventh workshop on Software Quality," Software Engineering - Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on , vol., no., pp.449-450, 16-24 May 2009.

[2] Mnkandla, E., Dwolatzky, B. (2006). Defining Agile Quality Assurance. Proc. ICSEA 2006: International Conference on Software Engineering Advances.

[3] Mendes, O., and Abran, A., (2004). Software Engineering Ontology: A Development Methodology, Position Paper, Metrics News 9:1, August, pp. 68-76.

[4] Wille, C., Dumke R., Abran A. and Desharnais J.M., (2004). E-learning Infrastructure for Software Engineering Educations: Steps on Ontology Modeling for SWEBOK, Proceedings of the IASTED International Conference on Software Engineering, pp. 520-525.

[5] Calero, C., Ruiz, F. and Piattini, M., (2006). Ontolgies in Software Engineering and Software Technology, Springer

[6] Uschold, M., and Gruninger, M., (1996). Ontologies: Principles, Methods, and Applications, Knowledge Engineering Review, Volume 11 number 2.

[7] Perez, A. and Benjamins, V., (1999). Overview of Knowledge Sharing and Reuse Components: Ontologies and Problem Solving Methods, IJCAI-99 workshop on Ontologies and Problem-Solving Methods (KRR5), Stockholm, Sweden, August 2, 1999.

[8] Spyns, P., Meersman, R., and Jarrar, M., (2002). Data Modelling versus Ontology Engineering, ACM SIGMOD Record, v.31 n.4, December 2002

[9] Zhao Y., Dong J., and Peng T., (2009). Ontology Classification for Semantic-Web-Based Software Engineering, IEEE Transactions on Services Computing, v.2 n.4, 303-317.

[10] SWEBOK, (2004). Guide to the Software Engineering Body of Knowledge, ed. Bourque P., and Dupuis R. IEEE Computer Society Press, 2004. Available at: http://www.swebok.org

[11] Bishop R., Lehman M.M. (1991). A View of Software Quality. IEEE Col. on Designing Quality into Software Based Systems. London, 14 Oct. 1991.

[12] Pressman, R.S., (2005). Software Engineering: a Practitioner's Approach, Sixth edition. McGraw-Hill Inc.

[13] Bajnaid N., Benlamri R. and Cogan B. (2012), "An SQA e-Learning System for Agile Software Development", Proc. of the Fourth International Conference on Networked Digital Technologies, Dubai, UAE, April 24-26, 2012. Communications in Computer and Information Science(CCIS 7899) Series of Springer LNCS (294), 2012, pp. 69-83. ISBN: 978-3-642-30566-5. E-ISBN: 978-3-642-30567-2.

[14] Gómez-Pérez, A., Fernandez-López, M. & Corcho, O., (2004). Ontological engineering: with examples from the areas of knowledge management, e-commerce and the semantic Web, Springer-Verlag, New York; London.

[15] Apache Jena, Java framework for building Semantic Web applications (2012). http://incubator.apache.org/jena/index.html

[16] Gruber, T., (1995). Towards principles for the design of ontologies used for knowledge sharing, Int. Journal of Human-Computer Studies, Volume 43, No. 5/6.

[17] Gómez-Pérez A (2001) Evaluation of Ontologies. International Journal of Intelligent Systems 16(3):391–409

[18] Obrst L., et al. (2007). The Evaluation of Ontologies: Toward Improved Semantic Interoperability. Chapter in: Semantic Web: Revolutionizing Knowledge Discovery in the Life Sciences, Christopher J. O. Baker and Kei-Hoi Cheung, Eds., Springer.

[19] Vrandečić, D., (2009). Ontology Evaluation, Handbook on Ontologies, International Handbooks in Information Systems, 2nd edition, Springer, Heidelberg, 2009, pp. 293-313.

[20] ISO/IEC 25010: Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models, 2011.

[21] Bajnaid, N.; Benlamri, R.; Cogan, B.; , "Context-aware SQA e-learning system," Digital Information Management (ICDIM), 2011 Sixth International Conference on , vol., no., pp.327-331, 26-28 Sept. 2011 doi: 10.1109/ICDIM.2011.6093327

DOPROPC: a domain property pattern system helping to specify control system requirements

Fan Wu
School of Software
Tsinghua University
Beijing, China
wufan0924@yahoo.com.cn

Hehua Zhang
School of Software, TNList, KLISS
Tsinghua University
Beijing, China
zhanghehua@tsinghua.edu.com

Ming Gu
School of Software, TNList, KLISS
Tsinghua University
Beijing, China
guming@tsinghua.edu.com

***Abstract*—Model checking provides means to validate the correctness of systems. It is often desired by the developing process of safety critical control systems. However, it hasn't been widely used in industry. There are several causes for this problem. We think a primary cause is that industry experts are not familiar with formal logics, notations and formal semantics. On the other hand, they are very familiar with the domain notations in the applications. This paper describes domain property patterns of control systems called DOPROPC, which builds a bridge between domain knowledge and formal specifications by using property patterns. With DOPROPC, domain experts can easily understand the meaning of each pattern. Through the automatic translation underlined in our method, the formal specification of the properties can be easily obtained.**

Keywords-property pattern; domain knowledge; control system; formal specification

I. INTRODUCTION

Control systems are usually safety-critical. Recently, formal methods are widely used to ensure correctness and safety of control systems. Model checking is a formal method for software behavioral compliance checking [1], which can automatically check the behavior of software. Despite the automation, users of this technique still have to specify the system requirements in formal specification languages, such as Linear Temporal Logic (LTL [9]), which are unfamiliar to industrial engineers. This is also an important reason that model checking hasn't been widely adopted.

To overcome this difficulty, Dwyer et al. [2] firstly developed a pattern system for property specification. The property patterns are high-level abstractions of frequently used temporal logic formulae. They enable people who are not experts in temporal logics to read and write formal specifications with ease and thus make model checking tools more accessible to common software practitioners [3]. Although property patterns have already been in the abstract level, we found there is still a long distance from requirements to them. Existed property patterns are based on the semantic of formal properties, for example in [2] they defined *Response* pattern as 'A state/event P must always be followed by a state/event Q within a scope.' When an engineer tries to map a real property to *Response* pattern, he/she must recognize the order relation hidden in the property and must clear up which state/event is occurred before another one. This work is usually difficult to industrial engineers, since it needs knowledge about formal semantics. Intuitively, we come up with an idea: using what industrial engineers are most familiar with -domain knowledge- to do the work. That is to say adding domain knowledge to property patterns which can be a bridge between domain knowledge and formal semantics.

In this paper, we present a new property pattern system called DOPROPC (DOmain PROperty Pattern for Control systems), and an associated tool helping experts to use this pattern system. DOPROPC is a domain property pattern system to specify behavioral properties of control systems. It is a two-layer pattern system. The top layer is domain based, which uses domain knowledge of control systems. The bottom layer is logic based, which extends from Dwyer et al. property patterns [2]. This pattern system enables experts using their working domain knowledge to specify system properties.

The rest of the paper is organized as follows. Section 2 includes related works. In section 3, we describe DOPROPC, including the bottom layer and the top layer. In section 4, we conclude this paper.

II. RELATED WORKS

Dwyer et al. [2] created Property Patterns in 1999. We call it *qualitative property patterns*. In their work, Dwyer presented 8 property patterns and 5 scopes. Each pattern was represented in Computation Tree Logic (CTL [10]), LTL and Quantified Regular Expressions (QRE [11]). In their study, 92% of 500 properties can be described by their property patterns.

In the later work, V. Gruhn and R. Laue [4] presented *real-time property patterns*, which added time feature based on qualitative property patterns, and represented patterns to Timed Observer Automata. Real-time property patterns can be used in real-time model checking. S. Konrad and B. H. C. Cheng [5] also presented real-time patterns in 2005. In their work, they represented patterns in Metric Temporal Logic (MTL [12]) and Timed Computational Tree Logic (TCTL [13]). In 2008, Lars Grunske [6] developed *probabilistic property patterns* that was represented in Probabilistic Computation Tree Logic (PCTL [14, 15]) and Continuous Stochastic Logic (CSL [16,

This research is sponsored in part by National Nature Science Foundation (No.61202010, 91218302), National Key Tech. R&D Program (No.SQ2012BAJY4052) and 973 Program (No.2010CB328003) of China.

17]). Probabilistic patterns can be used in probabilistic model checking.

Property patterns have been used in different fields. M. Haydar et al. [7] developed specification patterns for formal web verification. They presented detailed patterns by categorizing quality assurance properties of web applications. J. Yu et al. [1] presented property patterns for service composition domain. They used ontology to present property patterns, and solved composition problems which have a great use in web service composition domain. These works use property patterns in certain domains, and provide evidence for our idea. From their work, we can see that applying patterns into certain domain is feasible and necessary. Control system domain is pervasive in our life and is safety-critical, but we haven't found any study presenting property patterns for this domain. Our work targets this problem and has practical significance.

III. DOPROPC PROPERTY PATTERNS

In our work, we developed DOPROPC as a two layer property pattern system. The bottom layer is the base of this system. The top layer is domain related. Considering the two layers are both abstracted, they can't be complete. We design the mapping relation, as shown in Fig. 1. Most of the top layer patterns can be mapped to the bottom layer patterns, but some may be mapped to formal logics directly. The mapping between each layer is totally based on formal semantics. We introduce each layer respectively as follows.

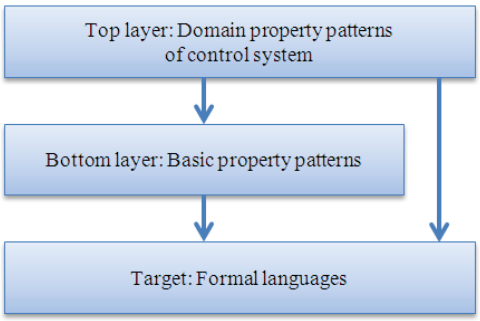


Figure 1. Structure of DOPROPC.

A. The Bottom layer of DOPROPC

The bottom layer depends on *qualitative property patterns* [2], *real-time property patterns* [4, 5] and *probabilistic property patterns* [6]. These three patterns have their own obvious features, so users need to make a decision about which pattern system to use at the first step to describe properties, which is an extra burden to users. Moreover, these three patterns have reduplications. For this reason, we merge these three patterns together to gain an overall view.

We classify our basic property patterns into two categories following Dwyer's classification of patterns: *Occurrence* and *Order*. Occurrence means something happen or not happen. *Order* describes the relation between two or more things. We present 15 property patterns. Table I gives a full view of them. We explain each of them as follows:

- **Recurrence:** A given state/event occurs repeatedly every k times.
- **MinmumDuration:** Once a given state/event is satisfied, it remains at least c units of time.
- **MaximumDuration:** Once a given state/event is satisfied, it remains less than c units of time.
- **ResponseInvariance:** If a given state/event P occurs, then a given state/event Q holds continuously.
- **Until:** A given state/event P holds without interruption until a given state/event Q holds.
- ***Others*:** Other patterns' basic meaning is same as qualitative property patterns. Due to the space limitation, we omit explanations of them, you can see [2] for more details.

TABLE I. BASIC PROPERTY PATTERNS

Pattern::= Occurrence | Order
Occurrence :: = Absence | Universality | Existence | Bounded Existence
| Recurrence
| MinimumDuration | maximumDuration
Order :: = Precedence | Response | PrecedenceChain1-2
| PrecedenceChain2-1| ResponseChain1-2
| ResponseChain2-1 | ResponseInvariance | Until

TABLE II. ABSENCE PATTERN

Pattern name	**Absence**	
Structured natural language description	It is never the case that [P] holds [(tl,tu)] [(pl,pu)]	
Formal language descriptions	***LTL***	G(!P)
	CTL	AG(!P)
	TCTL	$AG_{(tl,tu)}(!P)$
	PCTL	$G_{\geq p}^{\leq tu}(!P)$
	PRISM	$\mathbf{P}_{>=pl}\,[G_{[tl,tu]}\,!P]$ or $\mathbf{P}_{<=pu}\,[G_{[tl,tu]}\,!P]$
Examples	The probability that the system is free of failures in 2 hours is 0.98.	

Our pattern template includes four parts elements: pattern name, structured natural language description, formal language descriptions and examples. Table II shows Absence pattern as an example. Structured natural language description represents pattern's basic behavior, time and probability features. The "Structured natural language description" part is the biggest difference of our basic property patterns compared with those three existed property patterns, for it can describe time and probability features at the same time. In order to increase the applicability of our basic property pattern, we provide 5 frequently used formal languages-LTL, CTL, TCTL, PCTL and PRIMS- in the formal language description part. Considering the ability of expression, LTL and CTL can express qualitative properties. TCTL can express real-time related properties. PCTL and PRISM can indicate probability properties. Different model checking tool has different property

language, e.g. SPIN [18] can verify LTL properties, NuSMV [19] receive LTL and CTL specifications. UPPAAL [20] is a popular real-time model checking tool, and its specification language is a simplified version of TCTL. PRISM [21] can verify probability models, it subsumes several probabilistic logics such as PCTL, and its own language-PRISM-also can specify some probabilistic properties.

In Dwyer's study, they presented 5 scopes: global, before, after, between and after-until. In our basic property patterns, we just use global scope. That is because in Dwyer's survey [2], most examples (80%) used a global scope.

B. The top layer of DOPROPC

We concluded 39 domain property patterns of control systems, which are classified into 12 categories. The patterns are generalized from 104 properties of several real control systems, such as the Dual elevator PLC control system, the Pump control system and the stage control system. In domain property pattern, we use structured natural language description to distinct different patterns, and each of the 39 domain patterns is belong to a certain domain property category. To save space, we briefly introduce each category below, and list one domain pattern in the category as an example.

- **Analog quantity:** Express the range of variables in system within time and probability scopes. It contains three patterns, such as:

  "variable [x] is still [in a certain range]"

- **Time horizon:** Time is important and common features in control system properties. Actually, patterns in other categories can also describe time features, but it is more intuitive to single them out. This class contains three patterns, such as:

  "[P] continuously holds less than [c] units of time"

- **Mutual exclusion:** Two or more state/event can't occur at the same time within time and probability scopes. In this category, we provide two patterns, for example:

  "[P] and [Q] can't occur at the same time"

- **State reachability:** The system can reach a given state/event within time and probability scopes. It includes two refined patterns, for example:

  "After [P] holds, system can reach to [Q]"

- **State hold:** The system hold on a given state, or an event continuously happens within time and probability scopes. The difference between this and State reachability is that State reachability emphasizes something eventually happens, but this class stresses something continuously happening. It includes three patterns, for example:

  "When [P] holds, system will keep working in state [Q]"

- **Start up:** It includes four patterns about the system's start-up action or other entity's initialization in the system. Below is one of the four domain patterns:

  "Only after [P] holds, the [device/system] can start"

- **Stop:** It includes five patterns about the system's stop action or other entity's break down in the system. Below is one of the five domain patterns:

  "the [device/system] will eventually stop"

- **Manual control:** Control system has interactions with humans, and the system often receives control commands from humans. This category contains six patterns, for example:

  "After push button [A], [P] will hold"

- **Command control:** Control system also has interactions with physical devices, so the system can send or receive commands, and do related actions. We include two patterns in this category, for example:

  "After received command [A], [P] will hold/happen"

- **Fault handling:** To ensure the reliability of control system, fault handling is important in these systems. This class has three patterns, for example:

  "When the [system/device] break down, [P] will hold/happen"

- **Motion control:** It expresses properties about movement, motion distance and motion destination. This category has two patterns, for example:

  "After [P] holds, the [device] will move [x] units of distance"

- **Alarm:** It expresses properties about alarms including overtime alarm, fault alarm and alarm reset. This category has three patterns, for example:

  "After [P] holds, if [Q] don't happen in [x] units of time, then system will alarm"

TABLE III. ANALOG QUANTITY 2 DOMAIN PROPERTY PATTERN

Domain pattern name	Analog quantity 2
Structured natural language description	After [P] happens, variable [X] is [in a certain range] [(tl,tu)] [(pl,pu)]
Mapped basic property pattern	ResponseInvariance
Parameters mapping	*P*: P; *S*: ([x][in a certain range])
Formal description example	G(P→G ([x][in a certain range]))

Our domain property pattern template includes 5 parts, just like Table III shows. "Structured natural language description" represents the semantic of the pattern, and variable in brackets should be instantiated by users. "Mapped basic property pattern" and "Parameters mapping" together indicate the transformation from domain property pattern to basic property

pattern. “Mapped basic property pattern” gives the mapped basic pattern to the domain property pattern. “Parameters mapping” explains how to map parameters in the two patterns, and parameter in italic is the variable in the basic property pattern.

IV. Conclusion and Future Work

Property patterns have improved the adoption of formal methods in industry. To move forward the usability of property patterns, we have developed the Domain Property Pattern for Control systems (DOPROPC). Our work has three contributions: (1) merging existent property patterns [2-6] as a full-scale basic property pattern system, which is the bottom layer of DOPROPC, (2) presenting a domain based property patterns of control system as the top layer of DOPROPC, and (3) developed a specification editor to help users to use DOPROPC easily, but for the space limitation, we haven’t introduced the editor in this paper.

Markus Lumpe et al. developed a specification framework called PSPWizard [8]. In their work, they also merged existed property patterns. The difference between their work and our basic property patterns is that we added Bounded Existence pattern, and deleted their TransientState and SteadyState patterns. We can describe TrainsientState using Existence pattern when we set the upper time and lower time equally. Similarly, we can replace SteadyState using Universality pattern by not set time features. While our work has a different focal point against theirs. Our work focus on control system domain, but PSPWizard project is still focus on formal semantics.

In the future, we will optimize domain property patterns of control systems including the specification editor, and develop other domain property patterns. Furthermore, we’ll try to conclude a methodology from summarizing different domain property patterns as a general method to help different domain experts to develop their own domain property patterns.

Acknowledgment

This research is sponsored in part by National Nature Science Foundation (No.61202010, 91218302), National Key Tech. R&D Program (No.SQ2012BAJY4052) and 973 Program (No.2010CB328003) of China.

References

[1] J. Yu, T. P. Manh, J. Han, Y. Jin, Y. Han, and J. Wang. Pattern based property specification and verification for service composition. In K. Aberer et al., editor, Proc. 7th Int. Conference on Web Information Systems Engineering, WISE 06, volume 4255 of LNCS, 2006:156–168.

[2] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in property specifications for finite-state verification. In Proceedings of the 1999 International Conference on Software Engineering (ICSE’99), 1999:411–421.

[3] Gruhn V. Laue R.: Specification Patterns for Time-Related Properties. In 12th International Symposium on Temporal Representation and Reasoning (2005) 189 - 191, Burlington, Vermont, USA.

[4] V. Gruhn and R. Laue. Patterns for timed property specifications. Electr. Not. Theor. Comp. Sci, 2006, 153(2):117–133.

[5] S. Konrad and B. H. C. Cheng. Real-time specification patterns. In G.-C. Roman, W. G. Griswold, and B. Nuseibeh, editors, 27th Int. Conf. on Software Engineering, ICSE 05, 2005:372–381.

[6] L. Grunske. Specification patterns for probabilistic quality properties. In Robby, editor, 30th International Conference on Software Engineering (ICSE 2008), 2008:31–40.

[7] M. Haydar, H. Sahraoui, and A. Petrenko. Specification patterns for formal web verification. In ICWE '08: Proceedings of the 2008 Eighth International Conference on Web Engineering, Washington, DC, USA, 2008: 240-246.

[8] Markus Lumpe, Indika Meedeniya, Lars Grunske. PSPWizard: Machine-assisted Definition of Temporal Logical Properties with Specification Patterns. In Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering. ACM Press, New York, 2011.

[9] Z. Manna and A. Pnueli. The Temporal Logic of Reactive and Concurrent Systems, Specification. Springer-Verlag, 1992.

[10] E. Clarke, E. Emerson, and A. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Transactions on Programming Languages and Systems, 8(2):244-263, Apr. 1986.

[11] K. Olender and L. Osterweil. Cecil: A sequencing constraint language for automatic static analysis generation. IEEE Transactions on Software Engineering, 16(3):268-280, Mar. 1990.

[12] R.Koymans. Specifying Real-Time Properties with Metric Temporal Logic. Real-Time Systems, 2(4):255–299, 1990.

[13] Henzinger, T. A., X. Nicollin, J. Sifakis and S. Yovine, Symbolic Model Checking for Real-Time Systems, in: 7th. Symposium of Logics in Computer Science (1992), pp. 394-406.

[14] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. Formal Aspects of Computing, 6(5):512–535, 1994.

[15] A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In P. S. Thiagarajan, editor, Proc. of the 15th Conference on Foundations of Software Technology and Theoretical Comp. Science, FSTTCS 95, volume 1026 of LNCS, pages 499–513. Springer, 1995.

[16] A. Aziz, K. Sanwal, V. Singhal, and R. K. Brayton. Verifying continuous time markov chains. In R. Alur and T. A. Henzinger, editors, Proc. 8th International Conference on Computer Aided Verification, CAV 96, volume 1102 of LNCS, pages 269–276. Springer, 1996.

[17] C. Baier, J.-P. Katoen, and H. Hermanns. Approximate symbolic model checking of continuous-time markov chains. In J. C. M. Baeten and S. Mauw, editors, Proc. 10th International Conference on Concurrency Theory, CONCUR 99, volume 1664 of LNCS, pages 146–161. Springer, 1999.

[18] Gerard J. Holzmann. The Spin Model Checker: Primer and Reference Manual. Addison-Wesley, Boston MA, 2004.

[19] Alessandro Cimatti, Edmund M. Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. NuSMV 2: An opensource tool for symbolic model checking. In CAV ‘02: Proceedings of the 14th International Conference on Computer Aided Verification, pages 359–364, London, UK, 2002. Springer-Verlag.

[20] Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a nutshell. International Journal on Software Tools for Technology Transfer, 1(1–2):134–152, 1998.

[21] M. Z. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with PRISM: a hybrid approach. Int. Journal on Software Tools for Technology Transfer(STTT), 6(2):128–142, Aug. 2004.

A Mixed-way Combinatorial Testing for Concurrent Programs

Xiaofang Qi Jun He Peng Wang
School of Computer Science and Engineering
Southeast University
Nanjing, China
xfqi@seu.edu.cn, hecelery@163.com, pwang@seu.edu.cn

***Abstract*—Reachability testing is an important approach to testing concurrent programs. It can generate and exercise all partially-ordered synchronization sequences automatically and on-the-fly without constructing any static models. However, it suffers from the problem that the number in synchronization sequences is too large to be exhaustively exercised. In this paper, we present a new combinatorial testing strategy, called mixed-way reachability testing, which adopts the dynamic framework of reachability testing but reduces the number in synchronization sequences that are exercised. The reduction is based on the mixed-way combinatorial strategy, which covers all the valid combinations of receiving events in each subsystem of a concurrent program. We present an algorithm that implements the mixed-way testing and conduct our experiment on an industrial control simulator program. The preliminary experimental results indicate that the mixed-way reachability testing can keep the same effectiveness of fault detection as exhaustive testing while substantially reducing the number in synchronization sequences. Compared to the other existing reachability testing approaches, it achieves a good tradeoff between the effectiveness and efficiency of testing.**

Keywords: combinatorial testing; reachability testing; mixed-way testing; concurrent programming

1. INTRODUCTION

With the wide spread of multi-core architectures, concurrent programs are developed and used pervasively [1]. However, concurrent programs usually exhibit non-deterministic behaviors. Multiple executions of a concurrent program with a given input might select different synchronization sequences and generate different outputs. Non-deterministic testing and deterministic testing are adopted to deal with such non-deterministic behaviors [2-10]. During non-deterministic testing, a concurrent program with a given input is executed many times without any control in order that different synchronization sequences could be chosen [2]. In contrast, a specific synchronization sequence is selected and used to control the execution of the program during deterministic testing [3-10].

Reachability testing is an approach that combines nondeterministic and deterministic testing, in which program information is dynamically collected and thus avoid the inherent imprecision of static analyses [6-10]. It exercises all partially-ordered synchronization sequences of a concurrent program exactly once while no test history is needed to be saved. This exhaustive strategy is valuable in theory, but it is often impractical. Combinatorial strategy is adopted to reduce the number in synchronization sequences [9]. The combinatorial strategy, called t-way reachability testing, generates race variants to cover not all the valid combination of the race outcome changes in a given synchronization sequence, but only all the valid t-way combinations, where t is a small number.

In the t-way reachability testing, the strength of combinations is preset and fixed. This strategy is often blind to identify the strength of combinations, i.e. the value of t because it does not consider the correlativity among synchronization events in a program. If a concurrent system consists of one or more subsystems, each of them usually has different correlativity among synchronization events, i.e. interactions with different values of way. So the t-way combinatorial strategy is not appropriate for every subsystem. In this paper, we present a new mixed-way combinatorial strategy for reachability testing. This strategy first identifies the value of combinatorial way for each subsystem by analyzing synchronization events, and then implements the mixed-way combinatorial reachability testing.

The remainder of this paper is organized as follows. Section 2 uses an example to illustrate the reachability testing process. Section 3 describes the mixed-way reachability testing strategy, and presents an algorithm that implements the strategy. Section 4 reports experimental results that compare exhaustive, t-way, and the mixed reachability testing. Section 5 concludes this paper.

2 BACKGROUND OF REACHABILITY TESTING

Reachability testing is allowed to be applied to several commonly used synchronization constructs [7]. Fig.1 shows a concurrent program CP and a reachability testing process of the program. The program consists of three threads, which interact by accessing monitor *m*. When a thread *T* calls a synchronization method of *m*, i.e. read() or write(), a monitor *call* event occurs on *T* and we refer to the calling as a *sending event*. When *T* finally enters *m*, a monitor *entry* event occurs on *m* and we refer to the entry as a *receiving event*. The *sending event s* is said to be synchronized with the receiving *r* and <*s*, *r*> is a synchronization pair. *s* is the sending part of *r* and *r* is a receiving part of *s*.

Given an execution of a concurrent program, a SYN-sequence, short for synchronization sequence, is defined to be the totally ordered sequence of sending and receiving events that occurred on a thread or a synchronization object, as well as the synchronization pairs exercised in the execution. Fig.1(b) shows six SYN-sequences, namely Q_0, Q_1, Q_2, Q_3, Q_4 and Q_5. As illustrated in Q_0, $s_1 \rightarrow r_1$ represents a synchronization, in which s_1 is a sending event indicating that T_1 call *m*.read(), r_1 is a receiving event indicating that T_1 enters the monitor *m*.

Let s be a sending event, r be a receiving event, and <s, r> be a synchronization pair in a SYN-sequence Q. Let s' be another sending event in Q. We say there exists a race between s and s' with respect to r, if s' could be synchronized with r in a different execution, in which all the events that happen before s' or r in Q, and the synchronizations between these events are replayed. Note that the happen before relation is the usually one as in [7]. If the following four conditions are satisfied, s' is in the race set of r:

1. s' can be synchronized with r,
2. r does not happen before s',
3. if <s', r'> is a synchronization pair, then r happens before r', and
4. if a sending event s'' has the same source and destination as s' but happens before s', then there exists a receiving r'' such that <s'', r''> is a synchronization pair and r'' happens before r.

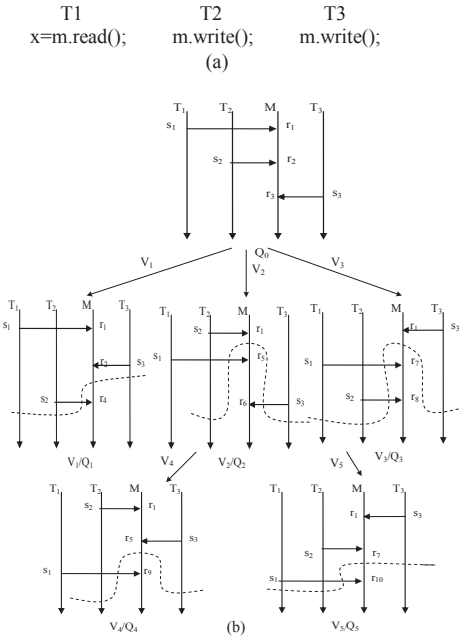


Fig.1. An example program and a reachability testing process of the program

The race set of r, denoted as *race_set*(r), is the set of sending events that have a race with s w.r.t r. As shown in Fig.1(b), in Q_0, *race_set*(r_1) is {s_2, s_3}, *race_set*(r_2) is {s_3}, and *race_set*(r_3) is empty.

A race variant V of a SYN-sequence Q is a prefix of Q by changing the sending event of one or more receiving events in Q while satisfying the following constraints for any one of these receiving events. Suppose that r is a receiving event that is synchronized with s in Q, and the sending part of r is changed to be s' in V. Then, (1) s' must be in the race set of r in Q; and (2) an event e must be not in V if r is in the control structure of e. Note that the control structure of e, denoted as *c-struct(e)*, is the set of events in Q whose existence might affect e. If e is the first event exercised by T, *c-struct(e)* is empty; otherwise, it is the prefix of Q that contains the event e' that T exercised immediately before e and all the evens that happened before e', as well as the synchronizations between them. As an example, in Q_0, *c-struct*(r_2, Q_0) is {s_1, r_1}.

For a SYN-sequence Q, one approach to generating race variants is to build a race table. Each row of a race table represents a unique race variant of Q. A race table contains a column for each receiving event in Q whose race set is not empty. Table 1 shows the race table for Q_0 in Fig.1. In a race table, the value v in the row for V and the column for r indicates how r in Q is changed to generate V. When v is -1, it indicates that r is removed from V. When v is 0, it indicates that no new sending partner is specified for r. When v is greater than *0*, it indicates that the sending partner of r in V is changed to the vth event in the race set of r. Once the sending event of a receiving event r in V is modified, all the events in Q whose control structure contains r must be removed.

Table 1 Race Table for Q_0 in Fig.1

Race Variant	r_1	r_2
V_1	0	1
V_2	1	-1
V_3	2	-1

Constructing a race table can be handled as a combination issue. However, not every combination will derive a valid variant. Each variant must be checked to ensure its validity. Given a combination c, denote the component values as *c[1]*, *c[2]*, …, *c[n]*. If it satisfies all the following rules, c is valid:

1. There exists at least one value *c[i]*, $1 \le i \le n$, such that *c[i]*>0.
2. *c[i]*=-1, $1 \le i \le n$, if and only if there exists an index j, where $1 \le j \le n$ and $j \ne i$, such that *c[j]*>0 and $r_j \in$ *c_struct*(r_i).
3. If *c[i]*>0, there does not exist an index j, $1 \le j \le n$, such that *c[j]*>0 and $r_j \in$ *c_struct(s)*, where s is the *c[i]*th sending event in the race set of r_i.

Reachability testing begins by executing a program non-deterministically. As for the example, we assume it exercises SYN-sequence Q_0 in Fig.1. Then, the race set of each receiving event in Q_0 is computed to derive the variants of Q_0, namely V_1, V_2, and V_3, which are shown in Table 1. Each variant is used to perform a prefix-based test run, in which the events and the synchronizations in the variant are controlled to be replayed. Thereafter, the test run continues non-deterministically again without controlling which SYN-sequence is exercised until it ends. Prefix-based testing with V_1, V_2 and V_3 exercises complete sequences Q_1, Q_2, and Q_3, respectively. Then new variant V_4 and V_5 are derived from Q_2 and Q_3, respectively. This process continues until no new variants are derived. Finally, the reachability testing stops.

3. MIXED-WAY COMBINATORIAL REACHABILITY TESTING

Exhaustive reachability testing covers all the valid possible combinations of the race outcome changes while t-way reachability testing covers all the t-way valid combinations. Assume there exist several groups of interaction relations

among the receiving events in Q. Instead, our mixed-way reachability testing covers all the valid possible combinations of the receiving events in every group.

In this paper, we present an approach to identifying the interaction relations by analyzing communication activities. When a monitor object in a concurrent program has no direct or indirect communication activities with other monitor objects, we say there exists an interaction relation among all methods in the monitor. In monitor-based concurrent programs, communication activities mainly include synchronization between threads, such as *wait()* or *notify()*, and accessing shared data member. If there exist direct or indirect communication activities among a group of monitor objects, an interaction relation among all methods in all monitor objects of the group exists.

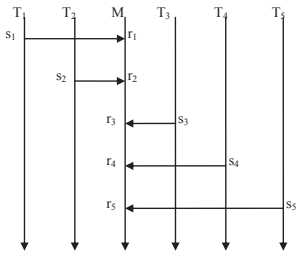
```
ConstrcutMixedWayRT (Q: a SYN-sequence, F: an interaction relation)
1. initialize table=(heading, A) to be a an empty race table, where A is a matrix;
2. R={r∈Q| | race_set(r) | >0};
3. let heading=(r1, r2, …, r|R|) be a topological order of R w.r.t the happen-
   before relation;
4. let D={d1, d2, …, d|R|}, where di=| race_set(ri) |
5. for each fk in F do
      add each l-way complete combination σ=(rk1.vk1, rk2.vk2, …, rkl.vkl) that
       satisfy rules 1, 2, and 3 into Ak, where l=|fk|, rk1, rk2, …, rkl∈ fk, and -
       1≤ rk1.vki≤ dki, and Ak is a matrix with |R| columns;
7.     add '-' into each element in Ak, which has not been assigned a value;
   end for;
8. let W=F, remove an element from W, let the element be denoted as f1;
9. let DealedEvents be {r| r∈f1}, A=A1;
10.while ( W is not empty)
11.    remove an element from W, let the element be denoted as fk;
12.    if (DealedEvents∩ fk =∅)
13.       for each row τ in Ak do
14.          if (there exists a row υ in A such that τ and υ can be directly
                merged without violating rule 2 and 3)
15.             for each i such that τ[i]≠ '-'
16.                replace each υ[i] by τ[i];
                end for;
17.          else
18.             add a new same row as τ in A ;
             end if;
          end for;
19.    else
20.       for each row τ in Ak do
21.          if (there exists a row υ in A such that τ and υ matches and they
                can be indirectly merged without violating rule 2 and 3)
22.             for each i such that τ[i]≠ '-'
23.                replace υ[i] with τ[i];
                end for;
24.          else
25.             add a new same row as τ in A ;
             end if;
          end for;
       end if;
26.    DealedEvents= DealedEvents∪ fk ;
   end while;
27. Fill an appropriate value into each element in A that is assigned with '-'
   such that it does not violate rule 1, 2 and 3;
28. return table;
```

Fig.2 Algorithm ConstructMixedWayRT

Assume *R* is a set that contains all receiving events in a SYN-sequence *Q*, and there exist *k* groups of sub-interaction relations among these receiving events. Let the *k* subsets corresponding to the *k* groups of interaction relations be denoted by f_1, f_2,.., f_k, respectively. The set $\{f_1, f_2, .., f_k\}$, denoted as *F*, is called the interaction relation of *Q*. Every element in F is called a sub-interaction relation of *Q*. These sub-interaction relations must be reduced before being used. They must satisfy the following conditions:

1. Let f_i, f_j be any two elements in *F*. If $f_i \subseteq f_j$, then f_i is removed from *F*, and vice versa,
2. For any receiving event *r* in *Q*, there exists at least one sub-interaction relation *f*, which contains *r*.

Given a SYN-sequence *Q* and the interaction relation *F*, its race table for the mixed-way combinations can be handled as a matrix, denoted as *A*. For each f_k in *F*, if there exists a sub-matrix of *A*(denoted as A_k) such that all the valid combinations of race outcomes of the receiving events in f_k occur in A_k at least once, we say that *A* satisfies f_k. If *A* satisfies all f_k in *F*, we say that *A* satisfies *F*. If *A* is the matrix that has the smallest number of rows, *A* is called an optimal race table for the mixed-way combinations.



(a)

$A^{(1)}$

r_1	r_3	r_4
1	-	-

U

$A^{(2)}$

r_1	r_3	r_4
-	0	1
-	1	-1
-	2	-1

(b)

A

r_1	r_3	r_4
-	0	1
-	1	-1
-	2	-1
1	-	-

⇨

A′

r_1	r_3	r_4
0	0	1
0	1	-1
0	2	-1
1	-1	-1

(c)

Fig. 3 Illustration of Algorithm ConstructMixedWayRT
(a) An example SYN-sequence Q (b) $A^{(1)}$ and $A^{(2)}$
(c) A mixed-way race table

Constructing an optimal race table for the mixed-way is a NP-Hard problem. As shown in Fig.2, we present a race table construction algorithm for the mixed way combination. The algorithm, named MixedWayRaceTable, adopts a merging strategy to construct an approximately optimal race table. For every sub-interaction relation, the algorithm builds one matrix that satisfies it, and thereafter merges these matrices into one

matrix, which satisfies all sub-interactions. Suppose that matrix A has m rows, the total number of complete combinations of all sub-interaction relations is c, the max number in elements in all sub-interaction relations is l. Then the worst complexity of the algorithm is $O(m \times c \times l)$. Fig.3 shows an example. There exist two groups of sub-interaction relations, i.e.$f_1=\{r_1, r_2\}$ and $f_2=\{r_3, r_4, r_5\}$. Then $race\_set(r_1)=\{s_2\}$, $race\_set(r_3)=\{s_4, s_5\}$, $race\_set(r_4)=\{s_5\}$, and $race\_set(r_2)= race\_set(r_2)=\{\ \}$. In Fig.3(b), $A^{(1)}$ and $A^{(2)}$ are generated respectively for f_1 and f_2. In Fig.3(c), A is generated by merging $A^{(1)}$ and $A^{(2)}$. Finally, A′ is returned after filling.

4. Experimental Results

The goal of our experiments was to evaluate the effectiveness of the mixed-way testing strategy, both in terms of its effectiveness for detecting errors and the reduction in the number of sequences that are exercised during mixed-way reachabilty testing, as compared to exhaustive reachability testing and t-way reachability testing. We adopted the framework of RichTest and developed a tool called MixedWayComRT, which implements our mixed-way combinatorial reachability testing algorithm.

The program under test is called MineDrainage, an industrial simulator for controlling mine drainage [1]. MineDrainage was developed in Java with nearly seven hundreds of lines. There are six monitors for representing the state of a motor, water flow, water level, CH4 level, CO level, and air flow. Accordingly, there are six threads for supervising the six monitor objects. There are two another threads: the water level processor for handling interruptions, and the main thread for starting and coordinating all the other threads. Synchronized methods in the air flow monitor generate one sub-interaction relation, and those in CO level monitor generate another one. Synchronized methods in the other four monitors generate the third sub-interaction relation.

TABLE 2 NUMBER OF SEQUENCES EXERCISED DURING TESTING

2-way	*Mixed-way*	*Exhaustive*
136	48	960

To measure the adequacy of the test sequences generated during reachability testing, we use mutation testing. A mutant introduces a single change, which simulates a program error. A mutant is killed if an execution of the mutant caused a runtime error or failed a correctness check. A mutant is said to be alive after testing if it is not killed by any test run. Less mutants alive indicates more effective testing. We generated a batch of mutants of MineDrainage using the Java-based mutation tool μJava [11]. For MineDrainage, we checked critical section and the main specifications of the program [1].

Table 2 summarizes the difference between the total number of sequences exercised during 2-way, mixed-way, and exhaustive reachability testing. We conducted 2-way and mixed-way reachability testing for 5 times. Table 2 showed the average number of sequences. Exhaustive reachability testing was conducted once. As shown in table 3, our mixed-way testing substantially reduced the number of sequences exercised as compared to 2-way and exhaustive testing.

Table 3 showed the results of 2-way, mixed-way, and exhaustive reachability testing. 2-way, mixed-way, and exhaustive testing killed all the mutants. The result showed that our mixed-way testing kept the same effectiveness of fault detection as exhaustive testing while substantially reducing the number in sequences. It is a good tradeoff between the effectiveness and efficiency of fault detection, as compared to the other two testing strategies.

TABLE 3 MUTANTS ALIVE AFTER TESTING

Mutants	*2-way*	*Mixed-way*	*Exhaustive*
185	0	0	0

Y. Lei et al presented a series of reachability testing techniques, including exhaustive reachability testing, t-way combinatorial reachability testing, and distributed reachbility testing [6-10]. Exhaustive reachability testing is often impractical. The hypothesis behind t-way reachability testing is might not be correct in many cases.

5. Conclusions

In this paper, we presented a mixed-way combinatorial strategy for reachability testing, which covers all the valid combinations of receiving events in each subsystem of a concurrent program. Our mixed-way reachability testing uses a merging method to construct a race table. The preliminary experimental results indicate that the mixed-way testing can maintain the same the effectiveness of fault detection as exhaustive testing while achieving a substantial reduction.

Acknowledgments

This work is supported by the National Science Foundation of China under Grant No.F020509, No.60873049 and No.60703086.

References

[1] A.Burns, A.Wellings. Real-Time systems and programming languages. Addison Wesley Longman, 2001.

[2] O. Edelstein, E. Farchi, et al. Multithread Java Program Test Generation. J. IBM Systems, 2002, 41(1): 111-125.

[3] C. Yang, A.L. Souter, L.L. Pollock. All-du-Path Coverage for Parallel Programs. In Proc. of International Symposium on Software Testing and Analysis(ISSTA), 1998, 153-162.

[4] R.N. Taylor, D.L. Levine, C.D. Kelly. Structural Testing of Concurrent Programs. IEEE Trans. Software Eng., 1992, 18(3): 206-214.

[5] C. Flanagan, P. Godefroid. Dynamic Partial Order Reduction for Model Checking Software. In Proc. of the 32nd Symposium on Principles of Programming Languages (POPL), 2005, 110-121.

[6] G..H. Hwang, K.C. Tai, T.L. Huang. Reachability Testing: An Approach to Testing Concurrent Software. J. Software Eng.and Knowledge Eng., 1995, 5(4): 493-510.

[7] Y. Lei, R. H.Carver. Reachability Testing of Concurrent Programs. IEEE Trans. Soft. Eng., 2006, 32(6): 382-403.

[8] R. H.Carver, Y. Lei. A Class Library for Implementing. Testing, and Debugging Concurrent Programs. Soft. Tool. Tech. Trans., 2010, 12: 69-88

[9] Y. Lei, R. H.Carver, et al. A Combinatorial Testing Strategy for Concurrent Programs. J. Software Testing, Verification, and Reliability, 2007, 17:207-225.

[10] R. Carver, Y. Lei. Distributed Reachability Testing. Concurrency and Computation: Practice and Experience, 2010, 22(18):2445-2466.

[11] http://cs.gmu.edu/~offutt/mujava

Model Driven Development for Internet of Things Application Prototyping

Ferry Pramudianto
ferry.pramudianto@fit.fraunhofer.de
Fraunhofer FIT
Schloss Birlinghoven
53754-Sankt Augustin, Germany

Indra Rusmita Indra
indrar14@gmail.com
Bonn-Aachen International Center for Information Technology
Dahlmannstraße,53113-Bonn, Germany

Mathias Jarke
jarke@informatik.rwth-aachen.de
I5 RWTH Aachen University
Ahornstr. 55
52056-Aachen, Germany

***Abstract*—We present an architectural view for the Internet of Things prototype development that emphasizes the separation of domain modeling from technological implementations. Using the provided model driven tool, domain experts are able to construct domain models by composing virtual objects and link them to the specific technologies. Having them linked, a Java prototype code can be generated by the tool. The developers may extend it into full applications simply by interfacing with the virtual objects without dealing with the communication to specific sensors and actuators. Subsequently, participants involved in the European research projects evaluated the architecture and the model driven tool using a software walk-through technique. The result shows that, for rapid prototyping, the participants are in favor of a simple domain specific language than a complex modeling language such as UML.**

Keywords-component; internet of things, architecture, domain model, code generation, model driven development, service oriented architecture.

I. INTRODUCTION

The Internet of Things (IoT) refers to an emerging paradigm which envisions seamless integration among smart physical objects, applications, and services that interact and communicate among themselves by exchanging data and information[1]. The growth of IoT community has been encouraged by the rapid development of wireless sensor and actuator networks, identification tags such as barcode and RFID, and electronic prototyping platforms such as Arduino[1]. Nonetheless, IoT is still very young research field where researchers and industry are still trying to find a common ground to establish standardized approaches. This has made IoT prototype development challenging.

According to our interviews to the developers involved in several IoT research projects in Europe, they often face problems during IoT developments caused by the lack of technology and architecture standardization. This is caused by the existence of different visions for IoT[2]. The network-oriented vision focuses on the communication for IoT devices. The "Thing" vision focuses on identification through ID tags. The semantic oriented vision focuses on processing the massive information generated by the IoT. Despite several IoT architectures exist there is still an open question on how the architecture reference could be designed in a way that the domain modeling could be decoupled from the implementation of specific IoT technology. Decoupling these allows the knowledge about the domain to be engineered by domain experts while the technology experts focus on addressing the implementation of the IoT technology.

Addressing this research question, this paper proposes a unique perspective on IoT architecture that separates the design of the domain model and the implementation of the IoT technology. Supporting the proposed architecture, this work also proposes a model driven development (MDD) tool for linking the domain model with the IoT implementations. Based on the model definition, the tool will generate Java artifacts consisting of the domain model as a virtualization of smart objects linked to the concrete implementation of IoT technology. This allows application developers to develop IoT applications using the virtual objects without having to deal with the complexities of any IoT technology.

II. RELATED WORK

The first use of IoT term was coined by The Auto-ID Labs in their work to solve product traceability problems for the supply chain management[3]. Together with the EPCGlobal they have proposed a standard architecture for universally identifying goods with RFID tags and a service registry network for querying information of the tagged goods through third party service providers[4] (Fig.1). However a survey claims that RFID is only a part of broader IoT vision where smart objects autonomously cooperate with each other[2].

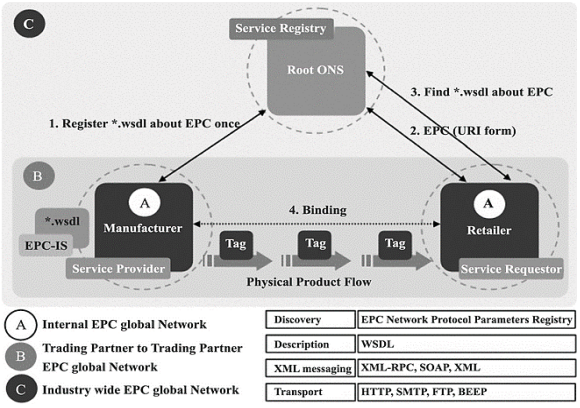


Figure 1. EPCGlobal Architecture[5]

Another survey presented a five layer architecture that placed the internet as a middle layer which functions as the main

[1] http://www.arduino.cc/

communication media (Fig.2) [6]. The edge layer manages devices such as embedded systems, sensors, actuators, and ID tags. The access gateway layer cares about bridging different communication technologies to the internet. The main task of this layer is performing a routing optimization, bridging the different communication protocols to the internet protocols (e.g. TCP/IP), and forwarding data from the edge nodes to the other end across the internet.

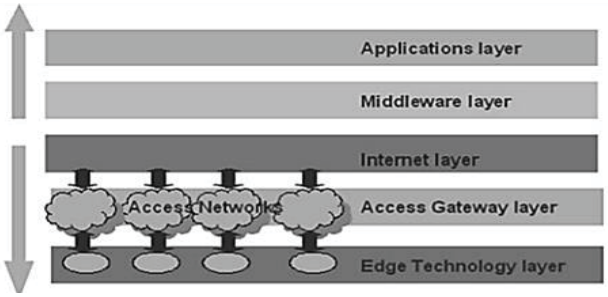


Figure 2. Generic Layered Architecture for IoT[6]

The middleware layer provides generic interfaces for the applications to communicate with the internet of things. Many approaches have been used for abstracting IoT devices e.g.: data oriented middleware uses SQL-like query languages to retrieve information from the sensor nodes, service oriented architecture (SoA) middleware has been proposed to support the integration of among "Things", legacy systems, and the necessary infrastructure while providing interoperable web services for the applications accessing them [7-9]. This layer may also perform device and information management by utilizing data fusion, semantic analysis, access control, information discovery.

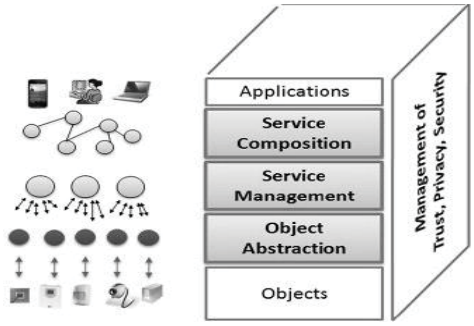


Figure 3. SOA-based architecture for the IoT middleware[2]

SoA middleware also introduces a service management layer that deals with service discovery, execution monitoring, and configuration. For the discovery purposes, a service registry is usually used. This approach provides an abstraction of various communication technology by encapsulating them with web services. SoA depends on workflow and web-service composition languages such as WSBPEL[2] to provide services that are more complex.

III. ARCHITECTURE

Our work offers a unique perspective of an IoT architecture that places the domain modeling in the center of the architecture. This is done so to emphasize the seperation of the knowledge engineering that happens during domain modeling with the technical engineering that happens during the implementations. As a result, this approach enables domain knowledge to be modeled by the domain experts that are not familiar with programming languages but familiar with languages used for managing knowledge in the domain e.g. ontology. This approach will also allow the domain model and the device implementations to evolve independently over time without having to reengineer the whole systems.

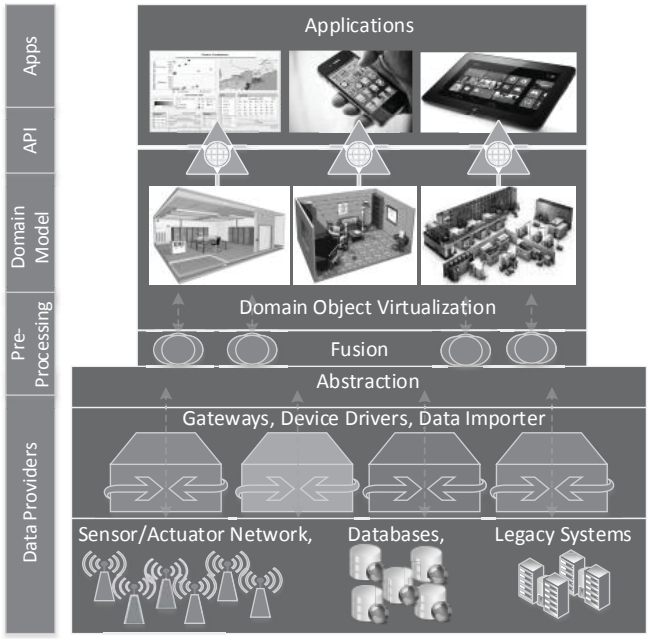


Figure 4. Architecture for object virtualization in IoT Applications

As depicted in Fig.4, the domain model layer in our architecture contains the domain knowledge such as the relationship of the objects, their capabilities and their properties as perceived by the domain experts. For instance when developing a monitoring application for a smart building, these objects may consist of the occupants, building structures (e.g. floors, rooms, windows), appliances (e.g. radio, monitor, air conditioner). Furthermore, this layer is responsible for virtualizing the physical "Things" that participate in the application domain. By virtualizing, we meant that the physical objects alone might not be able to interact with the applications without the support of devices such as sensors, actuators, and ID tags. Therefore, the representation of these objects may be composed of the supporting devices, which we refer as the enabler devices. These enabler devices should be completely transparent to the domain experts when they define the domain model.

[2] https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel

Modeling the relations among virtual objects can be done using modeling languages such as UML, Ontology, or a simplified Domain specific language (DSL) that can be understood by the domain experts. As a proof of concept, this work uses a simplified graphical DSL designed for rapid prototyping (see section 4).

On the lowest layer, the enabler devices are managed and abstracted with a common interface that is understood by the upper layer. A common approach used in the lowest layer is the use of bridge or gateways for different networks that allows communication to be estabished with applications using TCP/IP protocol. For instance in industrial automation, an OPC server is often used as a bridge to access BUS networks. This architecture pattern is also used by the emerging IoT technology such Zigbee[10] and 6LowPAN[11]. In contrast, communicating with legacy systems require various technologies which are diverse from domain to domain. Some of the legacy applications offer an application programming interface (API) in propriatery languages, some of them use database and log files to retrieve data, and some of them provide web services. In the industrial setting where Enterprise Resource Planing (ERP) and Manufacturing Execution System (MES) are invoved, ISA-95[12] is the common standard to retrieve and store data from these systems.

Additionally, since the internet and web provide a huge amount of information that can be useful, the architecture should take into account that the applications might need to access the online services such as, weather forecasts, stock prices, exchange rates, or information about people from their social network sites. For this purposes normally web service technologies can be used.

Abstracting the heterogenous technologies involved in the lowest layer may use web service technologies such as SOAP[13] and REST[14] since they offer interoperable services supported by different programming languages. However, at the moment the practicability of web service technology for time critical applications and resource constrain devices is still debatable. A more resource efficient protocols such as CoAP[15] is being developed for this purpose.

On the upper layer, the data delivered from the lowest layer sometimes must be be pre-processed to extract contextual information that is useful for the applications. This information could be as simple as determining a room temperature from the thermometer readings to a more complex information such as determining the activities, taking place in a room based on several sensor readings, or providing a contextual information of the users (e.g.: if the room temperature is too cold for this particular user). Thus, the fusion layer provides several generic pre-processing modules that are usefull for e.g.: filtering outliers, averaging the data over time and space, applying high-pass - low pass filter, interpolating the data (e.g.: Kalman Filter). The fusion layer can be extended by providing more domain specific fusion modules to derive information that is not possible to be sensed by a type of sensor.

When the domain model has been defined and the technology implementations in the first and second layer have been done, these layers need to be mapped in order to produce a functional application prototypes. The mapping follows a simple input output interaction between components in different layers and the virtual objects. For instance, the property of the virtual object "room" is linked to a processing module "average" that is linked to two thermometers. Based on this linked components, the property of the room will automatically be updated with the values coming out from the processing module, which contains the averaged temperature data coming from the two linked sensors. As the properties of the virtual objects always contain the actual values, the developers simply need to work with the virtual objects without worrying the technical details to access the thermometers.

IV. Domain Modeling Tool Development

As a proof of concept of the proposed architecture, we developed a domain-modeling tool that supports domain experts designing domain models containing virtual objects. We build the tool based on the requirements of twelve developers involved in several European research projects dealing with IoT. The requirement elicitation was done through a focus group workshop where the developers are given scenario to build IoT prototypes. Then, we discussed about a visionary tool that could help them solving their tasks rapidly. The outcome from the focus group reveals that the developers are in favor of a graphical model driven tool that would help the domain experts designing the domain model and then allows them to map the virtual objects to the sensor and actuators. They would like to have the domain model defined using simple notations that can be quickly explained to non-computer scientist users. Finally, the tool should generate a Java code that can be extended to develop their final applications. After the requirements are collected, we designed the user interface mockup and iteratively evaluated it with the users who.

A. User Interface Mockup

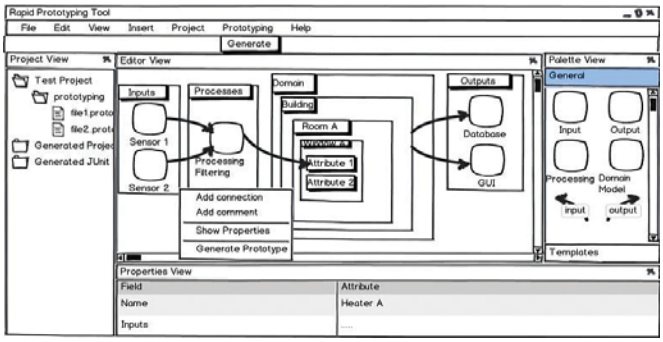


Figure 5. The mock up user interface for the development tool

The mockup GUI composed of several views including Project View, Editor View, Palette View, and Properties View (fig. 5). On the toolbar, there is a button to generate the necessary Java artifacts from the defined model including the Java code of the virtual objects, the mappings to the physical objects, and the library to access the physical objects. The Project View may contain several projects. Each project may consist of several domain model diagrams, which each of them contains the definition of the virtual objects, their properties, and links to the processing modules, data providers, and actuators. We created a set of simple notations to simplify defining a domain model and the mappings to sensors and actuators. However, our notations are not as expressive as UML and Ontology as it is not intended

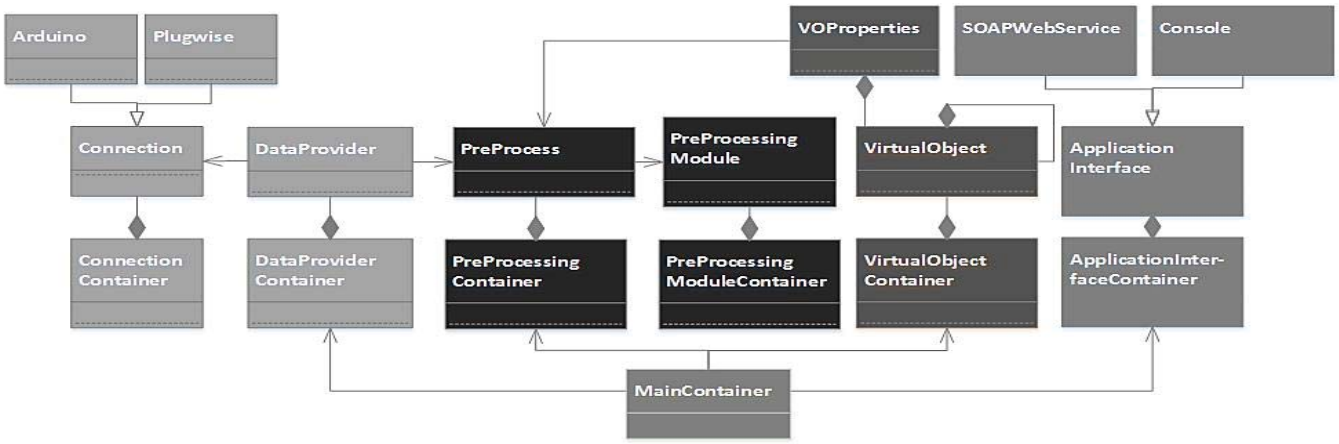


Figure 7. ECore Model of the Proposed Tool

to support development for complex applications. These notations are presented in the Palette view. The notations consist of rectangles that can be containers for other rectangles and linked using arrows to map the relationship among the objects. These atomic notations are grouped using a tabular menu depending upon the type of the notations. The Editor View is the main container where the developers could define the domain model using the provided notations. In the Property View users can modify the properties of the notations depending the type of the notation.

B. Modeling Tool

The modeling tool is built with the Eclipse Modeling Framework (EMF)[3]which is responsible for defining the meta model of the proposed architecture depicted in Fig.4. We use Graphical Modeling Framework (GMF)[4] for generating the MDD editor used for our domain specific language. Firstly, the tool was built by defining the *ECore* model, which is needed by EMF(Fig.7) as the meta model for the tool. The *ECore* model was defined to have a main container and several containers. These containers include containers for the Data Provider, Pre-Processing, Virtual Object, and Application Interface. Each of these containers could contain more than an implementation of the abstract classes depicted in the center row of Fig.7. We implemented the abstract classes as examples that are useful for evaluating the tool against the user requirements. These abstract classes are extendable when further IoT technologies to be added in the future.

After creating the meta-model, we use the EMF Generator Model to generate the plugin projects that we need to implement such as the "Model Code", the "Edit Code", the "Editor Code", and the Java Interfaces. After the skeleton is generated, we used GMF to create a diagram editor using GMF Tooling. We edited the *gmfgraph* to define the graphical notations and *gmftool* to define the tooling of the editor such as menu, and palate. Moreover, in the *gmfmap,* we mapped the notations to the domain model of the tool defined by the EMF. We use EMF also to provide serialization of the model defined by the users. Currently it only supports XMI[5] format, which can be stored and opened back to the editor view when the users want to continue working on them. The serialization can be done in other formats, however for the sake of simplicity we took the standard format provided by EMF.

The base classes are implemented as Eclipse plugins. This provides flexibility when further components need to be integrated into the tool. For instance the Connection base is implemented as an eclipse plugin which is extended by two other eclipse plugins containing implementations to create connections to the corresponding devices. These base classes provide an abstract factory to be used by the wizards in the eclipse IDE for retrieving the actual implementations of the plugins.

C. The code generator

The code generator is implemented using Xpand for generating Java code and the necessary artifacts based on a set of template codes. The template codes contain all implementations of the abstract classes defined in the ECore model that take into account adjustments that the users will define when modeling the prototype applications. These adjustments include adjusting the package and class names, assigning the values of the sensors to processing module, assigning the output of the processing modules to the properties of the virtual objects. These adjustments will be generated by the Xpand plugins that we have developed. After the template code is adjusted and generated, XPand additionally generate an eclipse Java Project and all the necessary artifacts such as libraries and a run configuration that the users need to run the generated project properly.

In the current implementation, the generated java project will consist of the chosen connections. There are two connections supported Plugwise[6] and Arduino which are connected through the serial ports.

D. The workflow of the tool.

When developing a new application prototype with the tool, several steps as depicted in Fig. 8 must be followed. The users start with creating a new project and entering its name. Next, the users create a new domain model diagram and enter its name. Then, the users can start designing the domain model on the

[3] http://www.eclipse.org/modeling/emf/

[4] http://www.eclipse.org/modeling/gmp/

[5] http://www.omg.org/spec/XMI/

[6] http://www.plugwise.com

editor view as depicted in fig. 12(B). Designing the domain is started by adding the main container for the virtual objects, and the virtual objects themselves in the container. The virtual objects can contain other virtual objects that denote "a part-of" relationship.

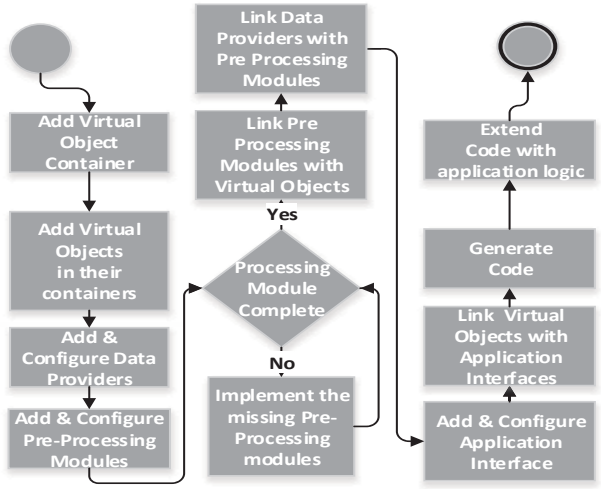


Figure 8. A simplified workflow of the development with the tool

After the domain model is defined, the users could add predefined processing modules. In case the needed modules is not available, software developers may implement the modules by extending the corresponding plugins. When the needed processing modules have been added to the Editor View, the users link the properties of the virtual objects to the modules and then the modules also have to be linked to the data providers. Finally, the users add the application interface such as SOAP, or REST, and link the application interface to the virtual objects so that the tool knows which virtual objects it needs to expose to the applications and with which technology it should be done. After all necessary associations have been done, the user can generate the Java project that can be run and accessed from their application through the chosen application interfaces.

V. EVALUATION

A software walkthrough [16] was performed to evaluate the users acceptance to the proposed architecture and the proposed MDD tool. The evaluation was done with 7 participants (6 male and a female). Six of them are system developers who are working in Fraunhofer FIT and participate in European research projects dealing with IoT implementations in different domains. A participant is a student of the technical university of Aachen (RWTH Aachen) who also works in an IoT project at Fraunhofer FIT. Their experience in application developments range between 2 until 6 years. The participants are between 25 and 35 years old.

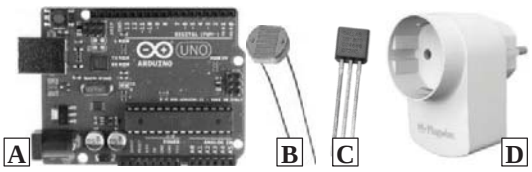


Figure 9. Arduino Board(A), Light Sensor(B), Digital Thermometer (C), Plugwise (D).

The equipment used to perform the evaluation consists of an Arduino board (Fig. 9. A), a light intensity sensor (Fig. 9. B), a digital thermometer (Fig. 9. C), and a Plugwise (Fig. 9. D). The Arduino board was used to retrieve data from the light and temperature sensors and send them through a serial port. To communicate with Plugwise, a Zigbee USB receiver was used.

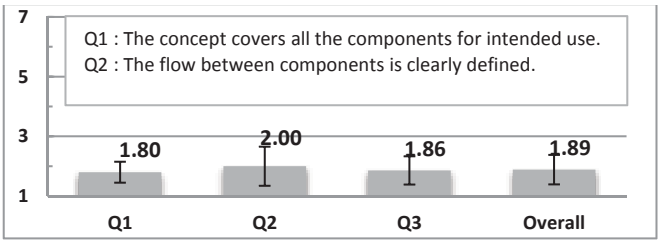


Figure 10. User satisfaction of the proposed architecture

The participants were given a task to display sensor values from a temperature, a light sensor attached to an Arduino board, and the power meter. After they had performed the task, they were given a questionnaire to review the proposed architecture, and the tool. To ensure the result of the study could be compared to similar works in the future, some questions of the questionnaire were taken from the IBM Computer System Usability Questionnaire[17]. The questions were presented with 7-level Likert-scale options where "7" denotes "Strongly Disagree" down to "1" which means "Strongly Agree".

As depicted in Fig. 10, the result of the questionnaire regarding the architecture shows that the users felt it was easy to understand *(M=1.86, SD=0.64)* and its functions were clearly defined *(M=2, SD=0.82)*. Secondly, the proposed architecture helped the user developing the intended functional prototype *(M=1.8, SD=0.4)*. Overall the users were satisfied to the architecture design *(M=1.89, SD=0.67)*.

Furthermore, to investigate the user satisfactions to the overall work, the participants were given a task to solve with the proposed tool and Eclipse's EMF tool. The order of the tool used by each participant was exchanged to minimize the learning effect. Then the users were asked to rate the overall experience working with the tool using DSL notations compared to the EMF using UML notations.

The questions were divided into four categories that include the overall experience with the framework (Overall), the functionalities of the tool (Tool Functions), the workflow to be done when working with the tool (Workflow), and the user interface of the tool (UI). The questions are again adopted from the [17].

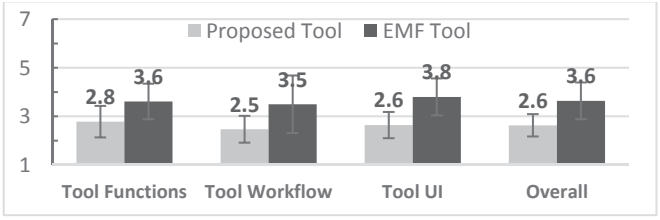


Figure 11. Comparisson between EMF tool & IoT Modeling Tool

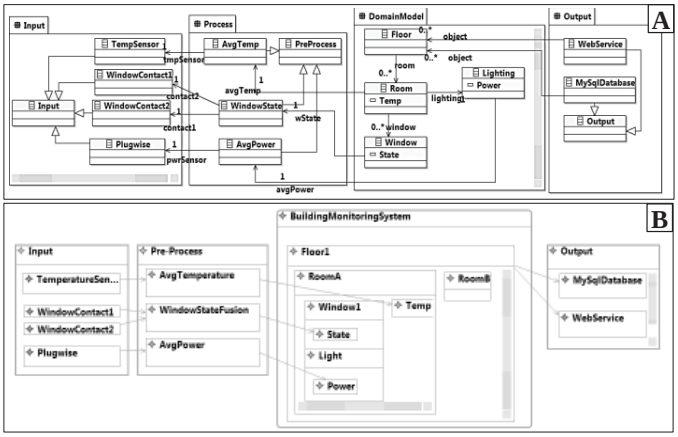


Figure 12. EMF with UML notation (A) vs. IoT Modeling Tool with DSL notation (B)

The score comparison between our work and EMF Tool are presented in Fig. 11. Overall, the writer's work scored better compare to the EMF tool with an UML diagram.

A paired sample T-Test analysis was performed to investigate if the difference between user's satisfaction to the writer's work and EMF tool is statistically significant. The result of the questionnaire shows that even though the proposed tool scored a better means in all categories there was no significant differences of user satisfaction for the Functions, Workflow, and Overall [$T(7)=1.2$, $p>.5$), ($T(7)=-1.38$, $p>.5$), ($T(7)=-2.04$, $p>.5$) respectively]. Interestingly, the user opinions were significantly affected by the user interface of the tools ($T(7)=-2.66$, $p<.5$).

The fact that the user opinions are affected by the user interface indicates that a simplified domain specific language serves a better purpose for simple prototyping tasks than complex modeling languages such as UML since the users are faced with simplicity and less options that may overwhelm them in solving the tasks.

VI. Conclussion & Future Work

The current approaches of IoT architectures have overlooked the importance of domain modeling in the application development. This work has presented a unique perspective that positions domain modeling and object virtualization in the center of the architecture. Moreover, this work has proposed a tool that augments the proposed architecture by allowing the domain model to be linked to the IoT implementations. Consequently, the complexity of IoT implementations is transparent for the application developers, as they only need to work with the virtual objects generated by the tool. The results of the preliminary evaluation support our claim that the proposed architecture and the model driven tool have a potential to ease IoT application development.

The next steps for this work are to provide an easy way for the domain experts to express their domain knowledge related to policies and rules that are difficult to express through a graphical notations. For this purpose, the tool could be extended by integrating a rule engine in the generated code. This approach allows policies to be dynamically modified without recompiling the application. Once the tool has enough features to support complex application developments, we also would like to perform evaluation with more users in a longer period of time.

VII. Acknowledgement

This work was co-funded by the European Commission through EBBITS (FP7-ICT-2009.1.3, GA No. 257852) and BEMOCOFRA (FP7-ICT-2011-EU-Brazil, GA No. 288133)

VIII. References

[1] Guillemin, P., and Friess, P.: 'Internet of Things: Strategic Research Roadmap', CERP-IoT Project, 2009

[2] Atzori, L., Iera, A., and Morabito, G.: 'The internet of things: A survey', Computer Networks, 2010, 54, (15), pp. 2787-2805

[3] Bose, I., and Pal, R.: 'Auto-ID: managing anything, anywhere, anytime in the supply chain', Communications of the ACM, 2005, 48, (8), pp. 100-106

[4] Johnson, F.A.J., Harrison, M., US, B.H.G., Mitsugi, J., Preishuber, J., CVS, O.R., and Suen, K.: 'The EPCglobal Architecture Framework', 2005

[5] Shih, D.-H., Sun, P.-L., and Lin, B.: 'Securing industry-wide EPCglobal network with WS-security', Industrial Management & Data Systems, 2005, 105, (7), pp. 972-996

[6] Bandyopadhyay, D., and Sen, J.: 'Internet of Things: Applications and Challenges in Technology and Standardization', Wireless Personal Communications, 2011, 58, (1), pp. 49-69

[7] de Souza, L., Spiess, P., Guinard, D., Köhler, M., Karnouskos, S., and Savio, D.: 'Socrades: A web service based shop floor integration infrastructure', The Internet of Things, 2008, pp. 50-67

[8] Jammes, F., and Smit, H.: 'Service-oriented paradigms in industrial automation', Industrial Informatics, IEEE Transactions on, 2005, 1, (1), pp. 62-70

[9] Eisenhauer, M., Rosengren, P., and Antolin, P.: 'A development platform for integrating wireless devices and sensors into ambient intelligence systems', in Editor (Ed.)^(Eds.): 'Book A development platform for integrating wireless devices and sensors into ambient intelligence systems' (IEEE, 2009, edn.), pp. 1-3

[10] Alliance, Z.: 'Zigbee specification', ZigBee document 053474r06, version, 2006, 1, pp. 378

[11] Mulligan, G.: 'The 6LoWPAN architecture', 'Book The 6LoWPAN architecture' (ACM, 2007, edn.), pp. 78-82

[12] Scholten, B.: 'The road to integration: A guide to applying the ISA-95 standard in manufacturing' (Isa, 2007. 2007)

[13] Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.-J., and Nielsen, H.F.: 'Simple object access protocol (SOAP) 1.2', World Wide Web Consortium, 2003

[14] Fielding, R.T.: 'Chapter 5: Representational State Transfer (REST)', Architectural Styles and the Design of Network-based Software Architectures, Dissertation, 2000

[15] Shelby, Z., and Team, C.A.: 'Constrained Application Protocol (CoAP) draft-ietf-core-coap-04', IETF work in progress, 2011

[16] Spencer, R.: 'The streamlined cognitive walkthrough method, working around social constraints encountered in a software development company', 'Book The streamlined cognitive walkthrough method, working around social constraints encountered in a software development company' (ACM, 2000, edn.), pp. 353-359

[17] Lewis, J.R.: 'IBM computer usability satisfaction questionnaires: psychometric evaluation and instructions for use', International Journal of Human‐Computer Interaction, 1995, 7, (1), pp. 5

Pattern-based Decentralization and Run-time Adaptation Framework for Multi-site Workflow Orchestrations

Selim Kalayci, S. Masoud Sadjadi
School of Computing and Information Sciences
Florida International University
Miami, FL, USA
{skala001, sadjadi}@cs.fiu.edu

***Abstract*—Scientific applications keep getting more complex, resulting in the need for more computational resources than may be available to scientists locally. As a result, for many scientists utilization of remote and heterogeneous computational resources has become a standard practice. However, effective utilization of these resources, especially for large-scale workflow applications, necessitates the employment of software tools that are efficient and adaptive. In this study, we propose a generic framework for the decentralization and run-time adaptation for the execution of large-scale workflow applications that span across diverse and heterogeneous resource domains. By exploiting the recurring DAG patterns, we come up with corresponding decentralization and adaptation patterns to be employed by peer workflow orchestration tools to cope with various resource-based challenges in the execution environment. Our framework adopts separation of concerns and consequently does not alter the business logic of the application. We provide a prototype implementation of our framework on a standard workflow orchestration tool. But, our framework is generic enough and can be easily incorporated by other orchestration tools.**

Keywords: *workflow, DAG, orchestration, adaptation, pattern*

I. INTRODUCTION

Scientific workflows are abstractions that capture the business logic of many complex applications in different scientific disciplines. More specifically, these workflows encapsulate and represent all of the various tasks and data artifacts associated with the application lifecycle. Regardless of the size and complexity of the specific scientific workflow, Directed-Acyclic-Graphs (DAGs) are a powerful and well-established method used by many scientists/developers.

Lifecycle of a typical scientific workflow begins with the specification of individual tasks and artifacts, and dependencies among them. This specification is free of some concrete run-time specific details, and it is mostly referred as the abstract workflow. Abstract workflows more often than not do not address run-time specific details, such as the exact names and locations associated with data artifacts and details about the exact mapping of tasks on physical resources. For such an abstract workflow to run successfully to completion, those details need to be determined prior to or during the execution of the workflow. This process can be referred to as the concretization of the workflow. However, the concretization of the workflow should not be the responsibility of the scientist, or even the application developer; as a matter of fact, it has to be automated as much as possible based on the criteria provided by the user.

During the concretization process, one essential step is the mapping of workflow tasks onto physical resources. The main goal here is to achieve the minimum makespan possible for the execution of the whole workflow. As such, characteristics of tasks (e.g. estimated runtime) and data artifacts (e.g. estimated size), as well as the availability and characteristics of physical resources play a major role during this mapping process. Based on the availability of resources, the resulting concrete workflow may span across multiple sites of resources. Such multi-site resources may be made available to the usage of a specific workflow application perhaps through research collaboration among multiple partners or through a national/international cyberinfrastructure platform (e.g. XSEDE). The key common attributes of such multi-site resources is the heterogeneity and dynamicity of the resources in terms of size and capability, as well as heterogeneous access and priority rights assigned to the users by local administrators. All these factors pose many challenges to the successful execution of workflows conforming to the makespan requirements of the user.

In this paper, we propose a generic framework that utilizes the common DAG patterns to alleviate some of the problems stemming from the multi-site execution of workflows. First of all, taking advantage of DAG patterns, we transform a concrete workflow in a manner that makes it possible to be executed in a peer-to-peer fashion. This transformation has the potential to improve the efficiency of the execution of the workflow by having local interactions to be managed by local workflow execution managers rather than having a single central workflow execution manager orchestrating the whole workflow. Also, employment of local managers improves the accuracy and efficiency involved within the decision-making and adaptation process to the changes in the execution environment at run-time. Second major contribution of this paper is to propose a generic framework for the adaptation of the workflow execution in response to the dynamic changes at run-time. This adaptation framework also utilizes common

DAG patterns and suits well with the peer-to-peer execution framework devised in the previous step.

A very important aspect of our proposed framework is the separation of concerns. The modifications and adaptations to the execution logic of the workflow due to our approach, do not affect the business logic of the workflow. We showcase the validity and generality of our approach via a prototype implementation on a standard workflow execution manager. Some of the implementation details are also discussed briefly.

To illustrate our ideas, first we introduce the concept of DAG patterns in Section 2. In Section 3, we introduce our decentralization framework utilizing DAG transformation patterns. In Section 4, we discuss the need for run-time adaptation and explain how we incorporate this behavior into our decentralized execution environment. Section 5 illustrates and discusses prototype implementation issues. In Section 6 we overview related literature and Section 7 provides a summary of the paper and a brief discussion about further issues.

II. DAG PATTERNS

In this section, we introduce the concept of recurring DAG patterns that form the basic building blocks for the steps that follow. The key point here is that all possible scientific workflows in DAG form can be represented using a proper combination of these DAG patterns.

Fig. 1 illustrates three DAG patterns, namely: Sequence pattern, Fork/Branch pattern and Join pattern. In these graphs, vertices correspond to the computational tasks, whereas directed edges correspond to the control and/or data dependencies between tasks.

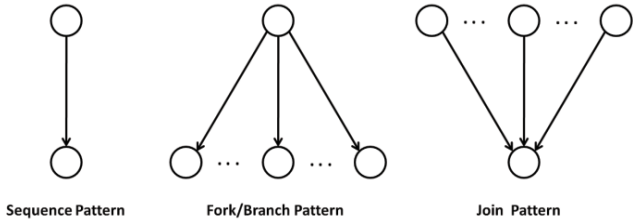


Figure 1. DAG patterns

III. DECENTRALIZATION

Regardless of being in an abstract or concrete form, a scientific workflow is usually crafted and enacted at a single central location. This means, except for a few proprietary solutions, the execution logic of the workflow is handled by a single, central workflow execution manager. This central manager keeps track of the progress of tasks and coordinates the timely execution of each task based on the specifications of the workflow.

The central workflow manager handles the execution of the workflow even if the mapped tasks of the workflow span across multiple sites. Especially in such a scenario, employment of a single central workflow execution manager raises several efficiency and decision-making accuracy issues. First of all, employing a single central manager necessitates each individual activity to be monitored and orchestrated by this central manager. For a large-scale workflow (i.e. comprised of a large number of tasks) that is mapped on multiple and potentially long-distance sites, orchestration efficiency becomes a real issue. Another important issue is the level of information sharing among partnering sites. The more detailed resource and workload information is shared among partners, the better decisions can be made by the workflow execution manager(s) to take actions in response to changing conditions. However, due to administrative and technical reasons (e.g. size of information, delay); it is not possible for a remote workflow execution manager to have the same level of information about a certain site resources compared to its local counterpart. This can cause the central workflow execution manager to make non-optimal decisions during run-time adaptation.

To overcome these issues, our proposal is to transfer the responsibility of orchestration of the whole workflow from a single workflow execution manager to several collaborating workflow execution managers. According to this, each local workflow execution manager is going to be responsible for the orchestration of the tasks that are mapped locally. At the same time, peer local workflow execution managers synchronize among each other when necessary, specifically, to fulfill the requirements of those control/data dependencies that span across them. By collaboratively carrying out these activities, the orchestration of the whole workflow is achieved without affecting the business logic of the workflow. Through this peer-to-peer orchestration approach, we are able to provide (i) improved efficiency for large-scale workflow executions, (ii) better results from run-time adaptation.

We propose a systematic and generic framework for transforming the centralized orchestration to a collaborative orchestration via the utilization of DAG transformation patterns. These patterns are built on basic DAG patterns introduced in Section 2, and illustrate the transformations on the original DAG specifications to meet the needs of the collaborative orchestration style. Each local workflow execution manager individually performs these transformations appropriate to its circumstances.

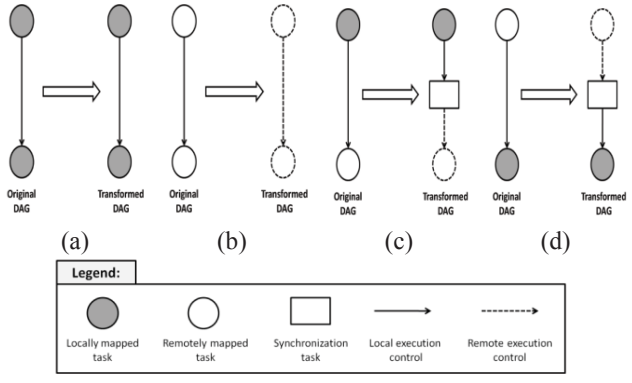


Figure 2. DAG Transformations for the Sequence DAG Pattern

Fig. 2 illustrates the set of DAG transformations on the Sequence DAG pattern corresponding to four possible mapping scenarios. If both tasks comprising the Sequence DAG pattern are mapped locally, as in Fig. 2(a), then no transformation is

necessary. If both tasks comprising the Sequence DAG pattern are mapped remotely, as in Fig. 2(b), then these tasks are marked to indicate that they will be orchestrated by another manager. Fig. 2(c) illustrates the case where the parent task is mapped locally, whereas the child task is mapped remotely. In this case, the transformation incorporates a synchronization task between these tasks. The purpose of this synchronization task is to basically inform the remote workflow execution manager of the completion of the parent task. Fig. 2(d) illustrates the case where the parent task is mapped remotely, and the child task is mapped locally. In this case, the transformation again incorporates a synchronization task between these tasks. However, in this case the local workflow execution manager waits to be informed by its remote partner about the completion of the parent task.

Fig. 3 illustrates the DAG transformations on the Fork/Branch DAG pattern corresponding to two possible mapping scenarios. We skip the two other possible mapping scenarios, in which all the set of tasks are either mapped locally or remotely, as the transformations will be very limited and done similar to the ones in Fig. 2(a) and Fig. 2(b). Fig. 3(a) illustrates the case where the parent task is mapped locally, whereas the children tasks are mapped remotely. In this case, the transformation incorporates a synchronization task after the parent task. This synchronization task will be informing the remote workflow execution manager(s) of the completion of the parent task. Fig. 3(b) is similar to the previous scenario, but this time parent task is mapped remotely and children tasks are mapped locally. This time, the local workflow execution manager is at the receiving side of the synchronization operation, so it has to wait for its remote partner to perform the associated synchronization task.

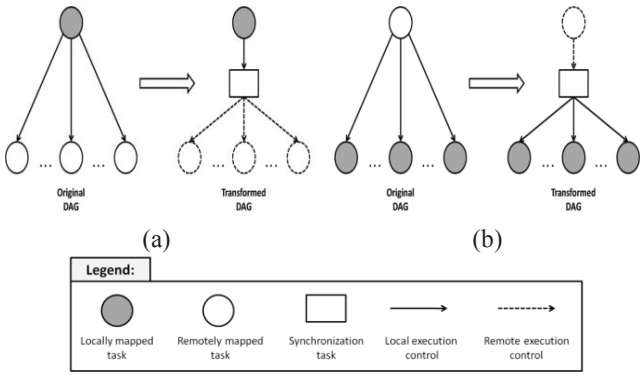


Figure 3. DAG Transformations for the Fork/Branch DAG Pattern

Fig. 4 illustrates the DAG transformations on the Join DAG pattern corresponding to two possible mapping scenarios. As in Fig. 3, we skip the two other possible mapping scenarios. Fig. 4(a) illustrates the case where the parent tasks are mapped locally, and the child task is mapped remotely. In this case, the transformation incorporates a single synchronization task before the child task to inform the completion of the parent tasks. According to the scenario in Fig. 4(b), parent tasks are mapped remotely and the child task is mapped locally. This time, the local workflow execution manager is at the receiving side of the synchronization operation, so it has to wait for its remote partner(s) to perform the associated synchronization task.

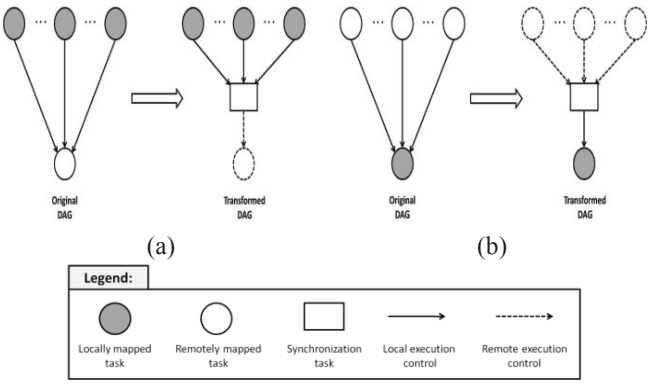


Figure 4. DAG Transformations for the Join DAG Pattern

One thing to note for the transformation of Fork/Branch and Join DAG patterns is that we incorporate a single synchronization task among the parent and children tasks, regardless of the number of parent and children tasks in the original DAG. This design choice significantly reduces the number of synchronization activities (hence overhead) among peers, and also simplifies the transformation process.

IV. RUN-TIME ADAPTATION

Due to the dynamic nature of the execution environment, certain changes may need to be made to the original workflow execution plan at run-time to meet users' QoS requirements. The most common and obvious dynamic change in the execution environment is the availability of hardware resources for the utilization of workflow tasks. The availability of these resources may change basically due to hardware failures, increased workload, and higher priority tasks being deployed in the system. Especially long-running and large-scale workflows are highly susceptible to this kind of changes in the execution environment. Under these circumstances, to be able to execute a workflow successfully within QoS requirements, proper changes have to be made to the original execution plan.

There are two main issues involved within the run-time adaptation process. First major issue is the planning phase for the run-time adaptation. This phase includes the continuous monitoring of workflow progress and resources, and detecting a situation that necessitates the run-time adaptation process. After the detection, an appropriate corrective action has to be planned to cope with the situation. Second major issue in the run-time adaptation process is the enactment of the proposed adaptation plan to the ongoing workflow execution process in an efficient and non-intrusive manner. In this paper, we focus on the second aspect of the run-time adaptation process.

A standard run-time adaptation plan basically makes changes to the original execution plan by modifying the mapping (hence, the execution) site of tasks. Modifications to the mapping site of tasks needs to be reflected and implemented accordingly by the workflow execution manager(s). Here, we provide a run-time adaptation framework that integrates with the decentralization framework explained

in the Section 3. One key aspect of our framework is the low-level of intrusiveness to carry out the adaptation process. Our pattern-based framework has little effect on the ongoing workflow execution process. The peer workflow execution managers implement the re-mapping of tasks without any disruption to the orchestration of the rest of the whole workflow.

We utilize DAG adaptation patterns at peer workflow execution managers to implement the re-mapping of tasks at run-time. Through this adaptation, the responsibility of orchestration of the set of tasks being re-mapped (S_T) is transferred from the originating site to the destination site(s).

At the originating site, DAG adaptation patterns transforms S_T from being local tasks to remote tasks. Also, if there is a synchronization task between S_T and child(ren) task(s), it is removed. Fig. 5(a) illustrates this scenario for the Sequence pattern, Fig. 6(a) illustrates the same scenario for the Fork/Branch pattern, and Fig. 7(a) illustrates the same scenario for the Join pattern. But, if the child(ren) task(s) of S_T is (are) mapped locally, then a synchronization task is incorporated after S_T. Fig. 5(b) illustrates this scenario for the Sequence pattern, Fig. 6(b) illustrates the same scenario for the Fork/Branch pattern, and Fig. 7(b) illustrates the same scenario for the Join pattern.

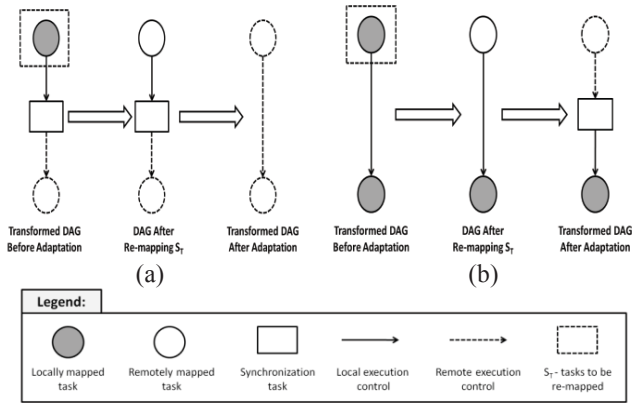


Figure 5. DAG Adaptations for the Sequence DAG Pattern

The destination site(s) basically captures the re-mapped tasks (S_T) and orchestrates them in accordance with the resulting DAG structure. However, attempting to capture and integrate this DAG structure with the transformed DAG specification at destination site(s) proves to be cumbersome to design and implement. For this reason, we propose the orchestration of these DAG structures in isolation from the transformed DAG structure(s) at destination site(s). In fact, we refer to these DAG structures that result from run-time adaptation processes and orchestrated in isolation as "patch DAGs". The combined orchestration of patch DAGs with the transformed DAGs provide the same business logic as the orchestration of the whole original DAG.

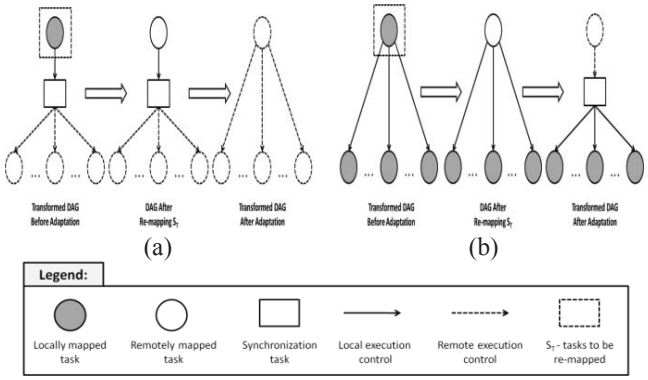


Figure 6. DAG Adaptations for the Fork/Branch DAG Pattern

The essential duty of a destination site involved in the run-time adaptation process is to capture S_T and compose the corresponding patch DAG. Once the patch DAG is ready, destination site orchestrates the patch DAG in isolation.

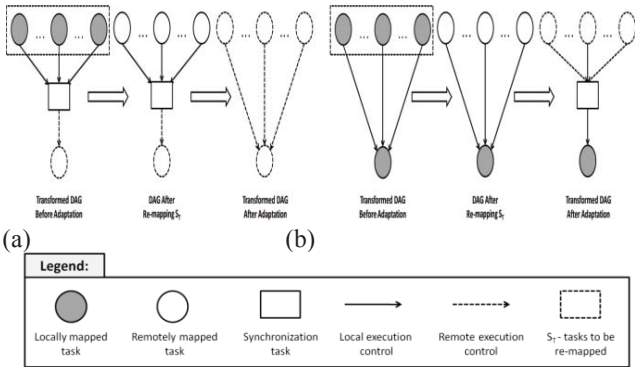


Figure 7. DAG Adaptations for the Join DAG Pattern

Fig. 8 illustrates the corresponding patch DAGs for the adaptation of Sequence patterns in Fig. 5. In both Fig. 8(a) and Fig. 8(b), the corresponding patch DAG has the same structure. The difference between them is the inner-workings of the synchronization task. In Fig. 8(b), the synchronization task synchronizes back with the originating site for the execution of child task. On the other hand, synchronization task in Fig. 8(a) synchronizes with a third site other than the originating site. In fact, this third site is the one that is responsible for the execution of the child task, and it is agnostic to the changes made during the adaptation.

Patch DAGs corresponding to the adaptation of Fork/Branch patterns look and operate the same way as in the adapted Sequence patterns. The only difference between these two patterns is the number of children tasks the patch DAG synchronization task enables for progression.

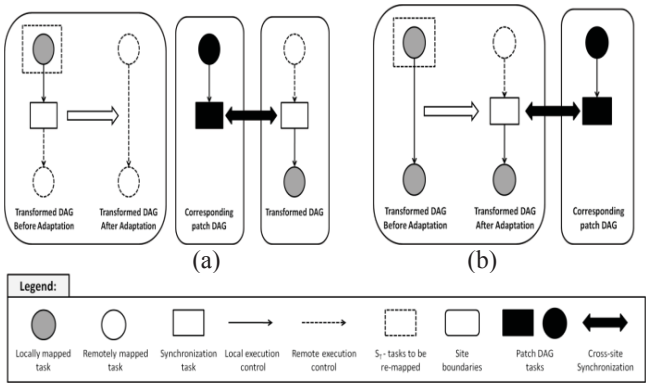


Figure 8. Patch DAG Patterns corresponding to the Adapted Sequence Patterns

The corresponding patch DAG structures to the adaptation of Join Pattern in Fig. 7(a) and Fig. 7(b) are also identical. As in the Sequence pattern, the difference between them is the inner-workings of the synchronization task. However, in a Join pattern, it is possible for more than one site to be involved in the re-mapping of S_T, due to the multiplicity of the number of tasks in S_T. To illustrate this point, we provide two alternative scenarios for patch DAG composition and operation corresponding to the case in Fig. 7(b). Fig. 9 illustrates the corresponding patch DAG structure in a scenario where only one site gets to re-map all the tasks in S_T. For this scenario, the composition and the operation of the patch DAG is quite similar to the previous patterns.

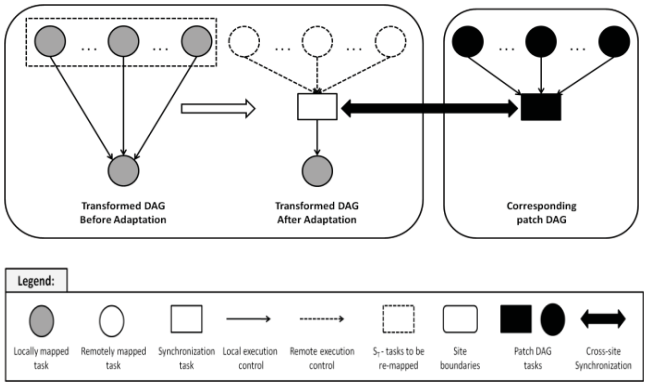


Figure 9. Patch DAG Pattern corresponding to the Adapted Join Pattern in a scenario where a single site re-maps S_T

Fig. 10 illustrates the corresponding patch DAG structure in a scenario where two destination sites are involved in re-mapping of the tasks in S_T. In this scenario, two patch DAGs need to composed, and one of those DAGs is designated as the primary patch DAG. Primary patch DAG is responsible for handling the synchronization with the originating site. Also, in this scenario, an additional layer of synchronization is needed between the destination sites.

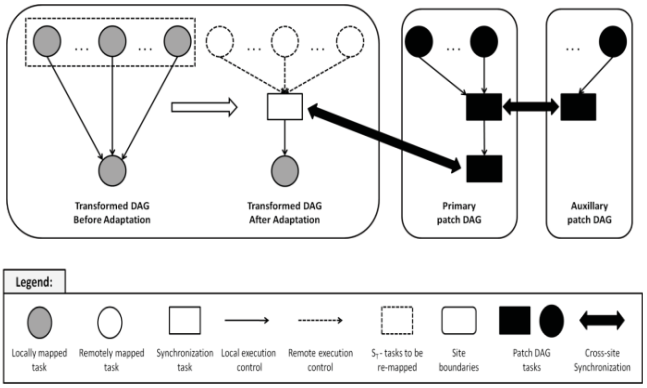


Figure 10. Patch DAG Pattern corresponding to the Adapted Join Pattern in a scenario where two sites re-map the tasks in S_T

V. PROTOTYPE IMPLEMENTATION

Our implementation is based on Condor DAGMan [5] workflow execution engine. Condor DAGMan is a widely used workflow execution tool that is available in most High-Performance and High-Throughput Computing environments. The original orchestration architecture is centralized and it does not provide native run-time adaptation support.

Decentralization

Condor DAGMan specifies a DAG structure by listing the tasks and their dependencies in a standard text file. This specification is parsed and then the execution of tasks is orchestrated by submitting them to local and/or remote resources by Condor DAGMan.

In our decentralized framework, each site has its own deployment of Condor DAGMan tool, and each engine submits computational tasks only to their local resources. Synchronization among peer engines is accomplished via transferring light-weight sync files.

Each local Condor DAGMan tool needs to orchestrate a transformed copy of the original DAG specification. The transformation process is implemented by performing the following basic actions. Remotely mapped tasks are labeled as DONE, so that Condor DAGMan will not attempt to submit them to local resources. DAG transformations illustrated in Fig. 2(c) and Fig. 3(a) is accomplished via insertion of a POST Script accompanying the parent task. DAG transformations illustrated in Fig. 2(d) and Fig. 3(b) is accomplished via insertion of a PRE Script accompanying the child(ren) task(s). DAG Transformations for the Join Patterns illustrated in Fig. 4 is accomplished via incorporation of a light-weight sync task in the original DAG specification.

Run-time Adaptation

As a fault-tolerance feature, Condor DAGMan provides a checkpoint-style recovery mechanism via the creation of rescue DAGs. The rescue DAG represents the specification of a DAG that was not run to completion, but specifies the completed tasks as DONE so that they will not need to be re-run. At the originating site, we utilize the rescue DAG mechanism to halt the DAG specification prior to the adaptation and then to re-enact the corresponding DAG specification after adaptation. To

generate the proper corresponding DAG specification, adaptation decisions should be reflected on the standard rescue DAG specification. This is achieved via labeling re-mapped tasks as DONE and also performing similar DAG transformation activities as was done for Decentralization purposes.

At the destination site, corresponding patch DAG specification needs to be generated. DAG pattern of the task(s) received from the originating site gives enough information to accomplish this. The destination site can then perform the same kind of DAG transformation activities to generate the corresponding patch DAG specification. Following this, destination site parses and starts orchestration of the patch DAG at its local site in isolation.

For more details about the prototype implementation and system design issues, please refer to [13].

VI. Related Work

Pegasus workflow management system [2] uses the Condor DAGMan [5] as the underlying workflow execution engine. Other DAG-based workflow management systems like Taverna [10] and GrADS [11] also employ centralized execution model. ASKALON workflow management system [4] is comprised of a hierarchical architecture where a master engine provides all enactment decisions whereas multiple slave engines perform the orchestration of the workflow under the administration of the master engine.

In the business process orchestration field, several studies [8,9] propose decentralization mechanisms for centralized BPEL process execution engines for scalability and performance improvement. Also, the study in [7] provides a pattern-based analysis of BPEL4WS workflows. Authors provide a comprehensive set of patterns pertaining to all aspects of BPEL4WS specification, whereas in our study the investigation of patterns is limited to the orchestration semantics of DAG-based workflow specifications.

Another pattern based study [12] investigates the usage of workflow patterns to incorporate policy-based fault-tolerant recovery mechanism at run-time.

Several studies [1, 3, 6] deal with run-time workflow adaptation problem through periodically rescheduling the workflow and re-enacting the workflow in correspondence with the new mapping layout. However all these methods necessitate the halting of the whole workflow execution in progress, hence they are highly-intrusive and not scalable for large scale workflows.

VII. Conclusion

In this paper, we propose a generic framework for the decentralization and run-time adaptation of large-scale workflow applications that span multiple sites of resources. Our framework is applicable to any scientific workflow application that is specified in a DAG form. By investigating recurring DAG patterns, we devise corresponding transformation and adaptation patterns to incorporate decentralization and run-time adaptation capabilities to standard workflow execution managers. As our framework does not alter the business logic of a workflow application, it is safe to use and can be evaluated under varying circumstances. We also provide a prototype implementation of our framework on a standard workflow execution manager.

In the future, we would like to investigate the implementation of our framework on other workflow execution managers as well. We are also interested in investigating various business and technical aspects of cloud-bursting on the orchestration and adaptation of workflow applications.

Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant Nos. OISE-0730065 and HRD-0833093.

References

[1] Sakellariou, R. and Zhao, H. 2004. “A low-cost rescheduling policy for efficient mapping of workflows on grid systems.” *Sci. Program.* 12, 4 (Dec. 2004), 253-262.

[2] Deelman, E, et al., "Pegasus: A framework for mapping complex scientific workflows onto distributed systems." s.l. : Scientific Programming, 2005, Issue 3, Vol. 13.

[3] Lee, Kevin, Norman W. Paton, Rizos Sakellariou, Ewa Deelman, Alvaro AA Fernandes, and Gaurang Mehta. "Adaptive workflow processing and execution in pegasus." Concurrency and Computation: Practice and Experience 21, no. 16 (2009): 1965-1981.

[4] Wieczorek, M, Prodan, R and Fahringer, T., "Scheduling of scientific workflows in the ASKALON Grid environment." s.l. : ACM, 2005, Issue 3, Vol. 34.

[5] Condor team, “The directed acyclic graph manager”, www.cs.wisc.edu/condor/dagmang, 2002.

[6] Yu Z, Shi W. “An adaptive rescheduling strategy for grid workflow applications.” *IPDPS*, IEEE Press, 2007; 1–8.

[7] Wohed, Petia, Wil MP van der Aalst, Marlon Dumas, and Arthur HM ter Hofstede. “Pattern based analysis of BPEL4WS”. QUT Technical report, FIT-TR-2002-04, Queensland University of Technology, Brisbane, 2002.

[8] Weihai Yu. 2009. “Decentralized Orchestration of BPEL Processes with Execution Consistency.” In Proceedings of the Joint International Conferences on Advances in Data and Web Management (APWeb/WAIM '09), Qing Li, Ling Feng, Jian Pei, Sean X. Wang, Xiaofang Zhou, and Qiao-Ming Zhu (Eds.). Springer-Verlag, Berlin, Heidelberg, 665-670.

[9] Pantazoglou, Michael, Ioannis Pogkas, and Aphrodite Tsalgatidou. "Decentralized Enactment of BPEL Processes." (2013): 1-1.

[10] Oinn, Tom, Matthew Addis, Justin Ferris, Darren Marvin, Martin Senger, Mark Greenwood, Tim Carver et al. "Taverna: a tool for the composition and enactment of bioinformatics workflows." Bioinformatics 20, no. 17 (2004): 3045-3054.

[11] Berman, Francine, Andrew Chien, Keith Cooper, Jack Dongarra, Ian Foster, Dennis Gannon, Lennart Johnsson et al. "The GrADS project: Software support for high-level grid application development." International Journal of High Performance Computing Applications 15, no. 4 (2001): 327-344.

[12] Kalayci, Selim, Onyeka Ezenwoye, Balaji Viswanathan, Gargi Dasgupta, S. Sadjadi, and Liana Fong. "Design and implementation of a fault tolerant job flow manager using job flow patterns and recovery policies." Service-Oriented Computing–ICSOC 2008 (2008): 54-69.

[13] Kalayci, Selim, Gargi Dasgupta, Liana Fong, Onyeka Ezenwoye, and S. Masoud Sadjadi. "Distributed and Adaptive Execution of Condor DAGMan Workflows." 22nd International Conference on Software Engineering & Knowledge Engineering (SEKE'2010): 587-590.

Framework for digital voting systems

Patricia Dousseau Cabral
Graduate Program in Computer Science
Federal University of Santa Catarina
UFSC
Florianópolis, Brazil
dousseau@inf.ufsc.br

Ricardo Pereira e Silva
Department of Informatics and Statistics
Federal University of Santa Catarina
UFSC
Florianópolis, Brazil
ricardo@inf.ufsc.br

Roberto Silvino da Cunha
Labsoft
Federal University of Santa Catarina
UFSC
Florianópolis, Brazil
rsc@inf.ufsc.br

***Abstract*— The electoral system is vital to the democratic process. Several countries have adopted different mechanisms for the electoral process, from paper ballots to electronic voting machines and online voting. There are several benefits of digital polls conducted over the internet, but it also comes with vulnerabilities and forms of manipulation. With the intention of facilitating the development and validation of online voting systems an object-oriented framework was developed for protocols and online elections, where it is possible to implement and test new protocols without the need to develop an entire system.**

Digital voting, object-oriented framework, voting systems, voting protocols

I. Introduction

The electoral process has changed throughout the years, popularizing the use of voting over the internet. It brings benefits such as a faster and more efficient tally, making it easier to cast the vote, eliminating the need of commuting to the polling station, possibility of process verification and reducing costs. But new threats can emerge, such as ease of coercion of voters and new opportunities for fraud. To reduce these risks, there are several safety requirements that the system must meet [1]:

- Accuracy: ensure that only valid ballots will be counted in the tally and cannot be changed or duplicated.

- Uniqueness: ensuring that only authorized voters participate in the voting, only voting once.

- Privacy: not allowing to link the vote to the voter (anonymity), not allowing knowing the option selected by the voter (non-coercion) and all ballots must be kept secret until the end of the tally (impartiality)

- Verifiability: there are two types of verifiability: individual, that allows the voter to verify that their vote was correctly determined; and universal, which shows that all the votes were counted correctly.

There is an inherent difficulty in meeting all these requirements since some tend to be self-exclusionary, such as the difficulty in proving that the vote of the voter was properly counted while not revealing their voting option. Or difficulty in allowing only authorized voters to cast a ballot without associating the vote with the voter.

Several voting protocols have been proposed in literature trying to satisfy the requirements mentioned above. Besides the difficulty of designing new protocols, there is the difficulty of implementing a complete system that uses it to be able to validate and analyze the proposed protocol. To facilitate this process, we developed an object-oriented framework for systems and digital voting protocols to simplify the deployment and management of online voting.

II. Context and Motivation

Due to the difficulty in satisfying all the requirements that makes an election secure, several authors strive to create protocols for specific situations, ie, that are reliable in certain contexts and that meet only a part of the requirements. An example is the Helios voting system [2], which is suitable for elections where voting should be secret, but where coercion is not a major threat. In this line of work we can cite polls for clubs, software communities and student communities. The system described by Chuan-Kun Wu and Ramesh Sankaranarayana [3] is suitable for coercion free elections, because the system surpasses this threat by allowing the voter to vote several times. Making it harder to force the voter to choose a particular option, since they can change it later. There are several proposed protocols and voting systems, as can be seen in [2] [7][8] [9] [10].Protocols structures are well formed and usually small changes in its logic can compromise system security. It is not an easy task to change a part of the protocol or extend it without considering the impact across its logic. Thus, typically they are not configurable, and neither the systems that use them, since they are dependent. So we have systems that are inflexible and difficult to reuse if you want to change their operating logic.

Given the need to develop a system of digital voting, it was difficult to reuse the systems' parts which were already developed. Through analysis of these systems, we found many commonalities, choosing to adopt a reusable approach from the very beginning of the project development. An object-oriented framework for protocols was considered appropriate, given the fact there is a large number of proposed protocols and the difficulty to validate them, since a new system is required to be built whenever you want to validate a specific change. Nevertheless, there are not many documented proposals. David Lundin [4] proposes a digital voting system based on components that can be interchanged and audited, so that you can add and remove components without impacting other parts of the system. Such a system ends up being difficult to use,

since it is necessary to create a new component every time you want to change something, being necessary to follow all the conventions of the component during its creation.

Stefan Popoveniuc and Poorvi L. Vora [5] propose a framework for voting systems using mixnet and paper ballots. They analyzed four systems whose front-end and back-end can be interchanged. Such a system is restricted to voting using paper ballots and mixnet, additionally in being completely necessary to build the back-end and front-end when you want something different.

The proposed object-oriented framework is more flexible than similar proposals, it allows you to implement specific changes rather than being required to build a component or a front / back end. Moreover, it is modularized and can make changes with little impact on other parts of the framework with a high code reuse, since most of the structure of the voting process is implemented in the framework, and reused every time a new voting system is developed. This is ideal for testing and validating new protocols and systems, since minor codes need to be written.

A. Voting technologies

According to Jörg Helbach and Jörg Schwenk [6] voting systems use different technologies for its implementation, the most used are:

- Homomorphic encryption: allows the sum of the votes and decryption of encrypted result as follows E(x1) + E(x2) = E(x1 + x2) where x1 and x2 are ballots. By way of this technique is not necessary to decipher each vote individually, making it more difficult to associate the voter and their vote, making it easier to proof that all ballots were counted correctly, and all ballots received were counted.
- Mixnet: shuffles the order of arrival of the ballots guaranteeing anonymity of the voters, making it impossible to determine the order of voting.
- Blind signature: allows signing documents without knowing their contents. This is useful in validating ballots so that the system can validate it without knowing the choice of the voter.
- Bulletin Board: works like a mural of information, allowing sensitive data to be safely released. It can be used to publish the election results, because it ensures that only authorized entities publish information, and ensures that the data has not been changed or deleted. [11]
- Asymmetric encryption: allows encryption and signatures of the contents in a secure way. To facilitate the use of asymmetric cryptography, a repository of keys and digital certificates are used, which are both implemented in the framework.
- Symmetric encryption: allows secure encryption of content, using only one key instead of two.
- Hash: generates unique identifiers making it easy to identify objects and analyzing if changes were made.

In the developed framework these technologies were considered as different primitives that will be used in the protocol. You can insert and remove the primitives from the structure of the framework in specific places without major impacts.

B. Vulnerabilities

Although there are several proposed protocols for digital voting that meet all or some of the requirements mentioned, none is used in a large scale election with high criticality [3]. The reason is that there are several difficulties in making a voting system reliable. According Chuan-Kun Wu and Ramesh Sankaranarayana [3] there are several aspects that make digital voting so complex and vulnerable.

- Reliability in software: the more complex the software, the larger the code and the bigger risk of errors. It is difficult to detect them and predict how critical they are.
- Reliability on the Internet: the difficulty of protecting themselves from attacks through the network, some of which may be quite harmful to the system.
- Reliability of database system: the data stored in the database can be compromised or violated.
- Confidentiality of electronic votes: votes can be intercepted or the system may not be trustful, allowing someone to discover the contents of the votes
- Detection of double voting: how the voter can vote over the internet, it is more difficult to detect if they have already cast an earlier ballot. In case there are two risks, cast the same ballots several times or asking the system for several different ballots.
- Vote buying: the voter can sell his vote or be coerced to vote in a particular way.
- Internet terrorist attacks: an online voting system is more vulnerable to terrorist attacks over the internet.

It is necessary that the authors of protocols and digital voting systems adhere to the above challenges, so that the system can be as reliable as possible. With the framework it is possible to reduce the development cycles and tests, and the maturation of the software becomes more reliable.

III. Framework

The object-oriented framework was produced considering the aspects that must be taken into account when envisioning a digital voting system: their vulnerabilities, technologies, potential threats to its integrity and the entire management of an election. Its structure consists of modules that interact with each other. One module is responsible for cryptographic primitives, another for the management of ballots and voting options, authentication, creating protocols, managing users and for managing elections (Figure 3).

A. Functionalities

A digital voting system should provide basic functionalities. Whereas there are in the system administrator,

voter, register agent and auditor profiles, we can list the following roles associated with the use cases of Figure 1 and 2:

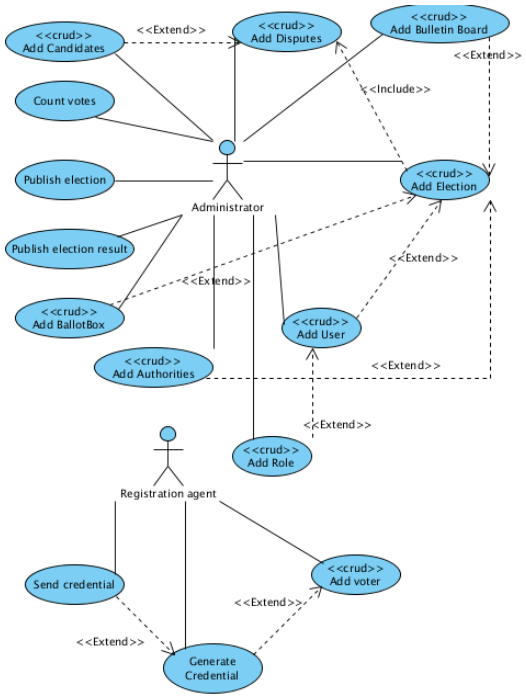


Figure 1. Administrator and register agent use cases

Administrator: The system administrator is responsible for registering elections and all information about it, such as voting date, title, voting options, etc. These steps are different use cases, since it can be performed at different times.

Voter: have the power to obtain a ballot, cast a ballot, verify that their ballot was counted correctly and check the result. Receiving the ballot and casting the ballot are parts of the voting process that are divided into two stages because they necessarily do not happen together. Checking if the ballot was properly audited takes place in the final stage of the voting which depends on the protocol.

Auditor: the system has two types of auditing. The configuration auditing and verification auditing. Every election may or may not enable them, leaving it to the system administrator. If at least one is enabled, it is necessary that at least one auditor is registered in the election. The configuration auditing concerns the data analysis of the election, that is, if all the information registered by the administrator is correct: election title, voting date, voting options, etc. The election can only be made public after the approval of all auditors. Since the auditing result concerns the analysis of the ballot counting and other evidences that the election may issue depending on the protocol, such as evidence that the mixnet shuffle was done correctly. Only after this analysis is that the result can be revealed. If the auditing is rejected by some problem, the election should be canceled because it proves that there was some fault or fraud during the process.

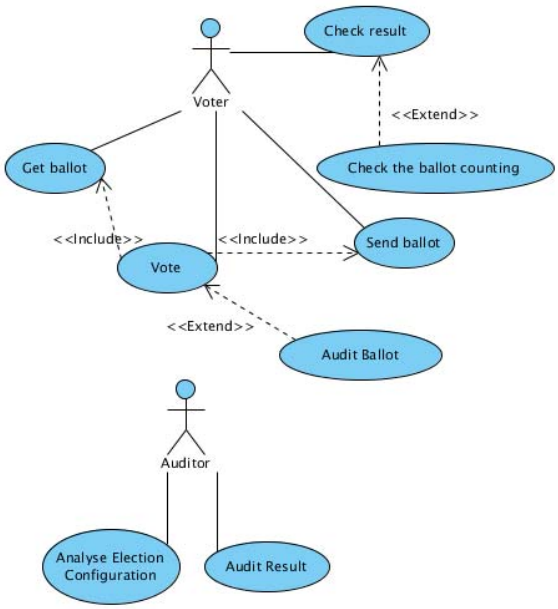


Figure 2. Voter and auditor use cases

Register agent: Responsible for the registration of voters in a particular election. This function can also be assigned to the administrator, eliminating this role.

Besides the specific use cases for each role, they all have the ability to log into the system and list all elections that they are related.

B. Adopted methodologies

The domain analysis to develop the framework started with the identification of similarities between different voting protocols described in the literature. For this, we selected the following protocols [2] [7] [8] [9] [10]. These choices have some characteristics relevant to voting systems and constitute a sample of the relevant field of voting protocols. In the case of Helios [2] it has both individual and universal verification. This allows the voter to verify that his vote was correctly casted and also allows the analysis of the vote count. In the case of Three-Ballot-based protocol [7], it was chosen because it makes use of three ballots for voting and three ballot boxes, and the most common is to use only one. Sensus protocol [8] was chosen because it requires three subsystems, validator, pollster and tallier. The Seas protocol [9] is based on the Sensus protocol, with minor modifications intended to eliminate the possibility of authorities voting in the place of voters who abstained from voting. It was an issue found in the Sensus protocol. Finally, the protocol proposed by Ray [10] makes use of three authorities, a ballot distributor, a certifying authority and a voter compiler.

From the domain analysis, it is concluded that it is possible to split a vote in four steps:

- Initialization: optional step, performed if there is need for some protocol initialization, such as the inclusion of previous blank ballots in the ballot box or mixnet creation servers.
- Obtaining a ballot: a step where the voter gets a ballot, and optionally performs the auditing. It is usually anticipated by the authentication process of voting, with some exceptions, as can be seen in [2], where the voter authenticates just in time to cast the ballot, allowing anyone to get a ballot and perform the auditing.
- Casting a ballot: it is the act of voting. A step after the voter selects their voting option, and then send their ballot to the system. Some protocols require that the voter authenticate themselves in this step, for example, Helios [2].
- Tally: a step started after the end of voting, where all votes are counted and the results are announced. This step can also contain the results of the audit conducted by the auditors of the election.

All these steps may employ some common mechanisms, such as authentication, use of ballots, user profiles, etc.

A common factor was the use of ballots containing the voting options and the definition of roles a user can assume in the system, normally being voters, administrators, register agents, auditors and fiscals. Another key aspect is the role of the authorities. Some systems, such as Helios, use only a controlling authority, which manages the entire voting process: voter authentication, receiving the votes, counting and publishing the results.

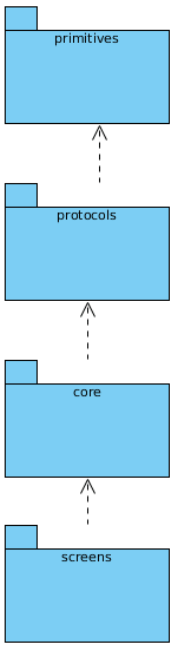


Figure 3. Framework modules

While others, such as [10] distributes the intelligence system between three authorities: one to identify the voter and to issue the ballot, another to verify that the ballot was cast and ensure that each voter submits only one ballot and the last for counting the votes and revealing the results.

C. *Framework structure*

The framework consists of four modules, each responsible for managing an aspect of the voting system. As shown in Figure 2.

1) *Screen*

Contains the screens that will be used by system users and a layer between the core system and screens. This middle layer is responsible for identifying which methods are available for each system profile, and it is through it that the screens will have access to features implemented.

2) *Core*

The system core is the main part of the framework. It manages the operation of the system. It is also responsible for the election, ballot boxes, disputes, voting options, user authentication and the management of user roles and ballots.

3) *Protocols*

Responsible for various implementations of the protocols that will be used by the elections. They are the heart of the election, being responsible, in large part, by the security of the electoral process. The protocol defines the structure of the ballot and how they will be obtained, how the votes will be counted, how the ballots will be casted, and which cryptographic operations the system will perform during the voting, among other things. You can extend the framework by adding new protocols.

4) *Primitives*

The framework contains several primitives already implemented, and others can be easily added to the system. They will help you compose a new protocol, and key points of its implementation. Often primitive structures are quite complex and its functioning is quite critical. If they do not work properly, the protocol will not work. An example of this is the mixnet, which requires several distributed servers working in a coordinated way. You can extend the framework by adding new primitives.

D. *Behavior*

There are key stages in the electoral process, such as the period that is allowed to vote and the voting stage where the votes will be counted. A poll consists of 13 states, which can be better visualized in Figure 4.

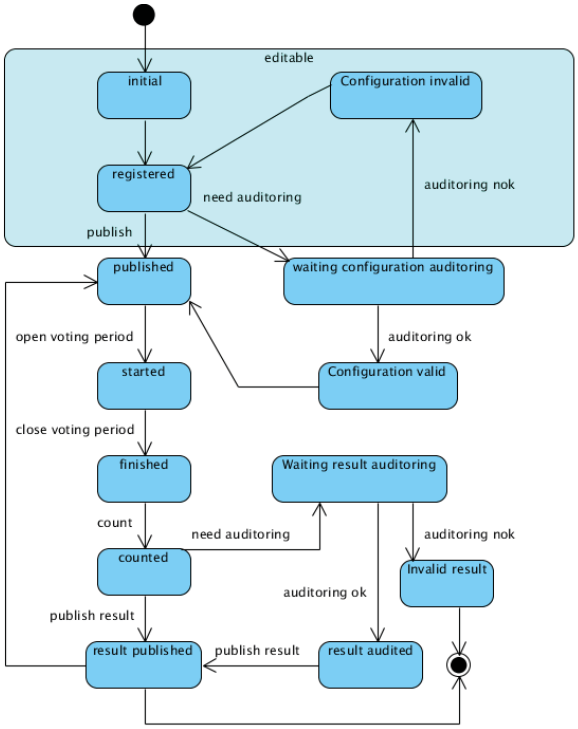


Figure 4. Election state machine

The registered state relates to an election that was entered into the system, but has not been validated, for not having passed the auditing or because its record was not finished. The published state concerns the election that has already been completely registered in the system and can no longer have their data changed. Therefore, it is waiting for the voting to open. The election that is started is one that is within the voting period, receiving the votes of registered voters. When the election is finished, it relates to the election whose voting period has been exceeded, and now awaits the determination of the results. The counted state relates to the election that had already counted the votes, but the result has not been disclosed. The election that has its result published means that the result was publicly disclosed.

There are two possible auditing states. The first is after the election registration, which is necessary that all auditors approve its registry so that it can be finally published. While awaiting the election auditing, their data cannot be changed. If at least one auditor rejects the settings, the election should be edited. This cycle continues until all auditors' approval. Another possible auditing is the auditing of the election results. This one is dependent upon the protocol, which in some cases gives evidence that some operations have been properly made, for example, the shuffle of ballots when a mixnet is used. In this type of auditing, if one auditor rejects, the election should be canceled because it means that there was some kind of fraud or malfunction which endangers the outcome of the election. If all auditors approve, your result may be disclosed.

Besides the logic of the electoral process, there are other mechanisms in the system that work to make the framework flexible and reliable. These mechanisms are within the core package, as well as the election.

1) Authentication

Authentication can be done in different ways and at different times of the voting process. For example, the voter can authenticate himself by using a login and a password, via a valid certificate, a token or by a unique identifier which has been informed to the user. This authentication can also take place at different stages: when the user enters the system, the time they request a ballot or the time they cast a ballot. In the case of the framework it was implemented an authentication by login and password, but is flexible to use other authentication mechanisms.

2) User management

The election process can contain different types of users, each with different requirements and obligations. The most common are voters, administrators, auditors, fiscals and register agents who are responsible for registering voters. More profiles and its responsibilities can also be added easily.

3) Ballot management

The structure of the ballot is not always trivial, depending on how the disputes and its options are organized. To facilitate its creation, a ballot is created and others are copied from the original ballot. If any additional voter data has to be added, like a ballot identifier, it is inserted after the copy.

4) Disputes and voting options

The disputes relate to matters which are voted in the election, and which voting options are available for each subject. It was determined that there are two types of disputes, candidates and plebiscite. The framework allows the creation of new types of disputes and voting options without much impact on the rest of the system. Disputes like plebiscites are open questions with simple answers. Another kind of dispute are candidate ones, that requires a complex structure normally required for candidates. For example, the relationship between candidate, party, coalition and slate.

5) Ballot box

The ballot box is responsible for maintaining the ballots so only those authorized can retrieve them in the same way as physical ballots.

E. Utilization

Three protocols have been implemented using the framework. A simplistic protocol, a protocol that makes use of blind signatures, and a protocol that makes use of mixnet [1]. All three allow you to create plebiscite and candidate disputes, they all have user management and make use of the ballot box and the state machine shown in Figure 4. If the developer wants to change some aspect of the system, they should override the responsible class and implement it following the structure of that class, making the framework flexible enough. For the implementation of a new protocol using primitives that are not configured in the system, it is necessary the creation of this new primitive and its inclusion in the primitives' package. It is also necessary to extend the protocol class and implement it according to its structure, as can be seen in Figure 5. Sometimes you may also need implementing other classes, such as those designed to do specific actions in the voter

machine. Below is an example of protocols developed and what classes were necessary to create or override.

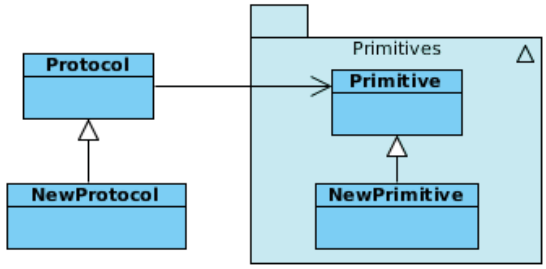


Figure 5. Creation and implementation of a new protocol and a new primitive

1) Simplistic protocol

For the implementation of the simplistic protocol it was necessary to override only the Protocol class and two of its methods. From 306 classes in the system, only one was overridden and 2 of its 25 methods were overridden and no new methods were created. Practically all the necessary infrastructure to implement the simplistic protocol is already developed. There is already all the basic structure of a digital voting system, including the administration, voting and presentation results screens.

2) Protocol with blind signature

To implement the protocol with blind signatures, it was necessary to override the Protocol class and two of its methods, besides being necessary to create three more. It was also necessary to create an applet to blind the voter ballot before sending it to the system to sign. In this case it was necessary to create a different interface with the voter, which was able to perform certain cryptographic operations on the users machine before sending it to the authority in order to increase confidence in the whole process.

The reuse in this case remains quite high. From 306 classes in the system, only one was overridden, with 4 of its 25 methods overridden and three new ones were created. Besides, it was necessary to create a new class with three new methods.

3) Protocol with mixnet

To implement the protocol with mixnet it was needed to override the protocol class and two of its methods. There was no need to create new classes or methods.

Through the implementation of the three protocols, it was possible to verify the feasibility of the framework to support the development of electoral systems, as well as to permit a considerable reuse in the development. All protocols had a reuse above 90%.

IV. Conclusion

Given the complexity to develop and validate new protocols and digital voting systems, the framework was proven to be able to facilitate the development and testing of new systems and protocols. It allows the developer to focus on the most important aspects of the development, since the basic structure of a voting system is implemented in the framework. It is advantageous with respect to similar proposals, which are less flexible and less comprehensive and also enabling less reuse.

References

[1] R. Samarone, “Protocolos Criptográficos para Votação Digital,” Unpublished master's thesis for mater's degree, Universidade Federal de Santa Catarina, Florianópolis, Brazil, 2002

[2] B. Adida, “Helios: web-based open-audit voting,” Proceedings of the 17th conference on Security symposium (SS'08). USENIX Association, Berkeley, CA, USA, 2008, pp. 335-348.

[3] C. K. Wu, R. Sankaranarayana, "Internet voting: concerns and solutions," Cyber Worlds, 2002. Proceedings. First International Symposium on Cyber Worlds, 2002, pp. 261 – 266

[4] D. Lundin, “Component Based Electronic Voting Systems”, In *Towards Trustworthy Elections*, David Chaum, Markus Jakobsson, Ronald L. Rivest, Peter A. Ryan, and Josh Benaloh (Eds.). Springer-Verlag, Berlin, Heidelberg pp. 260-273. 2010

[5] S. Popoveniuc, P. Vora, "A framework for secure electronic voting", In IAVoSS Workshop On Trustworthy Elections, 2008

[6] J. Helbach, J. Schwenk "Secure internet voting with code sheets," In Proceedings of the 1st international conference on E-voting and identity (VOTE-ID'07), Ammar Alkassar and Melanie Volkamer (Eds.). Springer-Verlag, Berlin, Heidelberg, 2007, pp. 166-177.+

[7] A.O. Santin, R.G. Costa, C.A Maziero, "A Three-Ballot-Based Secure Electronic Voting System," Security & Privacy, IEEE , vol.6, no.3, May-June 2008, pp.14-21,

[8] L.F. Cranor, R.K. Cytron, "Sensus: a security-conscious electronic polling system for the Internet," System Sciences, 1997, Proceedings of the Thirtieth Hawaii International Conference on System Sciences - HICSS '97 , 7-10 Jan 1997, pp.561-570 vol.3

[9] F. Baiardi, A. Falleni, R. Granchi, F. Martinelli, M. Petrocchi, A. Vaccarelli, "SEAS: A Secure E-Voting Applet System," Lecture Notes in Computer Science pp. 318-329, January 2004

[10] I. Ray, I. Ray, N. Narasimhamurthi, "An Anonymous Electronic Voting Protocol for Voting Over The Internet," In Proceedings of the Third International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems (WECWIS '01) (WECWIS '01). IEEE Computer Society, Washington, DC, USA, pp. 188-. 2001

[11] J. Heather, D. Lundin, The Append-Only Web Bulletin Board. Guildford, Surrey, UK, University of Surrey, 2008.

How do You Execute Reuse Tasks Among Tools?

A RAS Based Approach to Assist Software Asset Tailoring

Fábio P. Basso, Cláudia M. L. Werner, Raquel M. Pillat, Toacy C. Oliveira
COPPE – PESC – Federal University of Rio de Janeiro (UFRJ)
Rio de Janeiro, Brazil
{fabiopbasso, werner, rmpillat, toacy}@cos.ufrj.br

***Abstract*—Software reuse practices and tools have been proposed over the last three decades. From the reuser's point of view, it is necessary to provide facilitators in order to execute reuse tasks among tools. We propose the use of Reuse Assistants (RA) as a representation for tasks that tailors software assets to be used in chained executions of reuse tools. Since many tools do not interchange their input and output (IO) with others, reuse processes are limited to executing reuse tasks from a single tool. Moreover, reusers are required to manually adapt IO parameters between tools aiming to execute a more complete reuse process. However, such adaptations are subjected to inconsistencies and errors. In this sense, by specifying RAs through a common representation one would be able to execute a reuse process by chaining tools in an assisted way. This paper presents a work in progress to support software asset tailoring through RA.**

Keywords-component; Reuse technique; Software reuse; Reuse process; Tool chain; Reusable Asset Specification - RAS

I. INTRODUCTION

Software reuse has been used over the last three decades as a practice that leads software engineers to leverage on past experiences while creating new software systems [20]. As a result, several techniques and their corresponding reuse tools were developed, such as Product Line Architectures (PLA) [12][14], Component Based Development (CBD) [28], source-code generation based on Model Driven Development (MDD) [4][22], and Object-Oriented Framework Instantiation (OOFI) [23]. In this scenario, some tools are used together to develop a software project. However, most of the time they have incompatible input and output (IO) parameters (software assets), making their execution in a chain of reuse tasks complex. Thus, reusers are challenged to manually chain tools by adapting these assets in an error-prone task. Accordingly, this paper presents a reuse scenario composed by existing solutions (a toolset) that promotes software reuse with different reuse techniques. Therefore, we present a proposal to represent assistants that tailors software assets to be interoperated among reuse tools.

Software engineering practitioners commonly use more than one tool and technique to support activities that involve reuse. Such activities are executed using some software assets that are refined from higher abstraction levels to specific needs [3]. In this context, Aho et al. claim that, even in the case where a single technique is used to support such activities, interchanging information between reuse tools is a hard and error-prone task [1]. Authors exemplify a scenario where many tasks are required to support the development of web services functionalities based on MDD technique such as to design a UML model, to generate source-code for Java and SQL schemas, and to finally integrate everything and deliver a testable prototype. Each activity is supported by a specific tool and is manually synchronized by technical stakeholders such as developers and project leaders.

In this context, this work proposes a common representation to specify reuse tasks among tools as Reuse Assistants (RAs) [7]. RAs correspond to an EMF based model [9] that represents in high-level of abstraction the executable task required to adapt software assets to be used among tools. Examples of assets are programming APIs, model transformers or software domain designs. In this direction, we provided in [7] a common representation for RAs to allow tools chain as an extension of Reusable Asset Specification (RAS) [27]. RAS supports some software activities with descriptions and guidance tasks, helping reusers in performing reuse independently from a tool support.

In this sense, this paper presents a proposal to specify and adapt RAs to different execution environments. Experiences on practical application regarding these adaptations are discussed as well as ongoing work. Preliminary results are promising and suggest that the usage of reuse techniques among tools can be facilitated though the proposed approach.

Next sections are organized as follows. Section II provides an introduction about reuse assistants with motivation based in industry experiences. This section also presents an example of tools chain and points to the lack of current approaches to support reuse processes among tools. Section III presents a proposed architecture and reports experiences as means to validate the proposal. Section IV discusses the related work, and Section V presents some concluding remarks and limitations of our work.

II. MOTIVATION

We have large experience in supporting reuse techniques [4][12][28][13][33]. More specifically, we tailor tools and techniques to support model-driven based processes for different development scenarios. These processes require dealing with a set of reusable assets manipulated and executed with some tools. In this context, it is important that each activity be correctly executed and to ensure that assets used as input for further tasks are in conformance. This implies in a situation where tools must interoperate these assets with valid IO parameters. Besides, each activity requires variant guidelines to help stakeholders in executing transformations.

These are dependent on the adopted reuse techniques involved in each activity and the tools used towards modeling and source-code generation practices.

A. Illustrative Reuse Scenario

In this sense, the following activities can compose a reuse scenario that requires asset adaptations among tools:

1. Functional requirements (**asset 0**) are specified using a word processing template. Alternatively it could be specified with SPEMArti [29], a Domain Specific Language (DSL) to describe use cases.
2. A UML model (**asset 1**) is designed with Enterprise Architect (EA). Other alternatives to EA are also used and each one exports and imports models using different versions of XMI.
3. A second tool named MockupToME [6] is used to generate graphic user interfaces (**asset 2**) taking as input **asset 1**. Alternatively, **asset 1** can be imported by applying a reverse engineering using WCT module for Java reverse engineering from code to model [32] or be generated by a PLA approach such as Odyssey [12] or RDL-FI [13]. Besides, in case of **asset 0** being specified with the DSL then the generated user interfaces must be traced to **asset 0**.
4. A database schema is modeled using an Object-Relational Mapping (ORM) UML Profile [19] (**asset 3**). Alternatively, one can use WCT wizards to help in the application of some profiles. In any case, it is necessary to use guidelines because this task requires a high know-how to execute.
5. A model transformation (**asset 4**) is executed with FOMDA [4] to generate: a) SQL scripts (**asset 5**), used to create database schemas; b) entity classes (**asset 6**) annotated with ORM mapped for Hibernate [31]. Other transformation tools can be used as alternative.
6. SQL scripts are imported into an SGBD, which creates a database. Alternatively, one can leverage the database creation to Hibernate. In this case SQL scripts generation is optional as well as this step.
7. Other model transformers (**assets N**) are executed to generate each web application layer such as view, controller, validation and data access objects. Alternatively, some applications support Java Swing. In this case, the generation of graphical user interfaces for Java Swing and also a remote layer are required.
8. Finally, the generated source-code is changed by developers. Here, many assets are used such as Java APIs, existing source-code, documentation, etc.

B. Difficulties to Interoperate Reusable Assets Among Tools

Based on such activities, we can highlight some difficulties to chain assistants among tools and techniques: a) they support reuse activities that require different asset as inputs [15][11][30][25]. Some input must be changed after generated; b) tools are developed to interchange files and not to interoperate reuse tasks [24]. This was the reason why in [33] Wizards were developed to help stakeholder in a UML Profile application: UML modeling tools do not provide a simple solution for specifying it; c) incompatibility between input and output (IO) [1], which required manual adaptations from XMIs generated by UML tools to be executed by model transformers; d) tools do not allow to modify existing reuse tasks because most of the time they are black-box [23]. In this case, wizards must be developed outside reuse tools.

Due to the lack of appropriate guidance, important reuse techniques may be missed. This situation occurred when a company withdrew our proposed MDD techniques and tools to program everything by hand [5][6]. This was motivated by the following reasons: 1) It was required a long period of time to a modeler to learn a UML tool, when executing the second task, and by dealing with valid XMI files as input for transformations in the fifth task. Here we learned a lesson, that even in equal versions of XMI, tools export models differently. As consequence, model transformers did not work properly; 2) The learning curve was even bigger to execute the fourth task, because the application of UML Profiles requires experienced modelers. Here we learned that facilities such as guidelines help inexperienced modelers. However, they are costly to develop as wizards; 3) We learned about how to execute further activities after executing each model transformation (fifth and seventh tasks), such as to correct source-code pieces. With no guidance to pinpoint what should be changed in each generated source-code for each application layer, developers were confused about the required procedures to apply glue-code.

To fix these problems, we provided wizards as a solution for guidance and execution support [6]. This facilitated the proposed activities adoption by non-experienced stakeholders. Unfortunately, this scenario is not well explored by software engineers, leveraging reuse assistants as options to support reuse activities. Accordingly, in absence of a common representation for reuse assistants, reuse tools are strictly adapted by people who have strong know-how about the internal structure of the software, because a wizard is typically a black-box program. In this sense, it is not possible to know the activities and the assets generated by each assistant. As consequence, inconsistencies are faced in daily practice regarding MDD.

C. Desirable Scenario

Currently, MDD practitioners must manually adapt reusable assets interoperated among tools in an error-prone process [1]. The reason is that a non-common representation for reuse activities exists to interoperate these IO parameters. They change IO file formats and apply corrections to fit each software asset with activity parameters. This requires abilities, usually driven to highly technical stakeholders (i.e., software developers). However, to change this scenario, we claim that reuse assistants specification, which tailors software assets, should be specified by non-technical stakeholders [23].

This would imply in a common representation for reuse tasks, where each reuse tool interoperates simple information such as: 1) IO parameter with some constraints; 2) descriptions and guidance of assets; 3) reuse activities execution support. Besides minimizing incompatibility errors, it would also bring other benefits, such as to automate a software process. Moreover, these assistants could be chained based on CBD techniques and be interoperated using existing task execution

tools. Also, a common representation would allow a deployment of reuse processes into existing development environments such as Eclipse. However, tools are not prepared to support this kind of approach.

III. Proposed Approach and Experiences

In direction to facilitate the execution of reuse tasks among tools, we propose the architecture shown in Figure 1 that support reuse with three main steps: 1) asset specification; 2) asset acquisition; 3) and asset usage and execution among tools. The first step is to specify assets in a common representation and store it into a Reusable Asset Specification (RAS) based repository, discussed in Section III A and B. In the second step, the reuser acquires these assets by searching the repository as depicted in Section III C. Finally, a program called RAS Client Deployer configures user workspaces using assistants as input for transformations, as detailed in Section III D.

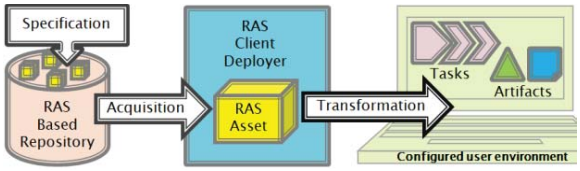


Figure 1: Proposed Architecture to Support Reuse Assistants

Existing reuse tools must be plugged into workspaces without recurring to black-box configurations such as those required by IDE plug-ins [10][21]. For this reason, we have decided to define assistant deployment information as a RAS model [7], since it allows detailing how artifacts must be used in a reuse context. Accordingly, we propose that assistants must contain a configuration about reuse practices and about the deployment of their owned artifacts. Besides, it also serves as a protocol between an asset repository and a user workspace. Therefore, any tool that supports our extended RAS metamodel can use a reuse assistant to configure a user workspace.

A. Reuse Assistant (RA) Model Specification

Figure 2 illustrates the activities used in the "Specification" step towards designing reusable software elements. Thus, it is important to know the differences between these elements. Accordingly, following definitions are important.

(1) **Artifacts** are elements produced and consumed along software process lifecycle. Examples are application models, APIs and libraries, source-code, documentation, reuse tasks as tutorials and how-to, etc.

(2) **Assets or Reusable Assets** are elements that document one or more artifacts. They provide guidelines that describe reuse activities required by artifacts to be adapted and used. These elements are described with standard RAS metaclasses and are packaged in a file composed by a single or a set of artifacts, as illustrated in Figure 1 by the "RAS Asset" package. However, despite RAS provides a rich set of data to detail reuse activities, currently this standard has no support to specify details such as execution and deploy support. Therefore, extensions for reuse assistants are required.

(3) A **Reuse Assistant (RA)** is a model representation about execution and deploy of reuse activities that allows adapting one or more reusable assets to support interoperation among different reuse tools. This means that it allows: a) to organize a set of tasks that can be performed by (re)users, with the help of a provided information through *RAS assets*; b) and/or it can be executed with task and process execution languages available, for example, in Eclipse IDE workspaces; c) and/or it configures reuse tools to support the asset development and refinement. This means that those assets discussed in Section II.A must have a common representation (as a RA) to be adapted and interoperated among those reuse tools.

(4) **Reuse tools** are any executable program/application that aims to facilitate the reuse of software artifacts. These programs can be chained in software production tasks, which can be represented in higher abstraction levels as reuse assistants.

A reuse assistant is designed to facilitate reuse of software artifacts. It is used to describe one or more reusable assets. Information about RA must guarantee that reuse activities (manual or executable), and artifacts are installed in the right place, configuring the supporting tools/plug-ins contained in a workspace. Thus, we assume that this kind of task is executed in the (re)user local machine with a focus on defining a few artifact specializations, such as the examples depicted in [23], or generating a source-code fragment, as exemplified in [4].

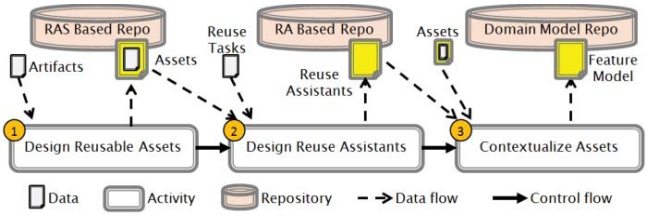


Figure 2. Activities in Support for Asset Specification

B. RAS Based Repositories

Activities shown in Figure 2 are executed by an asset producer. Thus, first two are used to specify artifacts, assets and assistants in a RAS compliant metamodel presented in [7].

In the motivating example, a set of assets are used among tools. To better support artifact reuse with RAs, asset repositories, also known as reuse repository [28], are required. A repository is recommended to place reusable software assets [16][30]. However, existing asset based repositories are not able to retrieve assets based on a reuse process context, a lack that implies in inconformity between asset dependencies. In Section II.A, an example of reuse process pointed that many assets can be composed in alternative ways, dependencies between assets must consider a tailored reuse process to bring assets and respective dependencies as exemplified by those three repositories shown in Figure 2. Thus, the main difference from existing repositories is that the reuser searches for assets and retrieves artifacts plus all RAs related to the selected artifacts considering a tailored process.

The last activity shown in Figure 2 is used to define the context for assets and RAs intending to tailor these elements based on communality and variant designed in a Feature Model [18]. In this sense, all this information must be stored into

specialized RAS based repositories that execute operation to retrieve data based on a selected user context. In this sense, we are extending an asset repository [28] to support contexts related to artifacts.

C. Reuse Assistant Contextualization

This section depicts the proposal for the "Acquisition" step shown in Figure 1. Activities shown in Figure 3 are executed by asset users and also by a model repository in response to user actions. In the first activity, a user searches the repositories for an artifact and receives a context as response in activity 3. As response, the repository must suggest reuse assistants that best fit to support a user's context (activity 2). After he/she searches for reusable artifacts, the repository would also suggest assets for acquisition/download packed in a *RAS Asset File*. Thus, to reach this goal, it is required to capture the user's context as illustrated by activity 3.

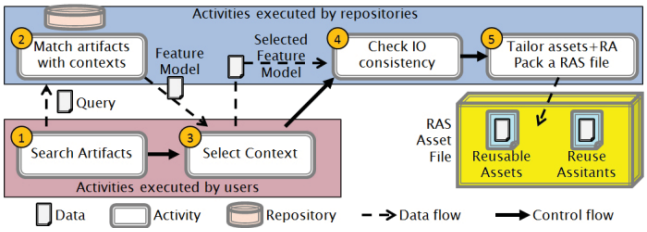


Figure 3. Retrieving Assets and RA from Repositories

A context can be manually provided by (re)users or be automatically captured, because the set of acquired assets can provide information about the user's intentions. Related works propose solutions to contextualize a user environment, such as [15] and [12]. These solutions can also be used in this research to support a more detailed asset contextualization.

After activity 3 is executed, in activity 4 the repository asserts that the selected context allows to return a set of reuse assistants that match the user context. Finally it tailors assets and packs a RAS file composed by reusable assets and reuse assistants (activity 5). This is a detail that deserves deep discussion because it validates the matching between the selected context and the packaged reusable assets.

Practical Application: Our proposal uses the Feature Model to contextualize reusable assets (see Figure 2 activity 3) as done in other works to solve other problems regarding reuse techniques [4][12]. It is the case for reuse tasks 5 and 7 discussed in Section II.A, where transformation processes/chains has been tailored successfully for a set of MDD tool configurations. Thus, these relationships between context variables can be used to support tasks that only fit to a selected feature model. In other words, associating features with assets allows retrieving assets and all other RAs (e.g., model transformers and wizards) related to those assets in the MDD technique example. This is illustrated by the *RAS Asset File* elements shown in Figure 3.

D. Reuse Assistant Executions with a Reuse Process

RAS documentation also suggests that reuse activities should be executed and assisted by programs. However, RAS does not support details to specify tasks execution. To overcome this deficiency, RAS suggests extending the metamodel to support details about an asset. Accordingly, our proposed extensions [7] allow linking executable assets with reuse process languages such as RDL (Reuse Description Language) [23] and also a transformation from a RAS model to task execution languages. This is illustrated in Figure 4 as a solution to support the "Transformation" step (see Figure 1).

A reuse assistant does not execute reuse tasks. Differently, it is a specification that is transformed into configuration files that allow execution support through workspaces plugins. Accordingly, the first activity shown in Figure 4 illustrates a transformation that is automatically performed from reusable assets and reuse assistants (retrieved from tasks shown in Figure 3) to configure workspace, guidance and executions.

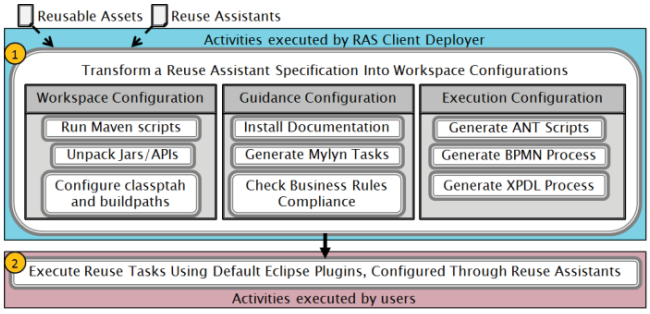


Figure 4. Configuring User Environment Throgh Model Transformations

Link Between RAS Model to Reuse Processes Specified with RDL: With our RAS extension it is possible to specify RAs with a common representation as ensured in the proof of concept [17] by generating RDL scripts. Currently, reuse processes specified with RDL do not allow executing assistants among tools without a reconfiguration/adaptation. In this sense, we will extend RDL to support activities that are linked to RAS elements, allowing the execution of RAs among tools.

Transformation from a RAS Model to Task Execution Languages: The translation from a RAS model to target task execution languages occurs by means of model transformations [22] in activity 1 shown in Figure 4. Such target languages are available in production environments such as Eclipse [10] in support to ANT scripts. Accordingly, proposals to support model-to-model and model-to-text transformations can be used to transform a RAS model into ANT scripts [4][11]. Therefore, our proposal is not to execute RAs, but to generate scripts available in workspaces that support such execution.

E. Ensuring Consistency in Tool Chain

Another need to facilitate the execution of the scenario discussed in Section II.A is related to tool chaining [1] to orchestrate RAs. Accordingly, a tool inclusion in a chain is also dependent on the assistant's IO parameters that must be interchanged between the corresponding supporting tools. This is subjected to errors, since the absence of a common representation does not allow programs to verify consistencies on bindings between tasks interoperated assets (parameter matching) [4]. Thus, a common representation presented in [7] can be used to fulfill activity 4 shown in Figure 3, by ensuring that tasks interoperate consistent reusable assets (parameters). Moreover, this requires the usage of rules that can be specified

using a Business Rule Management System (BRMS) [26] to check constraints inside the proposed asset repository.

Practical Application: An important work was recently presented that discussed about dynamic reconfiguration of features models using BRMS [26]. Thus, next step will focus in specifying rules to constraint RA compositions regarding a context defined by a feature model.

F. Final Considerations

We developed an Eclipse plug-in to deploy the RAS model into an Eclipse workspace as a solution to activity 1 shown in Figure 4. It enables the configuration of executable tasks as ANT tasks and Eclipse Mylyn tasks (to support documentation). Besides, by using the deployer plug-in, we successfully executed the first proof of concept [17], leading to conclude that our extension allows representing information to interoperate reuse tasks among tools [7].

IV. RELATED WORK

In this section, we describe existing work on RAs and tool support that promotes execution guidance and asset specification that can help to achieve an automated chain of reuse activities. This work is divided in four main contribution areas: a) asset specification; b) asset repositories; c) asset contextualization; and d) asset execution.

RAS is the OMG proposal to describe reuse assets. Accordingly, some related work focuses on asset documentation and classification to support a better cataloging, search and retrieval from a component repository [15][11] [16][25]. However, we have not found in literature evidence of work that extends RAS to support the specification of reuse task execution. Given that RAs are mostly like executable activities, this work proposed a RAS extension to grant a successful task deployment and consistency for program execution. Thus, our proposal is to formalize assets with deployment and execution support information as a complement to related work.

RAS related work exemplifies scenarios where assets are only documented. Therefore, it requires less formal definition when compared to what is needed to express executable activities. Nevertheless, some RAS proposals are suggestive to describe reuse tasks: 1) Hadji et al. [15] propose a RAS extension to improve user interactions with component repositories, suggesting the acquisition of artifacts according to the user's context. Our proposal is similar because the developer must acquire a RAS package according to the artifact context defined by a Feature Model [18]. In this sense, despite our work not being focused on contextualized reusable assets as it is required by the exemplified Figure 1, from steps 4 to 6, some PLA related work such as Atkinson et al. [3], Oliveira et al. [13], Gomaa et al. [14], Czarnecki et al. [8] and Antkiewicz et al. [2] and Odyssey [12] are important to contextualize assets using Feature Model; 2) Elgedawy et al. [11] propose a RAS extension to support a better description about SOA reusable components. With more formal specification, SOA components can be acquired and used in the developer's production environment. However, authors do not present a solution to deploy these assets; 3) HongMin et al. [16] propose a component repository to store, search and retrieve RAS assets with extensions to support better cataloging; 4) Accordingly, Park et al. [25] propose RAS extensions to support cataloging and documentation of reusable assets. All RAS extension proposals were added into the RAS metamodel designed with EMF and complemented our work.

Another class of related work proposes a non RAS compliant approach used to specify and to execute assistants. This means the usage of DSLs to describe reuse processes [23]. Ortigosa and Campo [24] claim that Software Agents are important to describe an execution flow in regards to an assistant for OOFI. This is a reuse technique where reuse assistants are programmed to generate source-code that specializes classes and operations of an O.O. framework. Accordingly, Oliveira et al. [23] propose a language named RDL to orchestrate reuse activities and a tool named ReuseTool to support execution. On the other hand, we are proposing that reuse tasks be specified using a RAS extension in an EMF based solution. Thus, by using model transformations, one can transform a RAS input model, designed with a generated EMF-based Eclipse plug-in, into a RDL program or ANT scripts.

Reuse guidance tools are related to IDEs. In environments such as Eclipse [10] and Netbeans [21], wizards are represented as a set of forms codified in Java that guides a reuser through a predefined sequence of data-collection tasks. Ortigosa and Campo [24] and Oliveira et al. [23] claim that a simpler solution to describe wizards is preferable, suggesting that wizard tasks be specified with reuse process specification languages. These proposals are interesting to describe and to execute a reuse process task. However, as they are focused in task sequences execution, they fail in detailing reuse activities that provide important instruction information along the execution of reuse assistants. On the other hand, approaches based on RAS provide extensive information about guidance instruction, but lack support to execution. Therefore, our proposal is to use the best of both worlds and bring the link between reuse process execution and guidance instruction provided by RAS.

V. CONCLUDING REMARKS AND LIMITATIONS

Reusable assets such as reuse practices documentation, reuse assistants and software assets exist in many reuse techniques but they lack integration capabilities. Our proposal is to aggregate such assets as interoperable reuse assistants. To achieve this goal we have discussed about a set of adaptations required to a tool set. Accordingly, we pointed to practical experiences, some in industry, which provide validity of this proposal. More important, recently, a study [17] ensured that a RAS extension [7] can be used to interoperate some reuse assistants.

Although this study suggests some success towards our proposal; some ongoing work is still being conducted: to support executable tasks as a RAS model it is necessary to check model element constraints. Thus, in activity 4 shown in Figure 3 of our proposed architecture, to ensure asset's model correctness, we are planning to support constraints in a RAS model by using a BRMS.

ACKNOWLEDGMENT

This work was partially supported by the Brazilian agencies CAPES and CNPq.

REFERENCES

[1] Aho, P.; Mäki, M.; Pakkala, D.; Ovaska, E. *MDA-Based Tool Chain for Web Services Development*. In Proceedings of the 4th Workshop on Emerging Web Services Technology. 2009. pp 11-18.

[2] Antkiewicz, M., Czarnecki, K., Stephan, M., 2009. *Engineering framework-specific modeling languages*. IEEE Trans. Software Eng. 35 (6). pp 795–824.

[3] Atkinson, C.; Bostan, P.; Fink, F. *Reuse-Oriented Deployment of Software Components: Congregation in Service-Oriented Development*. In Fourth International Conference on Software Engineering Advances, 2009. pp 65-72.

[4] Basso, F.P.; Oliveira, T.C.; Becker, L.B. *Using the FOMDA Approach to Support Object-Oriented Real-Time Systems Development*; In Proc. of Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing. Gyeongju, Korea. 2006. pp 374-381.

[5] Basso F. P.; Basso R. M. P; Oliveira T. C. *An Experience Report in Agile Development with MDA (in protuguese: Um Relato de Experiência no Desenvolvimento Ágil de Software com a MDA)*. In First Brazilian Workshop on Model Driven Development (I BW-MDD), 2010, Salvador – Bahia, Brazil. In Portuguese.

[6] Basso F. P; Basso R. M. P; Oliveira T. C. *Towards a Web Modeling Environment for a Model Driven Engineering Approach*. In Third Brazilian Workshop on Model Driven Development (III BW-MDD), 2012, Natal – Rio Grande do Norte, Brazil.

[7] Basso F. P; Werner, C. M. L; Basso R. M. P; Oliveira T. C. *A Common Representation for Executable Reuse Tasks*. In ICSR 2013 - 13th international Conference on software Reuse, 2013, Pisa – Italy. pp. 283–288.

[8] Czarnecki, K., 2004. *Overview of generative software development*. In: Proceedings of Unconventional Programming Paradigms (UPP) 2004, 15–17 September, Mont Saint-Michel, France, Revised Papers, volume 3566 of Lecture Notes in Computer Science, Springer-Verlag. pp 313–328.

[9] *Eclipse EMF*. Av. at < http://www.eclipse.org/modeling/emf/>. At 15/12/2012

[10] *Eclipse IDE*. Av. at <http://www.eclipse.org>. At 15/12/2012

[11] Elgedawy I. *Reusable SOA Assets Identication Using E-Business Patterns.* In World Conference on Services - II, 2009.

[12] Fernandes, P.; Werner, C.; Murta, L. *Feature Modeling for Context-Aware Software Product Lines*. In International Conference on Software Engineering and Knowledge Engineering, San Francisco, USA, july 2008. pp 758-763.

[13] Filho, M. I.; Oliveira, T. C.; Lucena, C. J. P. *A framework instantiation approach based on the Features Model*. In Jornal of Systems and Software 73 (2004). pp 333–349.

[14] Gomaa, H., July 2004. *Designing Software Product Lines with UML*. Software: Practice Experience 36 (13), 1443–1465 (11, 2006).

[15] Hadji, B. H.; Kim, S.; Choi, H. *A Representation Model for Reusable Assets to Support User Context.* In IEEE International Symposium SOSE ’08 - Service-Oriented System Engineering, 2008. pp.91-96, 18-19 Dec. 2008

[16] HongMin, R.; Zhi-ying, Y.; JingZhou, Z. *Design and Implementation of RAS-Based Open Source Software Repository*. Sixth International Conference on Fuzzy Systems and Knowledge Discovery, 14-16 Aug. 2009. pp: 219-223

[17] *Interchangeable Reuse Assistants - White Paper 01*. Available at <prisma.cos.ufrj.br/wct/irawp01.pdf> At 06/02/2013.

[18] Kang, C. K.; Cohen, S. G.; Hess, J. A.; Novak, W. E. ; Peterson A. S. *Feature-Oriented Domain Analysis (FODA) Feasibility Study (CMU/SEI-90-TR-21)*. Software Engineering Institute, Carnegie Mellon University. November, 1993.

[19] Kim, Y.; Stohr, E.A. *Software Reuse: Issues and Research Directions*. Published in: Center for Digital Economy Research; Stem School of Business; Working Paper Series. IS-9 1- 15. 1991. Available at <http://archive.nyu.edu/bitstream/2451/14370/1/IS-91-15.pdf>

[20] Krueger C. W. *Software Reuse*. In ACM Computing Surveys (CSUR) Volume 24 Issue 2, June 1992. pp 131 – 183.

[21] Netbeans IDE. Available at <http://netbeans.org>. At 15/12/2012

[22] Object Management Group MDA Specifications. October 2004. Available at <http://www.omg.org/mda/specs.htm>.

[23] Oliveira, T.C.; Alencar, P. S. C.; Cowan, D. D. *ReuseTool - An extensible tool support for object-oriented framework reuse.* Journal of Systems and Software 84(12): 2234-2252 (2011).

[24] Ortigosa, A., Campo, M., 1999 June. *Smartbooks: a step beyond active-cookbooks to aid in framework instantiation*, Technology of Object-Oriented Languages and Systems 25. IEEE Press.

[25] Park, S.; Park, S.; Sugumaran, V. *Extending reusable asset specification to improve software reuse.* In Proceeding of SAC '07 Proceedings of the 2007 ACM symposium on Applied computing, 2007. pp 1473 – 1478.

[26] Pillat, R. M., Basso, F. P.; Oliveira, T. C., Werner, C. M., *Ensuring Consistency of Feature-based Decisions with a Business Rule System*, International Workshop on Variability Modeling of Software-intensive Systems (VaMoS), January 23-25, 2013, Pisa. pp 81-88.

[27] *Reusable Asset Specification*. Av. at <http://www.omg.org/spec/RAS/>. At Mar. 2013.

[28] Santos, R.; Werner, C. *Analyzing the Concept of Components in the Brechó-VCM Approach through a Sociotechnical and Software Reuse Management Perspective*. In Fourth Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS), 2010. pp 26-35.

[29] Silva M.; Oliveira T. *Towards detailed software artifact specification with SPEMArti*. In Proceedings of the 2011 International Conference on Software and Systems Process (ICSSP '11). ACM, New York, NY, USA. pp 213-217.

[30] Schuenck, M. , Negócie, Y. , Elias, G. , Miranda, S. , Dias Jr., J., Cavalcanti, G. *X-ARM: An Asset Representation Model for Component Repository Systems.* In Proceedings of SAC’06, April, 23-27, 2006, Dijon, France. pp 1690-1694.

[31] Shan T. C; Hua W. W. *Taxonomy of Java Web Application Frameworks*. In 2006 IEEE International Conference on e-Business Engineering (ICEBE 2006), 24-26 October 2006, Shanghai, China. pp 378-385.

[32] *WCT: Work CASE Toolkit - Project Description*. At March 2013. Available at <http://prisma.cos.ufrj.br/wct>

[33] *WCT: Adapit Company - Tutorials and Softtwares*. At March 2013. <http://www.adapit.com.br/portal.html?sys=wct>

Using Prolog Rules to Detect Software Design Patterns: Strengths and Weaknesses

Hamdi A. Al-Jamimi and Moataz Ahmed
King Fahd University of Petroleum and Minerals
Information and Computer Science Department
Dhahran, Saudi Arabia
{aljamimi, moataz}@kfupm.edu.sa

***Abstract*- Software design patterns reflect best practice solutions applied to frequent design problems. In the literature, various approaches have been proposed to detect the occurrences of design patterns. Prolog rules and similar formalisms have already been utilized for reasoning about the structural and behavioral properties of the software design. Nevertheless, these approaches suffer some weaknesses that we aim to reveal and discuss. In this paper, we discuss Prolog based approaches to detect the design patterns in software. In addition, we evaluate the capability of Prolog rules and highlight the strengths and weaknesses.**

Keywords— design pattern; software design; prolog rules; detection.

I. INTRODUCTION

As models become larger and more complex, they become more difficult to comprehend and maintained. Since design patterns are recurring and well-understood design fragments, it follows that analyzing models in terms of design patterns as well as detecting occurrences of these patterns support models maintenance and comprehension. The design patterns are adopted to reuse the best practices solutions applied to reoccurring problems for the sake of software quality improvement [1].

The design pattern describes the design solution and the different roles and responsibilities. It specifies also the pattern's applicability, and the collaborations occurring between the components [2]. Each design pattern comprises a number of participants (classes, attributes, and operations). Each participant plays a certain role manifested by its name. However, when applying or composing the design pattern in an application, there is a need to adapt the roles names to reflect the application domain. Moreover, overlapping between design patterns makes it hard to identify the patterns. For instance, in software system design containing several design patterns, it is not easy to identify the participants of each design pattern. This is due to the implicit nature of design patterns in software system design. Therefore, detection of design patterns needs to be automated as much as possible. Furthermore, when detecting the occurrences of design patterns, it is beneficial to use the two aspects of the design pattern namely the static and dynamic aspects. The static aspect represents the structural connections, while the dynamic aspect represented by protocol of actions. Design patterns detection has received a great deal of attention in the literature. The logic approaches tried to detect the design patterns by using Prolog to represent software designs and patterns, and utilizing the Prolog engine to accomplish the search [3]. In general, logic based approaches use Prolog rules to represent the design patterns while the software design is described as Prolog facts. Then, Prolog queries are used to detect the occurrence of design patterns and identify them. Despite of the power of logic queries, these approaches suffer some weaknesses. This paper state the capability of using Prolog rules to represent the structural and behavioral aspects of the design pattern, in order to search for design pattern instances in the design. This paper details the strengths and weaknesses of the logic-based approaches based on the analysis of the previous studies and the Prolog programs we implemented.

The remaining of the paper is organized as follows. Section II surveys related work. The design patterns representation using Prolog rules is presented in Section III. Section IV discusses the strengths and weaknesses of Prolog rules-based detection approaches. Section V concludes and introduces open issues to be considered as future work.

II. RELATED WORK

In the literature, some detection approaches use logic based formalisms to encode pattern constraints and inference engines to detect them. Some of these approaches are discussed in the sequel.

Kramer and Prechelt [4] provide one of the first implementations to automatically recognize design patterns using Pat system. The Pat system detects structural design patterns, where rules are used to represent the patterns and prolog queries to execute the search. The design constructions are also represented in Prolog as facts. Kramer and Prechelt focus only on one type of design patterns i.e. the structural design patterns. For each pattern they only consider a single implementation variant [1]. Their evaluation shows that design patterns detection precision based on four benchmarks is between 14 and 50 percent which is not sufficient. Although the recall reaches 100 percent the result may still have false negatives[1]. Similarly, Wuyts [5] uses a logic meta language, called SOUL, to express, reason about and extracts a system's structure. The programs entities and the hierarchy structure of system are described as facts, where they were extracted using

[1] We mean by false negatives the set of design patterns occurrence that were not detected by a pattern detector but they are contained in the design.

static analysis. The design motifs are described as Prolog predicates on the facts. However, due to the use of a Prolog engine this approach has performance issues. Specifically, it cannot deal with variants automatically, also the inherent complexity when identifying subsets of classes matching design motif descriptions.

The static and dynamic analyses were combined by Heuzeroth et al. [6] to detect design patterns. To allow automatically generating the static and dynamic analyses, they provide two Prolog-based languages to specify design patterns. They use SanD-Prolog which is a low-level language that includes Prolog predicates to depict the static structure and dynamic behavior of design patterns. A high-level language, called SanD, has been used to specify design patterns. Despite of its effectiveness, this approach suffers some limitations: the specifications of SanD-Prolog seem to be lengthy and complex. SanD is tend to be more intuitive, however less powerful. Additionally, Huang et al. [7] present an approach to support pattern recovery. Their approach is based on the structural and run time behavioral analysis. Different types of design patterns are considered; structural, creational and behavioral patterns. The structural and behavioral aspects have been utilized. To depict this representation, they introduce a combination of predicate logic and Allen's interval-based temporal logic. Then to support pattern recovery, the formal specifications could be easily converted into Prolog representations. Stoianov and Sora [8] propose a logic-based detection approach used to detect a set of patterns and antipatterns. The approach defines Prolog predicates to describe the structural and behavioral aspects of design patterns and antipatterns. Prolog predicates simplify the description of patterns and antipatterns characterized by structural and complex behavioral aspects. They compared the results of the proposed approach with the results provided by a common tool on the same analyzed systems. In the case of two patterns out of five there are significant differences between the obtained results. TABLE I summarizes the studies presented in this section. Despite their scientific contributions, we believe that these approaches have a number of drawbacks that will be discussed in details in Section IV.

TABLE I SUMMARY OF THE LOGIC-BASED APPROACHES

Study	Evaluation Criteria				
	Structural	*Behavioral*	*Pattern Representation*	*System Representation*	*Matching Technique*
Kramer et al.[4]	√		Prolog	Prolog	Exact
Wuyts [5]	√		Prolog	Prolog	Exact
Heuzeroth et al. [6]	√	√	Prolog-based language	Prolog	Exact
Huang et al. [7]	√	√	Prolog	--	Exact
Stoianov et al. [8]	√	√	Prolog	Prolog	Exact

III. PROLOG REPRESENTATION

A knowledge representation (KR) is defined as reasoning about the world instead of taking action in it. The power of KR is reflected by the ability to make inferences, i.e. draw new conclusions from existing facts. In this work, Prolog facts represent the software design, while Prolog rules represent the design patterns. The design pattern describes structural connections between two classes through call, delegation or inheritance relations. In addition, it depicts interactions between objects of these classes and sequences of actions. In the following, we demonstrate an experiment where we represent design patterns and software design by a set of Prolog rules and facts respectively. Our goal is purely to determine whether a given fact (software design) can be inferred efficiently from the specified rules (design patterns) or not. The implemented Prolog program consists of three sets: *first*, rules that represent each design pattern in terms of its participating classes and their relations. *Second*, facts corresponding to the software design. The classes name might be different from the classes presented in the rules; however, they might be matched in the relations, methods and exchanged messages. *Third*, a Prolog inference algorithm unifies the defined predicates and extracted facts to recognize classes playing roles in a particular design pattern.

A. Pattern Representation

Our representation is inspired by earlier work of Kramer et al. [4]. Here, we consider the structural and behavioral aspects of the design pattern. Conversely, in [4] they consider only the structural aspect. We represent a set of design patterns, using a set of Prolog rules which can be defined as a predicate expression. The presented rule specifies essential and adequate characteristics of classes, their different relations, and method invocations to form a pattern instance. In the following we demonstrate the syntax of the design pattern representation that we used in our experiment.

```
patternName(participating classes):-
  class(classType ,participatedClass),
  attribute(participatedClassM,attrName),
  operation(oprType,oprScope
        participatedClassM,oprName,calleeClass),
  inheritance(participatedClassX, participatedClassY),
  association(participatedClassX, participatedClassY),
  aggregation(participatedClassX,MultiplicityX
        participatedClassY, MultiplicityY).
```

The above representation of the design pattern starts by the pattern's name and classes participating in the pattern. Then for each participated class, its name and type either abstract or concrete are provided. Similarly, another clause is added to describe the mandatory attributes and their classes. The clause describing the operation provides more information; since it describes operation type (abstract or concrete), operation scope (public, private, or protected), name of the class to which operation belongs, operation name, and the callee class in case the operation is called by another class. Finally, the representation contains three additional clauses to represent the relations between the participating classes. The represented relations include inheritance, association and aggregation. For each design pattern, Prolog rule assembles all the required facts in order to identify a pattern instance. Our Prolog program contains the representation for five design pattern namely Strategy, Bridge, Factory Method, Strategy, and Iterator. We focus to represent at least one pattern from each design pattern category; creational, structural and behavioral patterns. Due to space limitation, we cannot show the representations of the considered patterns.

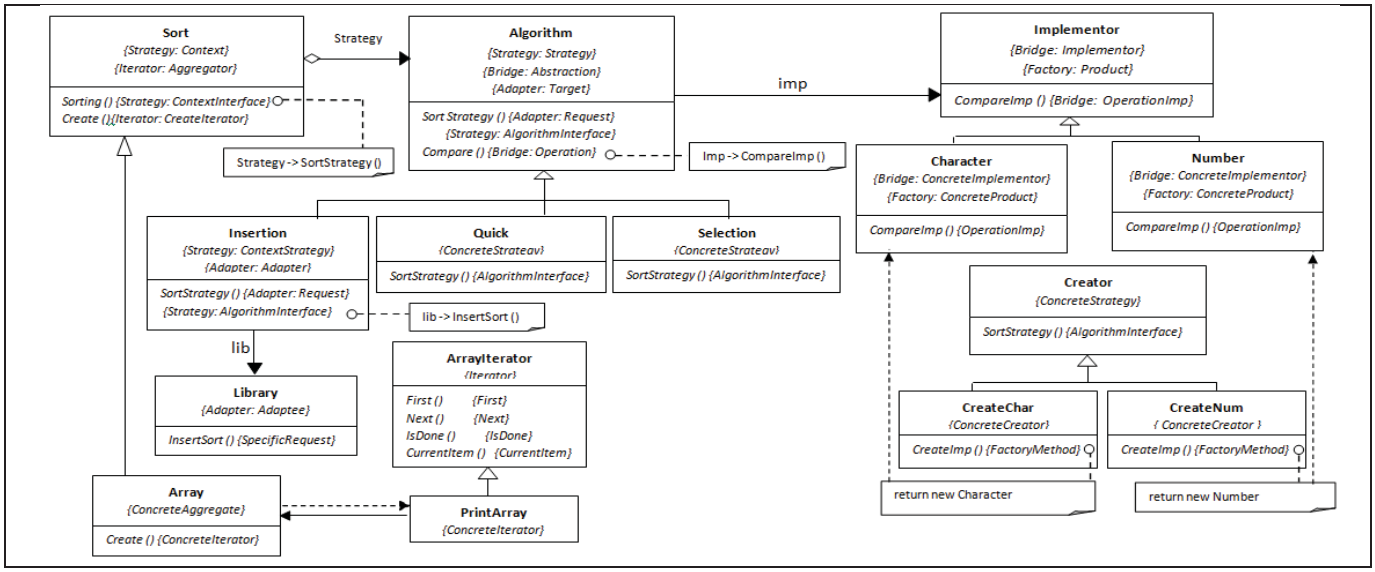


Figure 1. A design of system sort

B. System Representation

A set of Prolog facts is used to represent the extracted knowledge from a design of software system. Furthermore, information about the interactions among classes is required to detect the design patterns occurrences. Figure 1, adapted from [9], presents a design of system sort that includes the applications of several design patterns namely: Adapter, Strategy, Bridge, Iterator, and Abstract Factory. In the diagram, for each design element it is not clear in which patterns the design element participate. In our representation, Prolog facts are used to represent the system design. The Prolog fact can be defined as a predicate expression used to formulate a declarative statement about the problem domain. Prolog interpreter answers queries about the rules and facts represented and stored in its database. It is assumed, to represent what is true about the existence of design patterns. When asking Prolog whether it can prove that a certain design pattern is in the presented software design. It answers 'yes', if so, and it shows any variable bindings to the design pattern. Otherwise, it answers 'no' when it fails to prove the query is true.

C. Results

We used the above representation to query about the occurrences of the prescribed design patterns. The recall and precision measures were used to measure the performance of the detection. While recall indicates the portion of relevant design pattern instances that are retrieved, precision refers to the portion of retrieved design pattern instances that are relevant. As shown in Figure 1, the number of existing patterns is varying from a design pattern to another. Indeed, the results of the experiment show a full recall and precision, means that all the design patterns could be detected by using Prolog rules. Nevertheless, we were able to conclude several comments confirming the restrictions and weaknesses of the logic-based design pattern detection. Section IV discusses these comments based on the analyzed survey and the conducted experiment

IV. Evaluation and Discussion

In general, a KR language can be evaluated through four properties: representational adequacy, inferential adequacy, inferential efficiency and acquisitional efficiency. In the light of these properties, this section discusses the strengths and weaknesses of using Prolog rules to detect the design patterns.

A. Strengths

Representational adequacy is the ability of the language to depict all sorts of knowledge required in the described domain. When Prolog is used as a repository of design knowledge, a number of features can be gained.

1. Prolog rules represent the characteristics and constraints of each design element. Thus, these rules can be utilized to verify the design elements consistencies.
2. Prolog can perform exhaustive search for solutions during its execution. As a result, Prolog engine has the ability to detect all occurrences of a specific pattern.
3. Logic based languages in general are able to represent the real world more accurately.

Inferential adequacy refers to the capability to drive new knowledge from the available one through manipulating the representational structures.

1. Prolog assert and retract clauses can be utilized to add and remove structural facts about design elements.
2. Prolog is able to derive new rules from the existing rules contained within the knowledge base.

B. Weakneseys

Although the strengths and the scientific contributions of the Prolog-based studies [4, 6-8], in the following we list several drawbacks of Prolog when detecting design patterns.

The weaknesses are discussed in terms of *inferential efficiency* and *acquisitional efficiency*. The former refers to the ability to integrate more information into the knowledge structure to focus the attention on a promised way; while the latter indicates the capability of utilizing automatic methods to gain new knowledge when possible instead of human intervention.

1. Exact matching has been considered in all the surveyed papers (see TABLE I), also in the example we implemented. In case of the binary detection, even if high percentage of the design is corresponding to the design pattern, it cannot be detected. Moreover, the ranking for the detected patterns is not supported. In reality, the fuzzy conditions need to be represented when describing the patterns; also the detection approach should be able to detect the partially matching.
2. Due to the variations of design pattern instances, it is not easy to decide entirely whether a set of classes in software design is accurately a design pattern instance. The patterns can be formalized by different variants, i.e. a separate rule represents each variant even if the variation is minimal.
3. Prolog engine can detect the classes represented and their relations using the logical rules. That means it is difficult to detect directly similar, not identical, design pattern instances. Thus, the rules may need to be extended to include all the missing design elements. Accordingly, the rules grow rapidly and become impractical to manage.
4. The recall and precision measures show good results either in our experiment or in the surveyed studies. However, when referring to the aforementioned points this accuracy measures are satisfactory with exact matching to instances occurrences and with a specific design pattern variant.
5. Prolog is *closed world assumption,* means that it requires that all the atoms in a domain must be specified. For the not specified rules they will be considered there unless we put restriction which is difficult to be specified. For example, when querying about the occurrence of the *adapter* pattern, the *proxy* pattern may be reterived as *adapter*. The association between the subclasses is not considered because this relation was not described in the rule.
6. Prolog-based approaches have no ability to work directly on the given design models. These approaches require pre-processing to convert the original model representation into more suitable representation for the proposed formalism. In this case, there is a need to convert the design patterns into predicates to allow operating on them. Like this conversion may complicate the live integration with modeling tools.
7. The logic approaches require skills in mathematics and logic that might not be available or hard to learn. Pattern designers in the industry might not be familiar with the proposed mathematical formalisms, making them difficult to use to specify patterns.
8. Most of the surveyed approaches reproduce the design form the source code, which usually doesn't represent the actual design of the system. Not all the relations and interactions are reflected in this conversion.

V. Conclusion and Future Work

Prolog rules and similar formalisms have been utilized in several conducted studies for reasoning about the structural and behavioral properties of the software design. In this paper, we discuss Prolog-based approaches used to detect design patterns. The objective is to emphasize the capability of Prolog rules in representing and detecting the design patterns. We analyzed representative Prolog-based approaches. We also conducted an experiment where the structural and behavioral of a number of design patterns are represented using Prolog predicates. Our analysis and evaluation revealed a list of strengths and weaknesses of the Prolog-based detection approach.

Future work will focus on exploiting the strengths of the Prolog representation in building custom tool that overcome the encountered weaknesses. For instance, there is a need to make flexible query where it is possible to return the partial occurrences of design patterns with a degree of completeness. While previous detection appraoches often detect some patterns with good recall and precision; there is a need to investigate the generalization and scalability of these approaches for complex systems. Through this work we emphasized the importance of considering the static and dynamic analyses when representing the design patterns. However, several patterns have similar structural and behavioral features; as a result it is not easy to recognize them correctly. Therefore, semantic analysis is also required to improve the detection accuracy.

Acknowledgment

The authors would like to acknowledge the support provided by the Deanship of Scientific Research at King Fahd University of Petroleum and Minerals, Saudi Arabia, under Research Grant 11-INF1633-04.

References

[1] Gamma, E., et al., Design Patterns: Elements of Reusable Object-Oriented Software, 1995, New York, NY: Addison-Wesley Professional Computing Series, Addison-Wesley Publishing Company.

[2] Qiu, Q.J.M., et al., Detecting Design Pattern using Subgraph Discovery, in The 2nd Asian Conference on Intelligent Information and Database Systems(ACIIDS).2010.

[3] Clocksin, W.F. and C.S. Mellish, Programming in Prolog. Fifth ed. ed, 2003: Springer-Verlag.

[4] Kramer, C. and L. Prechelt, Design recovery by automated search for structural design patterns in object oriented software, in proceedings of the 3rd Working Conference on Reverse Engineering.1996. p. 208-216.

[5] Wuyts, R., Declarative reasoning about the structure of object oriented systems, in In proceedings of the 26th conference on the Technology of Object-Oriented Languages and Systems, 1998. p. 112-124.

[6] Heuzeroth, D., S. Mandel, and W. L¨owe, Generating Design Pattern Detectors from Pattern Specifications, in In Proc. of the 18th IEEE International Conference on Automated Software Engineering2003, IEEE Computer Society Press: Montreal, Quebec, Canada. p. 245-248.

[7] Huang, H., et al., A practical pattern recovery approach based on both structural and behavioral analysis. Journal of Systems and Software, 2005. 75 (1-2): p. 69-87.

[8] Stoianov, A. and I. Sora, Detecting Patterns and Antipatterns in Software Using Prolog Rules. Proceedings of International Joint Conference on Computational Cybernetics and Technical Informatics, 2010: p.253-258.

[9] Dong, J., Adding pattern related information in structural and behavioral diagrams. Information and Software Technology, 2004. 46(5):p.293-300.

Runtime Monitoring and Auditing of Self-Adaptive Systems

Daniel H. Carmo[*], Sergio T. Carvalho[*+], Leonardo G. P. Murta[*], Orlando Loques[*]
[*]Instituto de Computação, Universidade Federal Fluminense (UFF), Niterói, Brazil
[+]Instituto de Informática, Universidade Federal de Goiás (UFG), Goiânia, Brazil
{dheraclio, scarvalho, leomurta, loques}@ic.uff.br

***Abstract*— Self-Adaptive Systems are target of frequent research regarding different aspects. However, they still present several challenges related to assurance, dependability, verification, and validation. Adaptations can be related to a set of concerns (i.e., why, what, when, where, who, and how), which are evaluated during, operation and post operation phases. We propose the application of configuration management techniques to provide means for monitoring and auditing Self-Adaptive Systems. We introduce a tool named CM@RT that registers how the system architecture configuration evolves over time and provides different visualizations to track such evolution. For evaluating our approach, some Self-Adaptive Systems scenarios were tackled with the help of CM@RT. The results show that our approach is capable of providing means to perform monitoring and auditing with valuable benefits to the selected Self-Adaptive scenarios.**

Keywords- Self-Adaptive Systems; Configuration Management; Monitoring; Auditing; Product Lines

I. INTRODUCTION

Self-Adaptive Systems (SAS) adapt their behaviour in reaction to changes in the runtime context [1]. Today, there is an increasing demand for SAS [2], even for safety-critical applications [3], since SAS are capable of operating in highly dynamic environments [3]. However, considering that SAS are conceived to autonomously react to changes in the runtime context, researches on dependability [4], verification and validation [3], quality [5], among others [1], [2], [6] are crucial, especially for safety-critical applications.

SAS may perform different types of adaptations in response to changes in the runtime context. These adaptation types are based on techniques ranging from parameterization to architecture reconfiguration [2]. The later allows deeper adaptations because parts of the system (i.e., components) can be added, removed, replaced, or reconnected with the remaining parts, resulting in new architecture configurations (AC) [2]. Investigations on AC level adaptation have provided significant results for SAS [7], [8], [9]. For instance, when considering safety-critical applications, the use of Dynamic Software Product Lines (DSPL) techniques leverages dependability by allowing software architects to define, in a preplanned manner, possible AC and transitions among them [10], [11], [12], [13].

However, tracking the AC evolution of SAS, even when adopting DSPL techniques, is a complex task [14]. SAS adaptations can be described in terms of six basic concerns (5W+1H) [2]: *where*, *when*, *what*, *why*, *who, and how*. Identifying and closely following such concerns is expected during operation phase [2]. In addition to this phase, safety-critical applications also require post-operation analysis, for example, to assign responsibilities in the context of a given health care system failure [6]. In this case, software auditing is essential to verify compliance with design specifications. Therefore, these tasks require specific support for evaluating 5W+1H concerns during two phases:

- *Operation phase*. This could help to answer questions such as "why the system did lots of adaptations in the last week?"
- *Post-operation phase*. This could help answer questions such as "which adaptation patterns led to a system failure?"

A natural way to support such tasks is adopting Configuration Management (CM) techniques during runtime, helping to track the architectural evolution in terms of the 5W+1H concerns. Evolution of AC in SAS and its relation to CM has been explored before. First, van der Hoek [15] made initial efforts to track runtime AC evolution via architecture description languages. After, van der Hoek et al. [16] extended the previous work and applied CM to support DSPL anytime variability. Latter, Georgas et al. [4], [17] leveraged dependability by recording adaptations and providing architecture recovery operations and visualization features over SAS historic behavior. However, such researches do not consider ways for describing adaptations in terms of the main SAS 5W+1H concerns [2], concentrating only on the architectural modifications themselves and ignoring the motivation behind them. This lack of information may jeopardize the comprehension of SAS evolution.

This paper proposes the application of CM techniques to provide support for evaluating 5W+1H concerns with monitoring and auditing purposes. To do so, we designed a CM system to work at runtime, named CM@RT. Our approach comprises two complementary phases: information acquisition and information analysis. First, during information acquisition, CM@RT registers in a repository adaptation related aspects, e.g., runtime contexts and AC achieved. Then, during information analysis, CM@RT supports the visualization of the AC evolution via different perspectives, described in Section III. With this tool, users are able to closely follow the SAS evolution during operation and post-operation.

To evaluate the benefits of CM@RT, some scenarios associated to monitoring and auditing tasks were tackled. With the use of our tool, we were able to collect enough information to answer the 5W+1H concerns for these relevant scenarios.

The remaining sections of this paper are organized as follows. Section II gives more details about SAS and highlights the importance of monitoring and auditing for SAS. Section III introduces the CM@RT approach. Section IV provides some implementation details of CM@RT. Section V presents how to perform monitoring and auditing analyses with CM@RT. Section VI describes some related work. Finally, Section VII presents final considerations and outlines some future work.

II. SELF-ADAPTIVE SYSTEMS

SAS usually adopt the MAPE-K adaptation loop [2], [5] to manage their behaviour. This adaptation loop consists of four phases [18]: (1) monitoring the external environment in which the system is executing, (2) analysing the context attributes of the environment, (3) planning for a possible adaptation to react from changes in the environment, and (4) enforcing the adaptation in the system. The adaptation loop usually counts on some kind of adaptation knowledge that supports each of the four previously mentioned adaptation phases.

Researchers have explored different techniques to implement adaptation loops. Among them, we can cite model-driven [10], [13], [19] and contract-based (also called strategy or policy) [2], [12], [20] techniques. However, despite of the differences in adaptation techniques, it is possible to describe and further comprehend the adaptation decisions through the evaluation of the 5W+1H concerns [2]:

- *Where*: questions over adaptation location, e.g., which layer or components should be adapted?
- *When:* questions over temporal aspects, e.g., when to adapt or how often?
- *What:* questions over adaptation strategies, e.g., should we reconfigure the AC or substitute a component?
- *Why:* questions over reasons to adapt, e.g., which is the adaptation goal?
- *Who:* questions over sources of adaptation, e.g., was it a human-driven or context-based adaptation?
- *How*: questions over actions performed in the adaptation, e.g., in which order the changes should take place?

Controlling software evolution is one of the main concerns of CM. It traditionally works at development time and has files as first class artifacts [21]. In an usual CM setting, a version control system is responsible for registering detailed information about modifications in different artifacts, e.g., source code [22] and architecture configuration [15]. On the other hand, an issue tracking system is responsible for registering change requests, which identify the issue context, affected artifacts, and required corrections. In addition, integration between version control and issue tracking has demonstrated as an effective way to clarify 5W+1H concerns [23].

The expected operation of a SAS consists on performing adaptations if and only if they are necessary. However, in some situations the SAS may not perform a prescribed adaptation (i.e., false negative) or perform an unnecessary one (i.e., false positive). The evaluation of the internal conditions would help to identify the reasons of false positive (FP) or false negative (FN) adaptations. In addition, replicating the conditions that lead to them is also complex. Appropriate development and test processes would help to identify such problems at development time, before they actually happen. However, it is very difficult to guarantee that all problems are caught during development. This motivates the use of a monitoring mechanism at runtime. Finally, an auditing mechanism could help to identify after-the-fact malfunctions. For example, wrong adaptations (FP or FN), which were not caught at development time or at runtime. This can be useful considering highly dynamic current pervasive computing scenarios.

III. THE CM@RT APPROACH

The CM@RT approach performs runtime CM over SAS in order to provide means for evaluating 5W+1H concerns during monitoring and auditing activities. These activities are performed through the integration between the CM@RT-Repository and the CM@RT-Visualizer modules. Technology specific components integrate CM@RT to the SAS infrastructure. Thus, allowing both on-line and off-line analysis. The following subsections present how information is acquired by CM@RT-Repository, and subsequently analyzed through CM@RT-Visualizer.

A. Information Acquisition

The runtime information gathering is performed by the before mentioned integration component. This component is responsible for mapping the application specific SAS entities into the CM@RT-Repository metamodel.

After importing the SAS into CM@RT-Repository, the runtime information captured is registered in the metamodel shown in Figure 1. Its relationship to the 5W+1H concerns is described in the following.

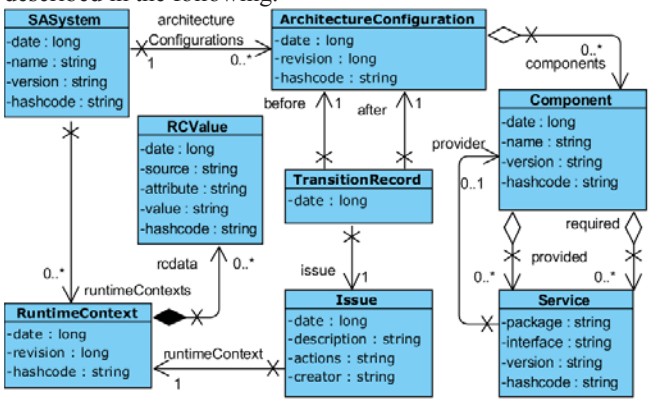


Figure 1 CM@RT metamodel

To answer *where the adaptation should be performed*, CM@RT tracks *Architecture Configurations* materialized

during runtime. This tracking enables comparing AC to reveal the differences between them. The relationship between the AC before and after adaptation is represented by the *Transition Record* entity. It also includes references to the *Issue* entity, which is described further in following paragraphs.

The answer to *when to adapt* depends on when the demand arose and when it was detected. Since SAS are context-aware, the demand appears first on the runtime context, represented by the *Runtime Context* entity. The moment of the demand detection is the time of the *Issue* entity creation. However, not every runtime context results in new demands. In addition, *when to adapt* involves deciding whether to adapt or not, so demands may go unattended. The CM@RT-Repository stores *Runtime Contexts* and *Issues* even if they do not result in AC transitions.

The answer to *who detected the issue* can be found in the attribute *creator* of the *Issue* entity, and it may consist on the SAS itself or on a human. SAS issues are generally concerned with self-* properties [21], e.g., self-healing, or user needs, e.g., controlling room temperature. On the other hand, human-driven issues may have different concerns, e.g., corrective maintenance during operation, testing components during development, or evaluating system behaviour after adaptations. In any case, they relate to the current runtime context and determine modifications in the AC.

The answer to *why the* SAS *adapts* requires evaluation over the *Runtime Context* and the *Issue* associated to a transition between AC. The runtime context shows detailed information over SAS runtime environment, allowing engineers to reason over them. Issues deal with several concerns (discussed in the previous paragraph), which can be registered in the attribute *description* for future analysis. Thus, *Runtime Contexts* and the corresponding *Issues* complement each other for answering this question.

The answer to *how the adaptation occurred* is also associated with the *Issue* through adaptive actions it registers. However, these actions may consist on high-level modifications over the current AC, which leave aside low-level dependencies among components and required services. For example, contracts are concerned with DSPL composition rules only [24], thus not required to address version conflicts of shared libraries among components. It is necessary to evaluate the differences between AC before and after the adaptations to realize the full extent of *how* the SAS was adapted. This could reveal if the SAS low-level dependency solution harmed contract results.

B. Information Analysis

The CM@RT-Visualizer is intended to work connected to CM@RT-Repository, retrieving information from its repository and exposing update events. Through a combination of views, the application provides analysis features that can be applied during *monitoring* and *auditing*.

Monitoring support allows software architects and engineers to evaluate system behavior during runtime. For example, they may evaluate transitions among AC online, since the CM@RT-Visualizer updates its representations just after adaptations are recorded in CM@RT-Repository. It is also possible to evaluate the runtime contexts and issues as soon as they are recorded.

Auditing support is only viable because all information is recorded in a repository, allowing after-the-fact analysis. It includes, among other features, a retrospective view. It is able to graphically replay the transitions performed at runtime in terms of AC diffs, runtime contexts diffs, and associated issues (see Section IV.D for more details).

IV. CM@RT PROTOTYPE

With the aim of evaluating our approach, we develop four prototypes. The first is a SAS framework, which follows the MAPE-K principle. The second is a version of the SCIADS[1] [12], [24], [26] compliant to the SAS framework. The third is the CM@RT-Repository module and the fourth is the CM@RT-Visualizer module.

The main adaptation technology employed in our prototypes is OSGi[2]. OSGi leverages the construction of dynamic systems as sets of components, which can bind to services dynamically and automatically. The following subsections describe the developed prototypes.

A. The SAS Framework

The SAS Framework was implemented as independent OSGi components. The main component is the SelfAdapter. It supplies the basis of the framework, defining service interfaces, and model classes, and providing utility classes for others SAS Framework components. Other OSGi components (Knowledge, Monitor, Analyzer, Planner, and Adapter) implement specific services following the interfaces required by the SelfAdapter component. For example, the Monitor component implements the monitoring service of MAPE-K defined by the Monitor interface.

B. The SCIADS Version Compliant to the SAS Framework

SCIADS is a safety-critical DSPL-based SAS. Its AC is reconfigured according to adaptation actions determined by contracts and DSPL composition rules [12], [24]. The contracts use runtime context variables to determine high-level modifications in current AC, e.g., insertion or removal of components. In addition, SCIADS AC considers its patient specifics needs and residence characteristics. The combination of these factors results in a great number of possible AC and directly jeopardizes the predictability of system states [13].

C. The CM@RT-Repository Module

The CM@RT-Repository information acquisition module is the main element of our approach, as it provides services to all the other elements. The CM@RT-Repository provides a single API, which follows the Facade pattern, hiding internal services and their relationship.

There are two service groups: repository and diffing. Repository services include SAS registration, information

[1] SCIADS is a home health care system developed at UFF (http://www.tempo.uff.br/sciads/).

[2] Official site: http://www.osgi.org

storage, and querying over metamodel elements such as runtime context, issues, AC, and theirs transitions (see Figure 1). The diffing service supports runtime context and AC comparison.

D. The CM@RT-Visualizer Module

The CM@RT-Visualizer module depends on services provided by the CM@RT-Repository module. For example, the query services of CM@RT-Repository are used to populate the views of CM@RT-Visualizer.

After selecting a SAS among those registered in the repository, the information available for analysis is shown in the **Repository view**. It is organized in four groups: Runtime Contexts, Issues, Transitions, and Architecture Configurations. These groups can be seen in Figure 2, on the left hand side. Selected or general information can be analyzed through seven different views during monitoring or auditing activities, which are described in the following.

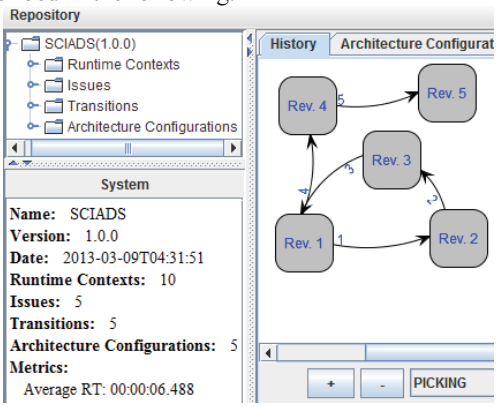


Figure 2 History view

The **History view** presents the SAS evolution as a whole through a graph. Figure 2 shows an example of runtime evolution history, in which SCIADS performed four different transitions. The nodes represent AC and the edges represent the transitions.

The **Architecture Configuration view** presents the SAS AC also through a graph notation (see Figure 3). In this case, the nodes represent components and are identified by name and version, while edges represent the AC topology. Figure 3 also shows metrics for assessing the quality of the AC [27]. The metrics are based in component provided service utilization (PSU) and required service utilization (RSU). The metrics are Average PSU, Average RSU, Compound PSU, and Compound RSU.

The **Architecture Configuration Diff view** (see Figure 4) shows the differences between two AC selected from the **Repository view** through check boxes. It uses the same graph representation used in **Architecture Configuration view**, annotated with color codes, indicating added (green), removed (red), replaced (blue), and unchanged components (gray). In addition, there is a textual description identifying the differences found between the selected AC. This feature also uses the before mentioned color codes to favor visual identification.

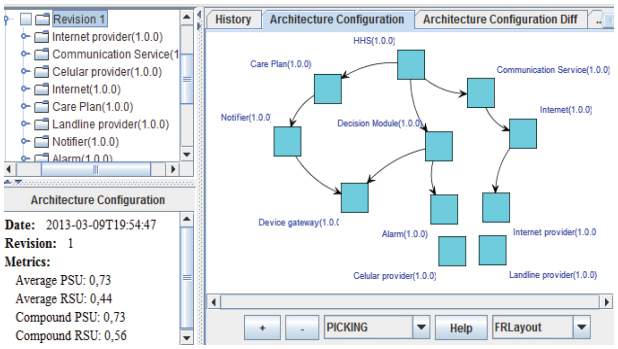


Figure 3 Architecture Configuration view

The **Runtime Context view** represents the context information monitored by the SAS in the form of a grid. This grid contains the date of last update, the source, the attribute name, and the collected value. Similarly, the **Runtime Context Diff view** (see Figure 5) shows the differences between two runtime contexts selected from the Repository view. The diffing is performed considering each attribute and its value, but ignoring dates and sources, as these data are expected to always change. The Status column shows the results using the same designation and color code of the **Architecture Configuration Diff view**.

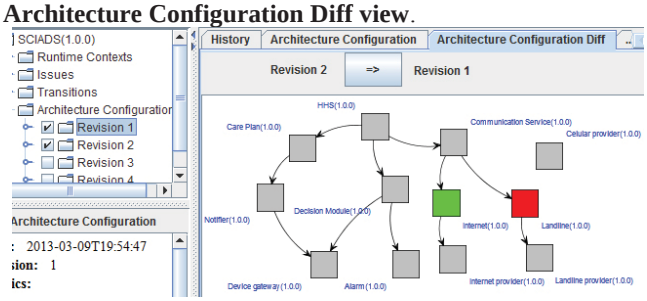
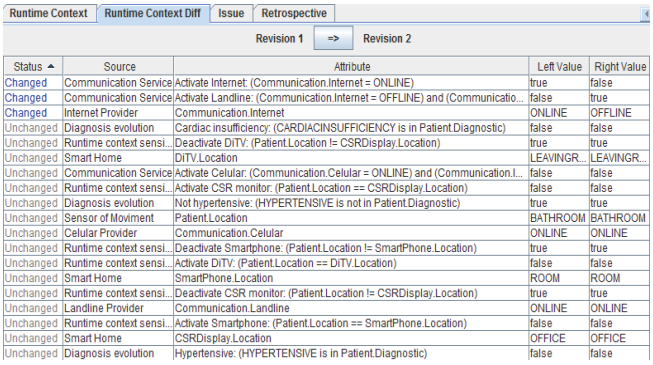


Figure 4 Architecture Configuration Diff view



Status	Source	Attribute	Left Value	Right Value
Changed	Communication Service	Activate Internet: (Communication.Internet = ONLINE)	true	false
Changed	Communication Service	Activate Landline: (Communication.Internet = OFFLINE) and (Communicatio...	false	true
Changed	Internet Provider	Communication.Internet	ONLINE	OFFLINE
Unchanged	Diagnosis evolution	Cardiac insufficiency: (CARDIACINSUFFICIENCY is in Patient.Diagnostic)	false	false
Unchanged	Runtime context sensi...	Deactivate DiTV: (Patient.Location != CSRDisplay.Location)	true	true
Unchanged	Smart Home	DiTV.Location	LEAVINGR...	LEAVINGR...
Unchanged	Communication Service	Activate Celular: (Communication.Celular = ONLINE) and (Communication.I...	false	false
Unchanged	Runtime context sensi...	Activate CSR monitor: (Patient.Location == CSRDisplay.Location)	false	false
Unchanged	Diagnosis evolution	Not hypertensive: (HYPERTENSIVE is not in Patient.Diagnostic)	true	true
Unchanged	Sensor of Moviment	Patient.Location	BATHROOM	BATHROOM
Unchanged	Celular Provider	Communication.Celular	ONLINE	ONLINE
Unchanged	Runtime context sensi...	Deactivate Smartphone: (Patient.Location != SmartPhone.Location)	true	true
Unchanged	Runtime context sensi...	Activate DiTV: (Patient.Location == DiTV.Location)	false	false
Unchanged	Smart Home	SmartPhone.Location	ROOM	ROOM
Unchanged	Runtime context sensi...	Deactivate CSR monitor: (Patient.Location != CSRDisplay.Location)	true	true
Unchanged	Landline Provider	Communication.Landline	ONLINE	ONLINE
Unchanged	Runtime context sensi...	Activate Smartphone: (Patient.Location == SmartPhone.Location)	false	false
Unchanged	Smart Home	CSRDisplay.Location	OFFICE	OFFICE
Unchanged	Diagnosis evolution	Hypertensive: (HYPERTENSIVE is in Patient.Diagnostic)	false	false

Figure 5 Runtime Context Diff view

The **Issue view** shows details of the registered issues. Figure 6 shows the inclusion date in the repository, the demand description, the actions required to modify the AC, and a textual description of the effects over the AC before the adaptation. It also has buttons to show the runtime context (**Runtime Context view**) and the difference between the AC before and after the adaptation (**Architecture Configuration Diff view**).

Finally, the **Retrospective view** uses the transition records from CM@RT-Repository to present an animation of the adaptations performed by the SAS system, replaying its operation for a chosen period. It is a composition of three views: **Architecture Configuration Diff view** (see Figure 4), **Runtime Context Diff view** (see Figure 5), and **Issue view (see Figure 6)**.

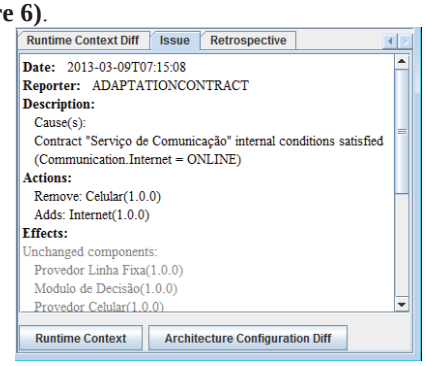


Figure 6 Issue view

V. MONITORING AND AUDITING ANALYSES WITH CM@RT

This section describes how CM@RT can help performing monitoring and auditing analyses under three scenarios. These scenarios use SCIADS as a concrete example. In this section, we assume that all prototypes are active during runtime in the OSGi platform. Monitoring and auditing performed using CM@RT-Visualizer features occurs as follows.

A. *False Positive and False Negative Adaptations Detection*

In SCIADS, identifying FP or FN adaptations require the evaluation of several contracts, considering their internal conditions. In Figure 5, the CM@RT-Visualizer shows examples of these internal conditions in the first 2 rows, along with corresponding values. The internal conditions were added to runtime context to enable their evaluation based in value differences between two runtime contexts.

To identify a FP the user can also check the registered issues. The Issue view reveals which demands the SAS found on a particular runtime context in the description field. To identify FN, it is necessary to go through the registered runtime contexts. The Runtime Context Diff view helps to identify significant changes between runtime contexts, necessary to locate situations where adaptations should have occurred but did not.

B. *Adaptation Cycles Detection*

Another possible monitoring scenario consists on evaluating the SCIADS adaptation rate in a patient's home. Besides adopting massive tests at development time, each patient home has its own requirements and features. Sometimes these features cannot be fully predicted, thus being a source of uncertainty to the SAS [28]

Monitoring new AC transitions thought CM@RT-Visualizer reveals the adaptation rate. Furthermore, it is possible to monitor the rate of runtime context updates and issues found. Monitoring runtime context and issues could reveal anomalies during operation which were missed during development. For example, the temperature thresholds configured according to the development site may be inadequate to the operation site, leading patient discomfort.

C. *After-the-fact Adaptations Tracking*

In SCIADS, if the patient is under dangerous health conditions, adaptations are severely restricted. During auditing of such behaviour, the manual analysis of textual system runtime logs would be counterproductive and error prone. The existence of a tool capable of representing historical information with semantic driven visualizations, make the auditing process more efficient and trustable if compared to textual analysis.

The Retrospective view represents the progression of the SAS adaptations, enabling their evaluation by specialists. The view shows at the same time what changed, why it did, when it happened, and who requested it. In addition, the progression of the runtime context may reveal the effect of the adaptations in the SAS environment.

VI. RELATED WORK

Few works provide runtime CM infrastructure to manage AC evolution. Van der Hoek et al. [15], show that the AC of dynamic system evolve as well as source code, and were the first to propose the use of CM for managing this evolution. They developed an integrated architecture-driven environment called Mae. Mae provides features such as architecture evolution control, runtime adaptation patch generation, and product variant selection. In [16], van der Hoek continues to explore architecture evolution control for any time variability on DSPL. The approach comprises two applications: Ménage for evolution control and SelectorDriver for evolution handling. Ménage is part of the Mae environment, but focus only on the development phase. Our approach is complementary to these in the sense that we propose an infrastructure for tracking the architectural evolution during operation and post-operation phases, while their approach focuses on the development phase.

Georgas et al. [4] use runtime CM to control architectural evolution and to leverage dependability on SAS through the use of Architectural Runtime Configuration Management (ARCM). ARCM is integrated to Eclipse IDE through a plugin [4], and comes with three main features: runtime architectural evolution control, graph-based visualization of architectural evolution, and architectural recovery facilities. In [17], Georgas et al. describe the use of ARCM to provide visibility and understandability over SAS runtime behavior and means for human intervention over the adaptation process. Despite the fact that it has some similarities with our approach, it is limited to AC evolution. Our approach tracks several other architectural evolution concerns, such as runtime context and related issue, and encompasses operation and post-operation phases.

VII. CONCLUSION

CM@RT represents our initial efforts on supplying CM at Runtime to provide a monitoring and auditing infrastructure for SAS, with special attention to DSPL. CM@RT-Repository

provides tracking functionalities over AC evolution and related information. Complementing the core application, we provide a repository visualization tool that supplies consolidated information and mechanisms for monitoring and auditing. Thus, CM@RT enables short, medium, and long time analysis over SAS behavior. In addition, since the CM@RT was designed to be deployed with the target SAS, it runs on-line in production and development environment.

We also demonstrated how to perform monitoring and auditing with CM@RT. This provides some initial evidences that CM@RT is capable of providing behavior information for monitoring and trace information for auditing. In addition, we provide some SPL metrics for quality assessment and analysis support (see Figure 3). In the future, we intend to perform user-centered experiments to raise more evidence of its benefits.

We believe the application of data mining on the repository would reveal the existence of significant adaptation patterns. For example, policies conflicts, components interoperability conflicts, or singular situations at runtime environment could be detected. With these patterns in hand, it would be possible to enhance the effectiveness of the adaptation contracts by treating previous patterns causes and, consequently, avoiding unstable architecture configurations. In addition, there are plans to use this information for developing automated analysis features.

ACKNOWLEDGMENT

The authors would like to thank CAPES, CNPq, and FAPERJ for their financial support.

REFERENCES

[1] B. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, and others, "Software engineering for self-adaptive systems: A research roadmap," *Software Engineering for Self-Adaptive Systems*, pp. 1–26, 2009.

[2] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM Trans. Auton. Adapt. Syst.*, vol. 4, no. 2, pp. 1–42, May 2009.

[3] G. Tamura, N. M. Villegas, H. A. Müller, J. P. Sousa, B. Becker, M. Pezzè, G. Karsai, S. Mankovskii, W. Schäfer, L. Tahvildari, and Wong, Kenny, "Towards practical runtime verification and validation of self-adaptive software systems," in in *Software Engineering for Self-Adaptive Systems 2*, vol. 7475, Springer, 2013, pp. 108–132.

[4] J. C. Georgas, A. Van Der Hoek, and R. N. Taylor, "Architectural runtime configuration management in support of dependable self-adaptive software," *SIGSOFT Softw. Eng. Notes*, vol. 30, no. 4, pp. 1–6, Maio 2005.

[5] N. M. Villegas, H. A. Müller, G. Tamura, L. Duchien, and R. Casallas, "A framework for evaluating quality-driven self-adaptive software systems," in *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, Waikiki, Honolulu, HI, USA, 2011, pp. 80–89.

[6] O. Loques and A. Sztajnberg, "Adaptation issues in software architectures of remote health care systems," in *Proceedings of the 2010 ICSE Workshop on Software Engineering in Health Care*, 2010, pp. 24–28.

[7] D. Garlan, B. Schmerl, and S. W. Cheng, "Software architecture-based self-adaptation," *In Autonomic computing and networking*, no. Springer, pp. 31–55, 2009.

[8] P. Oreizy, M. M. Gorlick, R. N. Taylor, D. Heimhigner, G. Johnson, N. Medvidovic, A. Quilici, D. S. Rosenblum, and A. L. Wolf, "An architecture-based approach to self-adaptive software," *Intelligent Systems and Their Applications, IEEE*, vol. 14, no. 3, pp. 54–62, 1999.

[9] B. Hayes-Roth, "An architecture for adaptive intelligent systems," *Artificial Intelligence: Special Issue on Agents and Interactivity*, vol. 72, pp. 329–365, 1995.

[10] N. Bencomo, P. Sawyer, G. Blair, and P. Grace, "Dynamically adaptive systems are product lines too: Using model-driven techniques to capture dynamic variability of adaptive systems," in *2nd International Workshop on Dynamic Software Product Lines*, Limerick, Ireland, 2008, vol. 2, pp. 23–32.

[11] T. Dinkelaker, R. Mitschke, K. Fetzer, and M. Mezini, "A Dynamic Software Product Line Approach Using Aspect Models at Runtime," *5th Domain-Specific Aspect Languages Workshop*, Mar. 2010.

[12] S. T. Carvalho, O. Loques, and L. Murta, "Dynamic Variability Management in Product Lines: An Approach Based on Architectural Contracts," presented at the IV Brazilian Symposium on Software Components, Architectures and Reuse, Bahia, Brazil, 2010, pp. 61–69.

[13] B. Morin, O. Barais, J.-M. Jézéquel, F. Fleurey, and A. Solberg, "Models@ Run.time to Support Dynamic Adaptation," *IEEE Computer*, vol. 42, no. 10, pp. 44–51, Oct-2009.

[14] N. López, R. Casallas, and A. Van Der Hoek, "Issues in mapping change-based product line architectures to configuration management systems," in *Proceedings of the 13th International Software Product Line Conference*, 2009, pp. 21–30.

[15] A. Van Der Hoek, M. Mikic-Rakic, R. Roshandel, and N. Medvidovic, "Taming Architectural Evolution," in *ACM SIGSOFT Software Engineering Notes*, 2001, vol. 26, pp. 1–10.

[16] A. Van Der Hoek, "Design-Time Product Line Architectures for Any-Time Variability," *Science of Computer Programming*, vol. 53, no. 3, pp. 285–304, 2004.

[17] J. C. Georgas, A. Van Der Hoek, and R. N. Taylor, "Using Architectural Models to Manage and Visualize Runtime Adaptation," *IEEE Computer*, vol. 42, no. 10, pp. 52–60, 2009.

[18] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *IEEE Computer*, vol. 36, no. 1, pp. 41–50, Jan. 2003.

[19] G. Blair, N. Bencomo, and R. B. France, "Models@run.time," *IEEE Computer*, vol. 42, no. 10, pp. 22–27, 2009.

[20] D. Garlan, S. W. Cheng, A. C. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: Architecture-based self-adaptation with reusable infrastructure," *IEEE Computer*, vol. 37, no. 10, pp. 46–54, 2004.

[21] P. Horn, "Autonomic Computing: IBM's Perspective on the State of Information Technology," International Business Machines Corporation, 278606109, 2001.

[22] B. Collins-Sussman, B. W. Fitzpatrick, and C. M. Pilato, *Version Control with Subversion*, 2nd ed. Sebastpol, CA, USA: O'Reilly Media, 2008.

[23] C. R. Dantas, L. G. P. Murta, and C. M. L. Werner, "Mining Change Traces from Versioned UML Repositories," in *Brazilian Symposium on Software Engineering (SBES)*, João Pessoa, Brazil, 2007, pp. 236–252.

[24] S. T. Carvalho, L. Murta, and O. Loques, "Variabilities as First-Class Elements in Product Line Architectures of Homecare Systems," in *4th International Workshop on Software Engineering in Health Care*, Zurich, Switzerland, 2012, pp. 33–39.

[25] K. Suzaki, T. Yagi, K. Iijima, N. A. Quynh, C. Artho, and Y. Watanebe, "Moving from logical sharing of guest OS to physical sharing of deduplication on virtual machine," in *Proc. 5th USENIX Workshop on Hot Topics in Security (HotSec 2010), USENIX, Washington DC, USA*, 2010.

[26] S. T. Carvalho, A. Copetti, and O. Loques, "Ubiquitous Computing System in Home Health Care," *Journal of Health Informatics*, vol. 3, no. 2, pp. 51–57, 2011.

[27] A. Van Der Hoek, E. Dincel, and N. Medvidovic, "Using service utilization metrics to assess the structure of product line architectures," in *Software Metrics Symposium, 2003. Proceedings. Ninth International*, 2003, pp. 298–308.

[28] N. Esfahani and S. Malek, "Uncertainty in Self-Adaptive Software Systems," in in *Software Engineering for Self-Adaptive Systems 2*, vol. 7475, Springer, 2012, pp. 214–238.

An ontology-based user model for personalization of educational content

Joice B. Machado, Gustavo L. Martins, Seiji Isotani, Ellen F. Barbosa
University of São Paulo (ICMC/USP)
São Carlos (SP), Brazil
email: joicebm@icmc.usp.br, gustavolivrare@gmail.com, sisotani@icmc.usp.br, francine@icmc.usp.br

***Abstract*—Personalization of educational content is a complex issue that needs to be addressed in order to enable learning systems to adapt the didactic materials and meet students' needs. Current educational approaches have been developed with interactive and personalized environments to fit the profiles of their users. Although many user models have been developed to date, most of them are specific for a given domain. Our research introduces the flexibility of considering different knowledge domains and its personalization for each learner, based on content modeling and ontologies mapping. We developed an ontological approach to content personalization based on user model, including learning styles, goals and background. In particular, this paper focuses on the presentation and validation of an ontology-based user model, referred as to ONTO-USERMODEL. To achieve our goal, we instantiated ONTO-USERMODEL with 300 user profiles and executed a variety of queries to check if the proposed ontology helps to store and retrieve information that support personalization of educational content. As a result different granularities were identified for each learner and content presentation was personalized according to the mapped profiles in the ontologies.**

***Keywords*-content personalization; ontologies; user model; educational content.**

I. INTRODUCTION

The increasing of semantic technologies has provided to the developments computational systems the ability of dealing with the meaning and relationships among information in a variety of applications. In this research line, user-oriented searches, focused on personalization issues, have emerged as a mean for promoting information retrieval [1].

Personalization aims at providing customized replies to particular actions of the users in order to achieve specific requirements. Besides that, it intends to give meaning to information that is not *intelligently* used in systems [2].

Several approaches [3], [2] for dealing with personalization have been investigated. Among them, we highlight the use of the semantic technologies and, particularly, the use of ontologies. Ontologies can be defined as formal specifications of a shared conceptualization [4]. In the educational context, they have been investigated in order to provide facilities such as recovery of learning objects and content personalization [5], [6], [7].

On the other hand, to support the implementation of personalization mechanisms a well-designed and structured user model is fundamental to provide information that can be used to change the materials according to individual's characteristics. Although many user models have been developed to date, most of them cannot be easily used and reused in different contexts due to their domain specificity.

Our approach were developed in order to enable the personalization of different knowledge domain to users distinct. Moreover, it allows the content modeling according to the IMA-CID approach [8], that facilitates the reuse of modeled learning objects. In a previous work, we proposed ONTOTOLEARNER, an ontological infrastructure for content personalization [9], [10]. It is composed of four main ontologies: (i) ONTO-IMACID; (ii) ONTO-USERMODEL; (iii) ONTO-NAVIGATION; and (iv) ONTO-DOMAIN.

In this paper we focus on the presentation of ONTO-USERMODEL, an ontology-based user model, discussing its role in ONTOTOLEARNER as well as its implications for the personalization of educational content. The results suggest that ONTO-USERMODEL enables the extraction and combination of information about users.

This paper is organized as follows. In Section II we provide a literature review regarding the ontological approaches for representing user models. In Section III we present an overview of the set of ontologies we have worked on. In Section IV we describe ONTO-USERMODEL, focusing on the benefits and impacts for personalization that the ontology provides. In Section V we discuss the automatic instantiation of ONTO-USERMODEL. Finally, in Section VI we present our conclusions and perspectives for further work.

II. RELATED WORK

Several ontological approaches for representing user's models can be found in the literature. Jovanovic *et al.* [7] used ontologies to enable the automatic annotation of content. As a result, content can be assembled in different learning objects and customized according to the students' knowledge, preferences and learning styles. However, the management of educational ontologies is not an easy task.

In this direction, Gaeta et al. [5] propose an approach to manage the life-cycle of ontologies that define personalized e-Learning experiences in blended learning environments. From the content annotated with well-managed ontologies it is possible to create rules to adapt the content presentation considering students' characteristics.

Our ontological approach for personalization differs from the initiatives in this section since it is based on an ontolog-

ical infrastructure that uses semantic mappings of relations as a mean to provide different granularities for content presentation and navigation.

Furthermore, despite the great benefits of previous research findings, there are still many challenges to be addressed regarding personalization. As discussed by Isotani & Mizoguchi [3], one of the main problems when developing systems that support some kind of personalization of content is to link well-designed domain-independent knowledge (ontologies) with domain content. Our research fits in this direction and our ultimate goal is to contribute for the development of semantic applications for information recovery and its adoption in semantic educational systems.

III. An Ontological Infrastructure for Content Personalization

Ontologies, combined with personalization approaches, contribute to knowledge representation/sharing and content personalization can occur simultaneously. In this case, the insertion of semantic relations and the use of inferences ontology concepts, performed during domain and user's instantiation, are essential to support knowledge formalization.

In this perspective, we have established of an ontological infrastructure for content personalization [9], [10]. In short, four ontologies were developed (available at http://www.labes.icmc.usp.br/~joice/ontologies/): (i) Onto-Imacid, which formalizes the concepts of an integrated approach for content modeling, called IMA-CID [8]; (ii) Onto-UserModel, which extends other previous achievements to further describe the student's characteristics and preferences; (iii) Onto-Navigation, which represents different ways to deliver and explore the content; and (iv) Onto-Domain, which refers to the subject domain.

Based on the proposed ontologies, we developed an infrastructure, referred to as OntoToLearner, responsible for integrating all concepts and relationships. A set of rules and mapping mechanisms were also defined and associated with this infrastructure in order to support the reasoning on the information presented in the ontologies. Figure 1 illustrates the composition of OntoToLearner. Properties relating the concepts of the ontologies were defined and mapped according to their meaning in the subject knowledge domain.

By the instantiation of user's models, the inferences can be extracted through queries, which are performed to present different content granularities for each student. In this paper we focus on the Onto-UserModel and its use to personalization of content. This process is highlighted on the right side of the OntoToLearner in Figure 1.

OntoToLearner combines a content modeling approach (IMA-CID) with capabilities of content personalization without hindering the maintenance, evolution or individual applicability of each developed ontology. The ontological infrastructure is also flexible to handle multiple domain ontologies and different personalization rules. To check the potential of OntoToLearner, we instantiated its concepts with several information. For instance, information of different types of users (novice, intermediate, advanced) were created and included in the user model [10].

Additionally, the created queries were also executed in the inference engine of Protégé in order to: (i) extract the elements of IMA-CID ontology; (ii) check the instances of the domain ontology; (iii) relate concepts from the other ontologies to concepts in the user model ontology; and (iv) check the navigation features in the navigation ontology. Finally, more complex queries involving the ontologies and their properties, characteristics of available educational content and other instances of concepts were described aiming at modeling and personalizing the content.

IV. The User Model Ontology

A user model can be defined as the representation of the user information in such a way that systems can consistently extract information from the model and use it to behave differently for different users [11]. Onto-UserModel encompasses the main characteristics and similarities found in the existing ontologies. Concepts such as *Performance*, *Preference*, *Role*, *Feedback* and *Learning Style* were extracted from the ontology proposed by Jovanovic *et al.* [7]. Similarly, concepts such as *Goal*, *Knowledge*, *Individual Traits*, *Competency*, *Activity*, *Background*, *Context Of Work* and *Identification* were considered based on the characteristics established by Brusilovsky & Peylo [12]. Moreover, concepts such as *Competence*, *Interest* and *Activity* were inherited from IMS LIP and IEEE PAPI patterns.

The concepts that integrates the Onto-UserModel are linked to the OntoToLearner (Figure 1 (a)) allowing us to represent different user profiles (Figure 1 (b)) and use them in different situations for different purposes. The personalization is performed for each user, based on his/her personal characteristics (Figure 1 (c)). Also, while some information can be extracted by the system during user registration (about 38,5% of the concepts), other information can be entered by the user through questionnaires (about 54% of the concepts) and other can be provided by the instructor himself/herself (about 7,7% of concepts).

Another important feature of Onto-UserModel is that a concept can be updated by the instructor either during the execution of the application (e.g., during the class) or later (e.g., through the reviews and reports obtained from the students' activities). In this case, the queries (by inference) provide results that allow to update some concepts.

Besides concepts, Onto-UserModel also defines the properties (i.e., relations) between concepts for creating axioms, mappings and inference rules. For example, we have created a property *isResult*, which gives the meaning of a feedback of the system when the user performs a proposed activity. Properties are created according to the concepts definitions, focusing on the context and on the

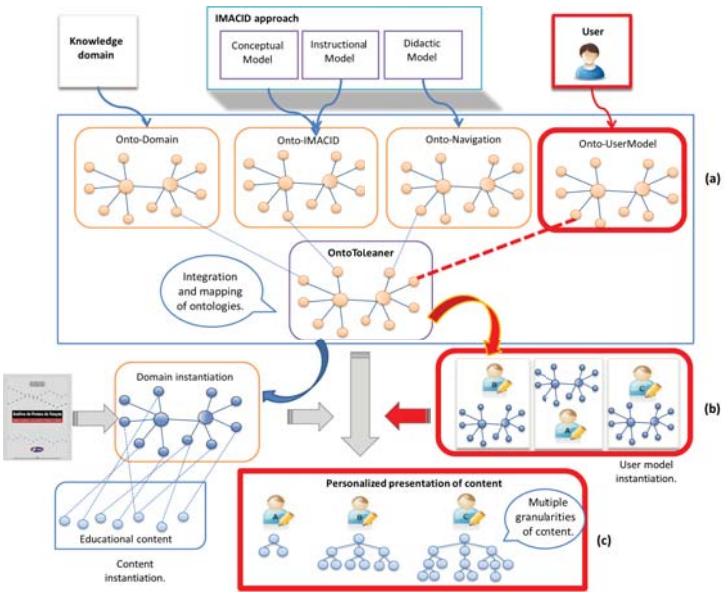


Figure 1. Composition of ONTOTOLEARNER.

inter-relationships that generate the user model. Another example of property is related to the concept *Performance*, which is necessary for analyzing the student's *Feedback* regarding a particular activity. This observation leads to the specification of the property *isNecessaryFor*.

The adoption of ONTO-USERMODEL can assist both in the creation of the educational content as well as in the assessments and in the activities required for a given course. Actually, the construction of a user model ontology allows the instructor to identify the general student's profile as well as its changes after each execution of a set of proposed activities.

V. A CASE STUDY: ONTO-USERMODEL APPLICATION

In order to validate the ONTO-USERMODEL ontology, we instantiated it with 300 user profiles and executed a variety of queries to check if the model helps to store and retrieve information that support personalization of educational content. Basically, the user profiles were created by automatically filling three questionnaires regarding users' learning styles, personal characteristics and knowledge. To ensure that the information from the questionnaires was properly stored and, at the same time, meaningful to adapt the content, we created a set of queries often used in personalization activities.

It is also important to notice that the concepts of the domain ontology (ONTODOMAIN) are directly involved in the queries performed. Thus, the knowledge domain is one of the parameters considered. In our case study, a domain ontology in the context of Function Point Analysis was developed, covering key concepts from the Software Engineering literature related to this metric.

To perform the personalization in agreement with the learning style and the preferences of each user, queries were created to identify the results inferred from the established relationships. Two queries are presented in Figure 2. In Figure 2(a) we intend to extract information regarding the user' learning styles; in Figure 2(b) we intend to get information about users' knowledge levels.

(a) imacid:Concept(?x) ∧ hasMediaOf(?x, image) ∧ userLearner(?y) ∧ user:hasLearningStyle (?y, visual) → concept(?x,?y)
(b) imacid:Concept(?x) ∧ user:Competence(?y, userKnowledgeAdvanced) ∧ main:Learner(?x, Advanced) → concept(?x,?y)

Figure 2. Examples of queries.

As a result, views of users and their profiles were generated. Also, different levels of granularity to present the content were established. Figure 3 presents the results of the queries executed in order to identify the different users' profiles in ONTOUSERMODEL.

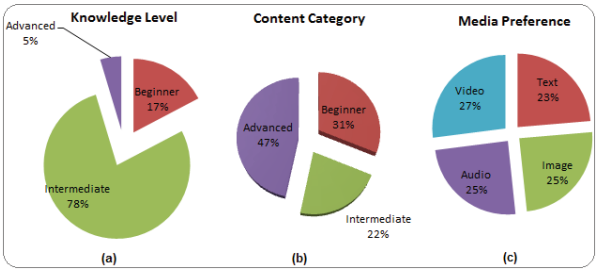


Figure 3. Results of the queries for the users' profiles

The users' profiles can be identified by means of semantic relations formally defined in the ontologies. Furthermore, not only the explicit results can be extracted, but also the implicit ones, which become known due to the combinations of relationships and properties, associated with the concepts of the ontologies. Thus, by knowing the learning style, interest,

knowledge level and preferences, the instructor is able to prepare a course accordingly, taking into account the profiles identified in order to better achieve the learning goals.

Additionally, the instructor can also combine the users' characteristics extracted in order to generate groups of students with similar profiles. As a consequence, significant new information can arise. For instance, it is possible to establish comparison parameters with respect to the performance of the groups of students identified aiming at applying more suitable evaluations. Figure 4 summarizes the queries results from the combination among learning styles in the ONTOTOLEARNER. Considering groups 7 and 8, for instance, the reflexive and visual styles are prevalent; therefore, a course designed for students of these groups should focus on activities that match the reflexive profile with a visual content.

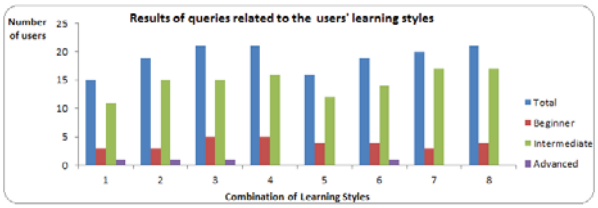


Figure 4. Percentage of learning style for each user profile.

In summary, ONTO-USERMODEL enables the extraction and combination of information about the students, contributing to the creation and delivery of more suitable, personalized content to them. The ontology also supports a better interaction between user and content; indeed, the media presented according to the student's preferences may provide a better user involvement.

VI. CONCLUSIONS AND FUTURE RESEARCHES

In this paper, we presented an ontology-based user model, referred as to ONTO-USERMODEL, which is part of a broader ontological infrastructure for personalization of educational content. In short, relations in ONTO-USERMODEL were semantically mapped, associating the content of the domain ontology (in this paper, the Function Point Analysis domain) to the user's profile. The user model ontology was automatically instantiated with 300 users with different profiles. As a result, personalized content was generated and delivered to each student, according to characteristics such as learning style and knowledge level.

From the results of the inferences in ONTO-USERMODEL, some remarks can be highlighted: (i) the educational content can be automatically personalized for each user through associations and semantic mappings among ontologies; (ii) the establishment of the student's learning style can help the instructor to choose the best evaluation method; and (iii) when the learning goals are not achieved (i.e., the learning is not effective), the reasons can be tracked more accurately based on the set of personal characteristics and learning profile of the user.

As future researches we point out the need of: (i) performing the semantic mappings automatically; (ii) building the queries in a formal rule language; (iii) extracting the user's characteristics implicitly; and (iv) investigating the generality of content in the ONTOTOLEARNER with experiments conducted in classrooms.

ACKNOWLEDGMENTS

The authors would like to thank the Brazilian funding agencies (FAPESP, CAPES, CNPq) and INCT-SEC (CNPq 573963/2008-8, FAPESP 08/57870-926) for their support.

REFERENCES

[1] L. Zhuhadar and O. Nasraoui, "Augmented ontology-based information retrieval system with external open source resources," ser. ITNG '10. IEEE, 2010, pp. 144–149.

[2] M. Baldoni, C. Baroglio, and N. Henze, "Personalization for the semantic web," in *LNCS Tutorial in Reasoning Web.* Springer, 2005, pp. 173–212.

[3] S. Isotani and R. Mizoguchi, "Adventures in the boundary between domain-independent ontologies and domain content for CSCL," in *International Conference on Knowledge-based Intelligent Information and Engineering Systems*. Springer-Verlag, 2008, pp. 523–532.

[4] T. R. Gruber, "Toward principles for the design of ontologies used for knowledge sharing," *Int. J. Hum.-Comput. Stud.*, vol. 43, no. 5-6, pp. 907–928, Dec. 1995.

[5] M. Gaeta, F. Orciuoli, and P. Ritrovato, "Advanced ontology management system for personalised e-learning," *Know.-Based Syst.*, vol. 22, pp. 292–301, May 2009.

[6] S. Jain and J. Pareek, "Automatic identification of granularity level of learning document," in *Technology for Education (T4E), 2010 International Conference on*, 2010, pp. 72 –75.

[7] J. Jovanovic, D. Gasevic, and V. Devedzic, "Ontology based automatic annotation of learning content," *International Journal on Semantic Web and Information Systems*, vol. 2, pp. 91–119, 2006.

[8] E. F. Barbosa and J. C. Maldonado, "Towards the establishment of IMA-CID: An integrated modeling approach for developing educational modules," vol. 17. Journal of the Brazilian Computer Society, 2011, pp. 207–239.

[9] J. B. Machado, "Estudo e definição de ontologias como apoio ao desenvolvimento de módulos educacionais," Master's thesis, Universidade São Paulo, Brasil, mar 2012, in Portuguese.

[10] J. B. Machado, S. Isotani, M. B. Ribeiro, and E. F. Barbosa, "Towards an ontological infrastructure for content modeling and personalization," in *(SMAP)*, 2012, pp. 107 –112.

[11] E. Knutov, P. De Bra, and M. Pechenizkiy, "Ah 12 years later: a comprehensive survey of adaptive hypermedia methods and techniques," *New Rev. Hypermedia Multimedia*, vol. 15, pp. 5–38, April 2009.

[12] P. Brusilovsky and C. Peylo, "Adaptive and intelligent web-based educational systems," *Int. J. Artif. Intell. Ed.*, vol. 13, pp. 159–172, April 2003.

Architectural Design Spaces for Feedback Control Concerns in Self-Adaptive Systems

Sandro S. Andrade[†] [§]
sandros@ufba.br

[§] GSORT Distributed Systems Group
Federal Institute of Education, Science
and Technology of Bahia (IFBa)
40110-150. Salvador-Ba. Brazil

Raimundo José de A. Macêdo[†]
macedo@ufba.br

[†] Distributed Systems Laboratory (LaSiD)
Department of Computer Science
Federal University of Bahia (UFBa)
40170-110. Salvador-Ba. Brazil

Abstract

A lot of current research efforts in self-adaptive systems community have been dedicated to the explicit modeling of architectural aspects related to system self-awareness and context-awareness. This paper presents a flexible and extensible representation of architectural design spaces for self-adaptation approaches based on feedback control loops. We have defined a generic representation for design spaces metamodeling and have instantiated it in order to provide direct support for early reasoning and trade-off analysis of self-adaptation aspects with the aid of a set of feedback control metrics. The proposed approach has been fully implemented in a supporting tool and a case study with a distributed industrial data acquisition service has been undertaken. Whilst preliminary experiences with the proposed approach indicate useful reasoning support when comparing alternative design solutions for self-adaptation, further investigation regarding scalability aspects and automatic handling of conflicting goals has been identified as future work.

Keywords: *software architecture design, self-adaptive systems, feedback control, model-driven software engineering.*

1. Introduction

The increasing cost of handling complexity and changing environments in modern large-scale distributed systems has motivated significant efforts [15] towards the design and development of self-adaptive systems [7]. Architecture-based approaches with explicit (first-class) modeling of feedback control loops [5, 9, 13] have currently been advocated as a promising research landscape, establishing the foundations of domain-independent mechanisms and enabling early reasoning about self-adaptation quality attributes.

In addition, software engineering researchers have been urged to drive their efforts towards an engineering discipline for software [16], which mostly involves the prospection of theories [8, 18] and organization of knowledge for routine use [3]. In particular, the use of design theories [11] for self-adaptive systems is still on its infancy.

A great number of diverse feedback control schemes for regulating and optimizing distinct software performance variables have been proposed over the last past years [14]. Despite some successful achievements of control goals under disturbances and environment uncertainties, most of these work propose ad-hoc domain-specific feedback control approaches. Software architects willing to endow their systems with self-adaptive behavior still have no systematic guidance about choosing among different feedback control strategies, evaluating alternative design trade-offs, and assessing the resulting self-adaptation quality attributes.

In this paper, we present DuSE[1], which is an extensible, navigable and machine-consumable representation of architectural design spaces for self-adaptation feedback control-based approaches. Our approach entails: i) a set of effectors which enable architecture model manipulation when navigating through the design space; ii) a group of OCL (*Object Constraint Language*) rules to verify the availability of variation points in a given design space dimension; and iii) a set of metrics for evaluating self-adaptation quality attributes in a given design space location.

The proposed architectural design space infrastructure relies on three models which address different supporting aspects: i) an architectural model (logical view) of the system to be endowed with self-adaptive behavior, ii) an auto-regressive with exogenous input (ARX) model [10] representing the system dynamics, and iii) the design space model describing the dimensions and associated architecture manipulations.

This research is partially supported by grant 006/2012/PRPGI from the IFBa's Research and Innovation Supporting Program.

[1] http://duse.sf.net

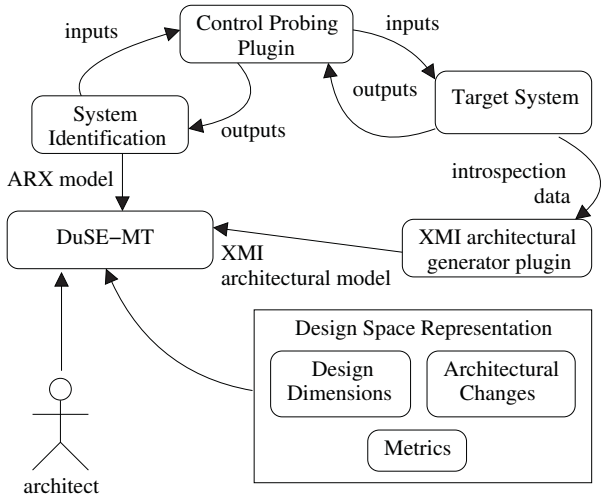


Figure 1. DuSE overview

The target system's architectural model is an UML representation annotated with a specific UML profile which enables the identification of available control parameters and performance variables. Such an UML model provides the means to accomplish automatic black-box system identification, yielding the ARX model which supports the use of self-adaptation quality metrics to guide navigation through design space.

We have implemented the proposed platform on top of the *Qt Modeling Framework*[2] as a multi-platform tool with a flexible architecture which enables its use in a range of technologies for the target system. The quality and feasibility of the architectural designs resulting from design space navigation have been evaluated in a case study undertaken on top of CIAO (*Component Integrated ACE ORB*) middleware.

The remainder of this paper is organized as follows. Section 2 presents the most prevalent design dimensions when using feedback control for achieving self-adaptation. Section 3 describes the proposed approach, supporting technologies, and implementation aspects. The evaluation of our approach when supporting reasoning about self-adaptation design alternatives for a distributed system based on CIAO is presented in section 4. Section 5 provides pointers to related work and section 6 draws some conclusions and presents future work.

2. Design Dimensions in Self-Adaptive Systems

Making design decisions concerned to system self-awareness, environment monitoring, adaptation strategies, and enacting mechanisms constitutes a central activity when developing self-adaptive systems. Preliminary studies about design dimensions for self-adaptive systems [1, 5, 7, 14] have identified degrees of freedom related to run-time system representation, system/environment observation, control strategies, adaptation identification, and adaptation mechanisms.

Whilst defining a comprehensive set of design concerns is paramount to further understand solution opportunities and analyze trade-offs, in this work we focus on sub-dimensions related to control strategies, control schema, and their impacts on overall system quality attributes. When dealing with such control aspects, architects should make decisions, for instance, about target system modeling (e.g., first-principles vs black-box models), control strategy (e.g., fixed-gain PID, adaptive model identification, model predictive), and control schema (e.g., single loop, nested control, hierarchical control, decentralized control) [10].

Although any guideline for selecting an effective feedback control solution from that design space would still require a careful assessment against local domain/requirements drivers, we believe the use of flexible and powerful tools to support trade-off analysis of alternative control designs may further advance the use of effective model-based approaches.

3. The Proposed Approach

The feedback control architectural analysis environment we propose defines a model-based meta-architecture which includes: i) a self-adaptation design space representation; ii) changes to be enacted in target system's original model when navigating through the design space; iii) OCL rules which define valid variation points in each dimension; and iv) a set of metrics to guide architects during design trade-off analysis.

As depicted in Figure 1, DuSE provides a technology-independent framework for reasoning about feedback-control approaches by combining run-time target system dynamics (represented by a first-order ARX model), target system's structural architecture XMI descriptions, and a design space representation.

A design space [4, 17] is represented as a tuple DS= $<DD, M>$, where *DD* is a set of design dimensions and *M* is a set of metrics defined for that design space. A design dimension DD_i is a set of variation points VP_{ij}=$<CT_{ij}, CH_{ij}>$, where CT_{ij} is a set of OCL constraints that must be satisfied in the target system's original architecture model in order to that design dimension point be available for use, and CH_{ij} is a set of changes to be enacted in the target system's original model SM_0.

Therefore, a specific design space location $<VP_{1i}$, $VP_{2j}, ..., VP_{|DD|m}>$ is only defined for SM_0 if SM_0 satisfies the set of constraints $CT_{1i} \cup CT_{2j} \cup ... \cup CT_{|DD|m}$. Likewise, the changes to be enacted in the original model SM_0 for a given design space location $<VP_{1i}, VP_{2j}, ..., VP_{|DD|m}>$ are the merge of all architectural contributions from each design dimension location: $CH_{1i} \cup CH_{2j} \cup ... \cup CH_{|DD|m}$. We denote by

[2] http://qt-project.org/wiki/QtModeling

Design Dimension	Variation Points	OCL Constraints	Architectural Changes
(DD_1) Control Adaptiveness: refers to the moment (design-time, run-time with predefined alternatives, run-time with full control reconfiguration) in which control tuning is undertaken	(VP_{11}) Fixed Gain: PID control with pre-defined gain values	(CT_{111}) *allOwnedElements()-> selectAsType(Component).provided-> exists(extension_MonitorInterface->notEmpty())* (CT_{112}) *allOwnedElements()-> selectAsType(Component).provided-> exists(extension_ControlInterface->notEmpty())*	(CH_{111}) *nPack.addOwnedType(s = sensorFactory("QPSensor", target.mInterface()));* (CH_{112}) *nPack.addOwnedType(a = actuatorFactory("QPActuator", target.cInterface()));* (CH_{113}) *nPack.addOwnedType(c = controllerFactory("QPIDController", s, a));*
	(VP_{12}) Model-Identification Adaptive Control (MIAC): on-line model estimation and control tuning	(CT_{121}) as defined in (CT_{111}) (CT_{122}) as defined in (CT_{112})	(CH_{121}), (CH_{122}) and (CH_{123}) as defined in (CH_{111}), (CH_{112}) e (CH_{113}) (CH_{124}) *nPack.addOwnedType(e = new QARXEstimator(s, a));* (CH_{125}) *nPack.addOwnedType(t = new QPIDPolePlacement(c, e, st));*
(DD_2) Control Nesting Degree: the number of nested control loops applied to the target system	(VP_{21}) Single loop	(CT_{211}) as defined in (CT_{111}) (CT_{212}) as defined in (CT_{112})	<no changes required>
	...		
	(VP_{2d}) *d* nested loops	(CT_{2d1}) *allOwnedElements()-> selectAsType(Component)-> select(shortestControllableChain()->size()=d)* (CT_{2d2}) post: *target.loop.stime() > target.longestControllableChain()-> collect(stime())->sum()*	(CH_{2d1}) *var i = 0;* *var currentTarget = lowestTarget;* *while(i < d) {* *instantiateControlLoop(currentTarget);* *currentTarget = currentTarget-> nestingControllableComponent();* *++i;* *}*

Table 1. Feedback control design space dimensions

$SM_{\{i,j,...,m\}}$ the architectural model resulting after applying contributions from all specific locations in each design space dimension. A metric M_i is a tuple $<ME_i, MG_i>$, where ME_i is the metric evaluation expression and MG_i describes if the metric is intended to be maximized (1) or minimized (-1). Metrics M_1 and M_3 can be evaluated at any design space location defined for the original SM_0 model. Metric M_2 is undefined when DD_1 component is VP_{12} since settling time information is not available off-line.

3.1. Analysing Feedback Control Strategies

In order to support early reasoning and analysis of feedback control approaches for self-adaptation, we have instantiated our design space model in terms of design dimensions, architectural changes, metrics, and constraints directly related to most prominent feedback control concerns.

As presented in Table 1, we have defined two initial design dimensions related to feedback control loops in self-adaptation. Dimension DD_1 captures the robustness of feedback loops and provides two variation points: i) VP_{11}: fixed-gain, which uses a set of pre-defined parameters systematically derived to work on a specific operating region; and ii) VP_{12}: MIAC, a sort of adaptive control that continuously identifies the target system and reconfigures control parameters. Deciding about such dimension involves a trade-off analysis between robustness vs control overhead. Dimension DD_2 represents the nesting degree of feedback loops. Hierarchically interacting loops have successfully been applied to support self-management in different scopes, simplifying the handling of non-linearities and widening control possibilities. On the other hand, hierarchical feedback control requires a more careful analysis so that the time-scales of loops at different levels do not intervene each other and control scalability is kept at accepted levels. The corresponding OCL rules and architecture changes are also presented in Table 1.

Table 2 presents the defined architectural metrics. Metric M_1 captures how much of control adaptiveness exists in the system. A highly control-adaptive system exhibits higher robustness, with the cost of increasing control overhead due to continuous system identification and control tuning.

Metric M_2 estimates high discrepancies in hierarchical loops. In order to prevent control interference, hierarchical loops should be time-separated and the outer loop time-scale is usually dominated by the slower inner loop stabilization time. M_2 measures the highest difference between the slower and faster inner loops at level *d*. Levels with high values in M_2 should be further designed if faster self-adaptation is to be achieved.

Finally, M_3 measures how much control operation depends upon a single top-level controller. High values for M_3 indicates better control scalability although probably demanding a more complex control programming model. A design solution is a function that maps each lowest-level

Description	Evaluation Expression	Goodness Factor
(M_1) Control Overhead	$ME_1 = \frac{allOwnedElements()->selectAsType(QARXEstimator)->size()}{allOwnedElements()->selectAsType(QPIDController)->size()}$	$MG_1 = -1$
(M_2) Maximum Settling Time Difference at level *l*	$ME_2 = cll->product(cll)->iterate(t:Tuple(first:QPIDController, second:QPIDController); maxDiff:Integer=0 \mid maxDiff.max((first.stime()-second.stime()).abs()))$; $where\ cll = allOwnedElements()->selectAsType(QPIDController)->select(allNestingLoops()->size()=l)$; $QPIDController::stime() = -4/log(max_i\|p_i\|)$; $max_i\|p_i\|$ is the magnitude of the largest closed-loop pole	$MG_2 = -1$
(M_3) Control Decentralization Degree	$ME_3 = cl->select(nestingLoops()->size()=0)->iterate(c:QPIDController; minDepth:Integer=1 \mid minDepth.min(1/[c.allNestedLoops()->size()+1]))$; $where\ cl = allOwnedElements()->selectAsType(QPIDController)$	$MG_3 = 1$

Table 2. Feedback control design space metrics

controllable component into a location in the proposed design space.

4. Case Study

The definition of new design spaces, its dimensions, variation points and corresponding OCL constraints and metrics, as well as the analysis of feedback control approaches against a specific target system model are currently available in DuSE-MT - our supporting tool. DuSE-MT supports the workflow depicted in Figure 1 by defining a flexible architecture which enables the use of connector plugins for system identification in a variety of platforms. Each connector plugin enables the probing of target systems developed for that specific platform, gathering input/output relationships between system's controlled/measured parameters and allowing off-line and on-line system identification. DuSE-MT currently provides connector plugins for the CIAO and Qt/DBus technologies.

A case study aimed at analyzing feedback control loops in a CORBA-based data acquisition service has been undertaken. The ARCOS platform [2] is a flexible component-based framework for industrial data acquisition, implemented on top of CIAO middleware. This case study investigated alternative feedback control approaches aimed at controlling service response time in the presence of disturbances, by adjusting three component parameters: thread pool's size, data cache size, and thread pool priority.

Figure 2 illustrates the initial non-adaptive architectural model provided as input in the case study. The case study's goal was to investigate to which extent our platform helps the architect when choosing among different control strategies and provides insights for trade-off analysis.

For that purpose, a running instance of the data acquisition server was probed by DuSE-MT and ARX models for the three controllable components (`ThreadPool`, `CacheManager`, and `DAISLeaderFollowers`) have been generated. In that particular scenario there are [2 (#VP_1) x 2 (#VP_2; since d=2 is the longest chain of nested controllable components)] ^ 2 (number of lowest-level controllable components) = 16 different feedback control strategies available. Each of those solutions exhibits different values for the defined self-adaptation metrics and, therefore, favors different quality attributes. For those cases where a running instance/prototype of the original non-adaptive system is not available, metrics *M1* and *M3* still provide valuable guidance when choosing among alternative candidate architectures.

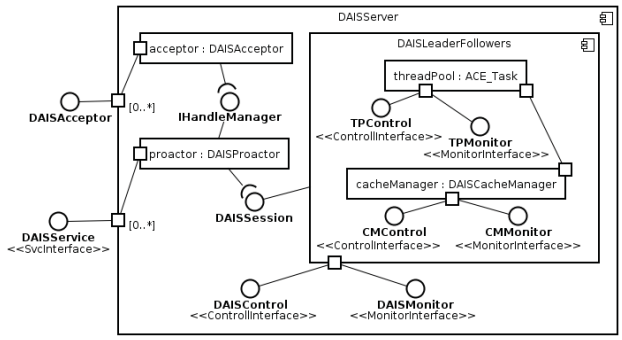
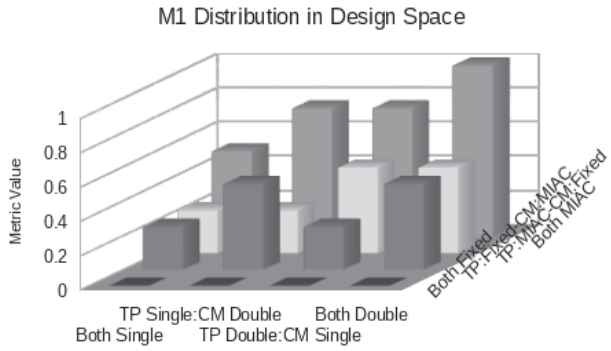
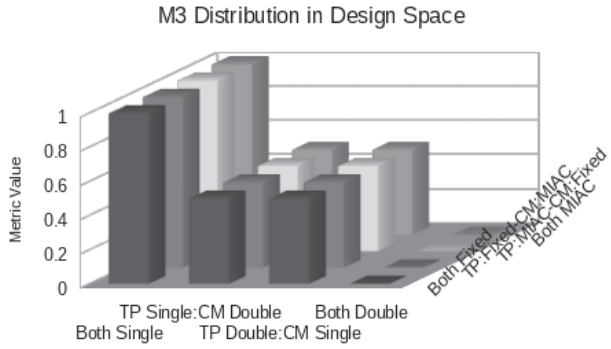


Figure 2. Initial (non-adaptive) DAIS server architectural model

Figure 3 shows the values for metrics *M1* and *M3* in the case study. Whilst obviously there may exist no single solution which fully maximizes all involved metrics, the obtained results may serve as helpful subsidies to support and well-inform the adoption of a specific feedback control architec-



(a)



(b)

Figure 3. Metrics M1 and M3 in case study

ture in detriment of alternative designs. For example, if control overhead is a prominent concern, perhaps because of target system deployments in embedded resource-restrictive platforms, then architects could use *M1* distribution as a guide to choose feedback schemas with associated low overhead implications. As a side effect, the target system must be carefully identified and operating environment should not be amenable to non-linearities and uncertainties. The analysis environment presented in this work provides the means for a more quantitative and interactive investigation of such design trade-offs.

5. Related Work

Over the past years some approaches for early architectural reasoning by explicit modeling of control loops have been proposed. In [20], a reference model for self-adaptation (FORMS) is presented. FORMS provides an unified view which integrates perspectives from reflexive, distributed computing, and MAPE-K [15] technologies. FORMS elements are described in Z specification language, which enables the formal reasoning of self-adaptation properties.

The use of megamodels at run-time to describe self-adaptation behavior is presented in [19]. The proposed notation entails the definition of multiple interacting feedback loops and relies on a model interpreter to dynamically adjust the adaptation logic. An UML Profile for feedback control modeling is presented in [9], along with a set of well-formedness rules to validate control schemas. Guidelines for modeling of multiple loops and elicitation of loop interferences are also presented. Actor-based approaches [12] and notations for self-adaptation with multiple objectives [6] have also been proposed.

Although some of the aforementioned proposals - such as FORMS - provide expressive and rigorous notations for feedback control modeling, they are mostly based on non-standardized and/or low-parsimony languages, provide no tool support, and still heavily depends on architect's skills, as a consequence of the lack of explicit design space representations.

Our work tries to overcome some of these shortcomings by rather proposing an architectural analysis environment based on MOF (*Meta Object Facility*) and UML (*Unified Modeling Language*) standards, an explicit domain-specific design space representation, and a set of effective metrics to assess adaptation-related architectural quality attributes. Finally, we observe that our tool leverages rapid modeling/analysis and the proposed design space representation is also liable to be applied in other application domains since we aim at a proper balance between modeling notation generality and expressiveness.

6. Conclusion and Future Work

The design and development of large-scale distributed systems with flexible and robust self-adaptation capabilities have become a promising approach to cope with continuous increases in system complexity. Furthermore, stringent demands to provide quality of service assurances in unpredictable and uncertain environments introduce additional challenges in such scenarios.

In this paper, we have presented DuSE - a flexible analysis environment for representing and comparing alternative architectural design choices related to feedback control approaches to self-adaptation. We have described the underlying design space model representation and its instantiation devoted to support early reasoning of architectural trade-offs regarding control loop robustness and interaction.

Our reference implementation, DuSE-MT, integrates the mechanisms we have proposed and allows for rapid modeling and analysis of self-adaptation scenarios. While our preliminary experience with DuSE indicates useful reasoning support, several avenues of future work may be identified.

We are currently performing experiments with larger design spaces and metrics sets in order to assess DuSE scalability aspects when dealing with large-scale models. DuSE

has currently no automatic support for handling conflicting design goals and deciding between metric satisfaction trade-offs. The use of search-based optimization approaches to find out a Pareto-optimal set of candidate architectures is also currently being investigated. The definition of annotated design space navigation traces to document design rationale and support for sharing models and design spaces over a network are also currently being considered.

References

[1] Jesper Andersson, Rogério Lemos, Sam Malek, and Danny Weyns. Software engineering for self-adaptive systems. chapter Modeling Dimensions of Self-Adaptive Software Systems, pages 27–47. Springer-Verlag, Berlin, Heidelberg, 2009.

[2] Sandro Santos Andrade and Raimundo José de Araújo Macêdo. A non-intrusive component-based approach for deploying unanticipated self-management behaviour. In *Proceedings of IEEE ICSE 2009 Workshop Software Engineering for Adaptive and Self-Managing Systems*, May 2009.

[3] Muhammad Ali Babar, Torgeir Dingsyr, Patricia Lago, and Hans van Vliet. *Software Architecture Knowledge Management: Theory and Practice*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[4] Frederick P. Brooks. *The Design of Design: Essays from a Computer Scientist*. Addison-Wesley Professional, 1st edition, 2010.

[5] Yuriy Brun, Giovanna Marzo Serugendo, Cristina Gacek, Holger Giese, Holger Kienle, Marin Litoiu, Hausi Müller, Mauro Pezzè, and Mary Shaw. Software engineering for self-adaptive systems. chapter Engineering Self-Adaptive Systems through Feedback Loops, pages 48–70. Springer-Verlag, Berlin, Heidelberg, 2009.

[6] Shang-Wen Cheng, David Garlan, and Bradley Schmerl. Architecture-based self-adaptation in the presence of multiple objectives. In *Proceedings of the 2006 International Workshop on Self-Adaptation and Self-Managing Systems*, SEAMS '06, pages 2–8, New York, NY, USA, 2006. ACM.

[7] R. de Lemos et al. Software Engineering for Self-Adaptive Systems: A second Research Roadmap. In R. de Lemos, H. Giese, H. Müller, and M. Shaw, editors, *Software Engineering for Self-Adaptive Systems*, number 10431 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany., 2011. Schloss Dagstuhl. Leibniz-Zentrum fuer Informatik, Germany. Full citation: http://dx.doi.org/10.1007/978-3-642-02161-9_1.

[8] Shirley Gregor. The nature of theory in information systems. *MIS (Management Information Systems Research Center) Quarterly*, 30(3):611–642, September 2006.

[9] Regina Hebig, Holger Giese, and Basil Becker. Making control loops explicit when architecting self-adaptive systems. In *Proceedings of the 2nd International Workshop on Self-Organizing Architectures*, SOAR '10, pages 21–28, New York, NY, USA, 2010. ACM.

[10] Joseph L. Hellerstein, Yixin Diao, Sujay Parekh, and Dawn M. Tilbury. *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.

[11] David Jones and Shirley Gregor. The anatomy of a design theory. *Journal of the Association for Information Systems*, 8(5), 2008.

[12] Filip Křikava, Philippe Collet, and Robert B. France. Actor-based runtime model of adaptable feedback control loops. In *Proceedings of the 7th Workshop on Models@run.time*, MRT '12, pages 39–44, New York, NY, USA, 2012. ACM.

[13] Hausi Müller, Mauro Pezzè, and Mary Shaw. Visibility of control in adaptive systems. In *Proceedings of the 2nd international workshop on Ultra-large-scale software-intensive systems*, ULSSIS '08, pages 23–26, New York, NY, USA, 2008. ACM.

[14] T. Patikirikorala, A. Colman, Jun Han, and Liuping Wang. A systematic survey on the design of self-adaptive software systems using control engineering approaches. In *Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2012 ICSE Workshop on*, pages 33 –42, june 2012.

[15] Mazeiar Salehie and Ladan Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems*, 4(2):14:1–14:42, May 2009.

[16] Mary Shaw. Research toward an engineering discipline for software. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*, FoSER '10, pages 337–342, New York, NY, USA, 2010. ACM.

[17] Mary Shaw. The role of design spaces. *IEEE Software*, 29(1):46–50, January 2012.

[18] Dag I. K Sjøberg, Tore Dybå, Bente Cecilie Dahlum Anda, and Jo Erskine Hannay. *Building Theories in Software Engineering*, chapter 12, pages 312–336. Springer-Verlag London, 2008.

[19] Thomas Vogel and Holger Giese. A language for feedback loops in self-adaptive systems: Executable runtime megamodels. In *Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2012)*, pages 129–138. IEEE Computer Society, 6 2012.

[20] Danny Weyns, Sam Malek, and Jesper Andersson. Forms: Unifying reference model for formal specification of distributed self-adaptive systems. *ACM Transactions on Autonomous and Adaptive Systems*, 7(1):8:1–8:61, May 2012.

Towards Coupled Evolution of Metamodels, Models, Graph-Based Transformations and Traceability Links

Chessman Corrêa Toacy Oliveira Cláudia Werner

Federal University of Rio de Janeiro
COPPE – System Eng. and Computer Science
Rio de Janeiro, Brazil
{chessman,toacy,werner}@cos.ufrj.br

***Abstract* — Model-Driven Development (MDD) approaches use metamodels, models and transformations specifications as the main artifacts for software development. Metamodels are used as blueprints for models and transformation specifications, creating a conformance dependency between them. Moreover, models can be generated from other models through transformations, which also creates dependencies. In this scenario, changes in one artifact can propagate to others creating a set of cascading fixes that might overwhelm software developers. In this work we propose a coupled evolution approach that encompasses metamodels, transformation specifications, models and traceability links. The conformance of transformation specifications and models with metamodels is supported by automatic transformation. However, differently from other approaches, the same transformation is used to recover the conformance of models and transformation specifications.**

Keywords- model-driven development; evolution; migration; synchronization, maintenance.

I. INTRODUCTION

Model-Driven Development (MDD) [1] aims at using models as the main development artifacts, transformations to automate the repetitive tasks related to artifacts creation [2] and traceability links to represent dependencies between elements from different models [3]. In MDD approaches based on the principle that "everything is a model" [4], transformation specifications and traceability links are also models [5-6].

The use of models, automatic transformations and traceability links requires formal specifications [7], typically represented as a metamodel [8-9]. A metamodel describes model elements, their properties, relationships between elements and constraints. Metamodels are also used in transformation mappings (or rules) for reading source models and generating the elements for target models. Metamodels are defined by a meta-metamodel, such as MOF [10] and Ecore [8].

The creation of artifacts from others establishes a dependency relationship between their elements. Therefore, there are dependencies between (1) metamodels and models (including transformation specifications and traceability links) and (2) between interrelated models, transformation specifications and traceability links.

A common scenario in software development is software evolution. Software evolution is achieved by adapting development artifacts to accommodate new functionalities or to fix defects. Due to existing dependencies, an artifact may need to be modified when an interrelated artifact is changed. Therefore, if a metamodel is modified, models and transformation specifications must be migrated (vertical co-evolution). It means that new versions of these artifacts have to be created, non changed elements must be copied, and corrections must be done to make them in conformance with the new version of the metamodel. Moreover, if models or transformation specifications are changed, other interrelated models and traceability links must be synchronized to conform to the new version of the changed model or transformation specification (horizontal co-evolution), i.e., it is necessary to create correspondent elements in interrelated artifacts according to the transformation rules.

The dependent (or coupled) evolution of metamodels, models and transformation specification is a known problem. Solutions for the co-evolution of metamodel and model were proposed in [11-15]. Solutions for the co-evolution of metamodel and transformation specifications can be found in [16-18]. A broader approach which can lead to metamodel/model and metamodel/transformation co-evolution is proposed in [19]. Although these works are related to vertical co-evolution, none of them considers the integrated coupled evolution of metamodel/model/transformation specifications. Moreover, in a real MDD scenario, if model or transformation specifications are changed, it is necessary to synchronize interrelated models and update traceability links.

The main contributions of this paper are the integration of vertical and horizontal co-evolution and the use of correspondences between the elements of two metamodel versions to migrate transformation specifications and models. Moreover, we believe that horizontal co-evolution can reduce the effort to perform vertical co-evolution in cases when synchronization can be executed to complete elements.

The remaining of this paper is organized as follows. In Section 2, dependencies and coupled evolution are discussed. In Section 3, we present the proposed solution and an example. Related works are discussed in Section 4 and conclusions are presented in Section 5.

II. CHANGES AND CO-EVOLUTION

The co-evolution of artifacts depends on the type of changes. A change classification for metamodel changes proposed by Gruschko et al. [13] was expanded to include transformation specifications and interrelated models [20]. **Non-breaking changes** (NBC) refer to changes that do not break consistency. Therefore, no change propagation is necessary. **Breaking but resolvable changes** (BRC) break consistency but these can be automatically solved. **Breaking and unresolvable changes** (BUC) also break consistency, but it is not possible to automatically resolve the problem, which means that the developer needs to make the corrections manually.

A. Metamodel Changes

The inclusion of any non-mandatory element in a metamodel is NBC. Therefore models and transformation specification don't need to be updated when a new optional metaclass (define the instance of a model element) or attribute (define a property for a metaclass instance) is included in the metamodel. On the other hand, if a new metaclass or attribute is mandatory, then changes are needed in both artifacts. In this case, transformation specifications should be modified to generate the mandatory metaclass instances or to define values for the corresponding property. Moreover, the mandatory instances must be created and property values must be defined for models. At a first glance, these changes are BUC, since the developer needs to create metaclass instances, resolve associations, define the correct values for new properties and create transformation mappings to generate mandatory metaclass instances or define the values for not null properties. However, if the creation of metaclass instances or assignment of property values can be performed through synchronization, BUCs can turn to BRCs. The most common synchronization approach is to execute again the transformation specification. Hence, if there exists a transformation mapping to generate an instance of a mandatory new metaclass or define the value of a property, the developer does not have to perform this task manually. However, if the model is not generated from another, the change remains BUC.

The main metamodel updates are renaming and modification of the lower bound of attributes or references. The rename operation is BRC, since it can be performed automatically to update models and transformation specifications. The change of a metaclass or attribute to non-mandatory is NBC (lower bound is reduced to zero); and the opposite is BUC (lower bound is increased). In the last case, one or more values have to be defined for the property corresponding to the changed attribute or new instances of the referenced metaclass have to be created (or associated). Transformation mappings also have to be modified to generate the correct property values or metaclass instances.

The exclusion of metamodel elements is BRC. If a literal is removed from an enumeration, it has to be replaced by another in the models and transformation specifications. The exclusion of an attribute means that the correspondent property values have to be removed from models. Moreover, value assignments and guard conditions associated with transformation mappings have to be deleted from transformation specifications. If a metaclass is removed, the corresponding instances have to be excluded from models and transformation mappings have to be removed from transformation specifications.

B. Model Changes

Models have to be consistent with interrelated models. When a transformation is used, these models must also be in conformance with the transformation specification. Moreover, traceability links must reference valid model elements and transformation mappings, if one of the elements has been generated by transformation.

The change operations that can be applied to models are inclusion and exclusion of metaclass instances and the change of properties values. Changes are NBC when applied to specific elements of a model. If a modified element is related to an element of another model and can be generated by automatic transformation, changes on this element are BRC since interrelated elements can be automatically synchronized and traceability links can be generated through the execution of the transformation. On the other hand, if the element is not derivable through automatic transformation, the changes are BUC, which means that elements of dependent models and traceability links will have to be updated manually. It is worth mentioning that the exclusion of a metaclass instance can be BRC even if there is no transformation mapping to be used during synchronization. This is because traceability links can be used to automatically identify interrelated elements that should be deleted.

C. Transformation Specification Changes

Transformation specifications may need changes for error corrections, refactoring, technology evolution or metamodel changes. The inclusion, update and exclusion of transformation mappings are BRC in relation to models, since the execution of the new version of the transformation specification allows the synchronization of models with it. It is worth mentioning that the inclusion of transformation mappings can result in the creation of metaclass instances and traceability links. In the same way, the exclusion of transformation mappings results in the exclusion of metaclass instances and traceability links.

The rename is the change operation that can be applied to the transformation mapping. This is BRC, since traceability links can be automatically corrected to reference the new transformation mapping name.

III. THE PROPOSED APPROACH

The proposed approach is based on Software Configuration Management principles and tools [21] to control the evolution of MDD artifacts in time and space. The evolution in time is controlled by the preservation of the change history of metamodels, transformation specifications and models, as well as maintaining existing traceability links between elements of

different model versions. The evolution in space is controlled by migration (vertical co-evolution) and synchronization (horizontal co-evolution). Due to space limitations, only evolution in space is considered in this paper. The example used to explain the proposed approach is described in Section *A* and the approach is detailed in Section *B*.

A. Example

This example illustrates the generation of a database model from a domain model through a triple graph-based transformation specification [22]. A partial representation of the metamodel for domain models is shown in Figure 1a. *Entity*, *Attribute* and *Reference* are domain elements. Therefore they have a *name*. The *Entity* element represents a concept from the problem domain; *Attribute* represents a characteristic on an *Entity*, and *Reference* defines a relationship among two entities. An entity can have many attributes. The first version of the database metamodel is presented in Figure 1b (partial view). The *Relation* and *Field* are database elements (*DbElement*). Therefore they also have a name. A relation can have one or more fields.

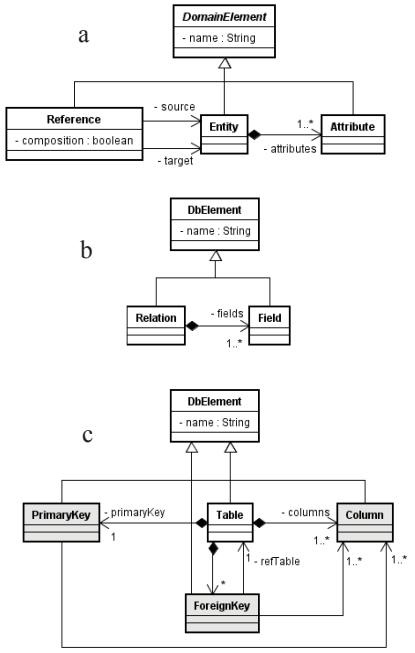


Figure 1. Metamodel for domain models.

The transformation specification[1] to generate a database model from a domain model is shown in Figure 2. This transformation has two rules: one for creating relations and identification fields from entities (rule 1) and other for creating fields from attributes (rule 2). These rules are executed to generate the elements of the database model from the domain elements and vice-versa (graph-based transformations were conceived to be bidirectional). The elements on the left hand side (LHS) reference the *Entity* and *Attribute* elements on the domain metamodel. The elements of the right hand side (RHS) reference the *Relation* and *Field* elements of the database metamodel. The middle elements are the corresponding mappings. Therefore, the *e2r* mapping defines the generation of relations from entities (forward engineering) and entities from relations (reverse engineering). In the same way, *a2f* mapping defines the generation of fields from attributes and attributes from fields. The composition relationship between *Relation* and *Field* means that *Field* instances is associated with a *Relation* instance. This is the same for attribute instances. The composition with the stereotype *create* among *Relation* and *Field* in *Rule 1* means that an identifier field is created to a table.

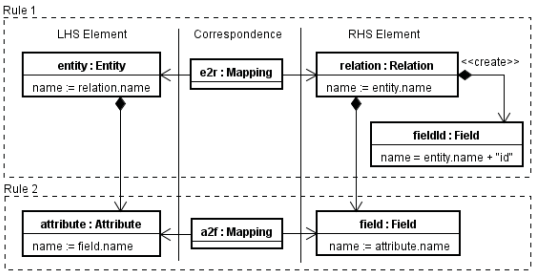


Figure 2. Transformation specification.

The domain model for the example is shown in Figure 1a. The model contains the following *Entity* instances: *Client*, *Sale*, *Product* and *Item*. The execution of the transformation specification results in the database model shown in Figure 3. Some fields were created manually. The dashed arrow lines represent traceability links between the correspondent elements of the domain and database model generated during the transformation.

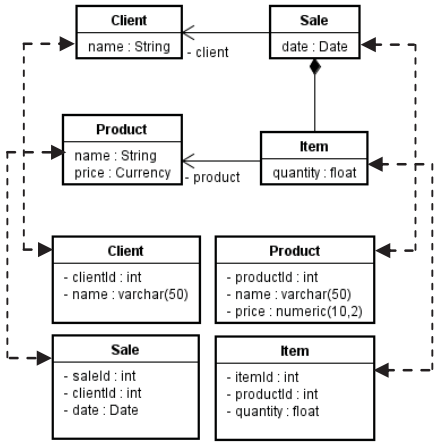


Figure 3. First version of the domain and database models.

After the generation of the database model, the metamodel is modified (Figure 1c).

B. Description of the Approach

When a new version of a metamodel is released, the two-model comparison algorithm of EMF Compare [23] is used to create a model that records the matching and differences (inclusions, updates and exclusions) of the two versions of the

[1] The notation was adapted from triple graph-based transformation specifications defined for some known tools.

metamodel. The matching algorithm was adapted to use versioning information to match elements. In our approach, each element of a metamodel, transformation specification or a model is a configuration item identified by an *id*. The *ids* are kept together with their model elements (as well as other versioning data) and are preserved if elements are renamed, allowing precise matching of existing elements. In the example, *Relation* was renamed to *Table*, but they are matched because their *id* is the same.

Following the example, the changes described in Table I are identified. The second step is to classify the change operations as NBC, BRC and BUC (Table I).

TABLE I. DATABASE METAMODEL CHANGES

Operation	Classifi-cation	Correction Needed	Correspond-ence (↔)
Relation is renamed to Table	BRC	Yes	Relation ↔ Table
Field is deleted	BRC	Yes	-
Column is created do replace Field	BUC*	Yes	Field ↔ Column
PrimaryKey is created to define the primary key of the tab (in this example, primary keys are mandatory)	BUC*	Yes	Field ↔ PrimaryKey
ForeignKey is created to define optional foreign keys	NBC**	-	Field ↔ ForeignKey

**Becomes BRC after the creation of the migration transformation.*
***Becomes BRC in relation to the domain model after the new version of the transformation specification is ready.*

After the classification, the last version of transformation specifications and models that depend on the metamodel are analyzed. The objective is to verify if these artifacts really have the identified breaking changes. For example, a model may not have instances of a metaclass deleted from the metamodel and the transformation specification may not have transformation mappings related to the deleted metaclass. Existing breaking changes are recorded as *correction needed*. The objective of this task is to reduce the effort necessary to update transformation specifications and migrating models, since only existing breaking changes will be considered. In the example, the transformation specification has mappings and the database model has instances of the changed metaclasses. Therefore, these artifacts need corrections due to the breaking changes identified, as specified in the column *Correction Needed* of Table I.

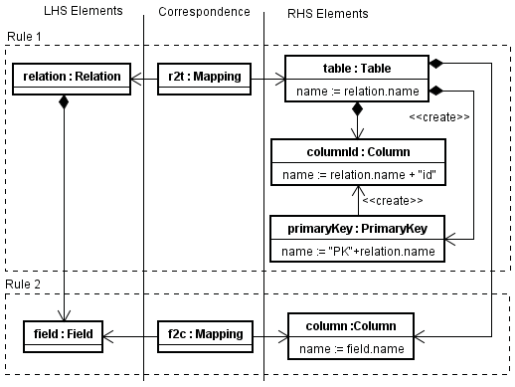


Figure 4. Migration specification.

The next step is the creation of the correspondence model (Figure 4) using the same language of the transformation specification. This model is the migration specification to be executed for migrating transformation specifications and models. The mappings of this model relate elements of the previous and new releases of the database metamodel. The mappings are created based on the *matching and difference model* created by EMF Compare.

The correspondence mappings between elements that have not been changed are generated automatically. This results in a simple copy of non-modified elements. Correspondence mappings are also created for identifiable BRC. In the example, a correspondence mapping is created to generate tables from relations (Rule 1 in Figure 4). However, correspondence mappings must be manually defined for BRCs that cannot be automatically identified and BUCs. This is the case of *Column* and *PrimaryKey*. Although *Column* was created to replace *Field*, the second was deleted and the first was included. Therefore, *Column* has no versioning data to be used for associating it with *Field*, as happens with the *Relation*. Therefore, the correspondence mapping must be created manually (Rule 2). *PrimaryKey* is a new element. Hence, there is no transformation mapping related to this element and no instance of this metaclass in the database model. However, since this element is mandatory and the database model is generated from the domain model, the transformation specifications need a mapping to generate this element and the database model needs an instance of it. Due to that, a correspondence mapping is necessary to generate instances of this element during migration. In the example, Rule 1 was hand complete to generate the primary key and the corresponding column. Since *ForeignKey* is optional, a migration mapping is not necessary.

Worth to mention that although the creation of *PrimaryKey* is BUC, the migration mapping generates complete instances, including the necessary references to other elements. Therefore, this change becomes BRC in relation to models.

Next, the transformation specifications need to be migrated. The migration is necessary because the breaking changes cannot be read when the transformation specification is opened. Hence, our approach is to automatically create a new transformation specification and copy all mappings, making the necessary corrections during the copy, according to the correspondence (migration) mapping. Our strategy to migrating transformation specification is based on the mathematical transitive principle (if A = B and B = C then A = C). This approach ensures that the transformation specification will generate the same model after migration. Going back to the example, *e2r* mapping (Figure 2) associates *Entity* (A) to *Relation* (B) and *r2t* mapping (Figure 4) associates *Relation* to *Table* (C). Hence, the RHS of *Rule 1* of the transformation specification (Figure 2) is replaced by the RHS of the migration transformation (Figure 4). The correction of the assignments is performed according to the same principle. For example, the assignment *name := relation.name* of the LRS *Entity* of Rule 1 (Figure 2) is replaced by *name := table.name* (it is realized only when there are correspondent properties). The same is done for Rule 2 (*a2f* mapping - Figure 2). It is important to mention that, at first glance, *a2f* mapping should be deleted, since *Field* has been removed from the metamodel. However, since *Column*

corresponds to *Field*, the mapping *a2f* mapping can be reused. During this process, new traceability links are generated to relate the new instances with the metaclass instances of the domain model.

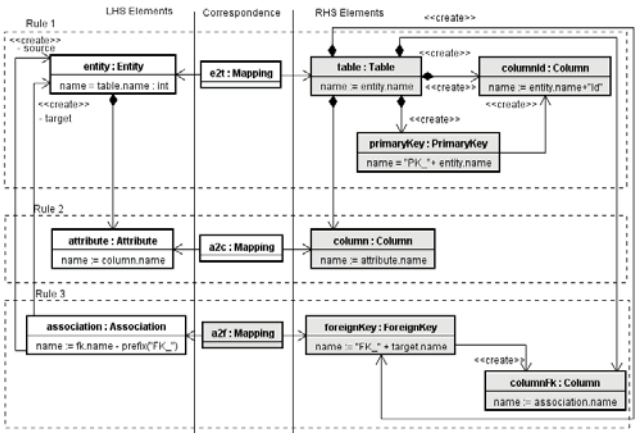


Figure 5. New version of the transformation specification.

After the generation of the new transformation specification, the mapping to generate the *ForeignKey* instance needs to be created manually. This turns the change as BRC, since it breaks the consistency of the database model with the transformation specification. The new version of the transformation specification is shown in Figure 5 (the changes are in gray).

Next, the model is migrated through the execution of the migration transformation. It results in the copy of the instances of the metamodel elements that have not been modified and the generation of new instances. This results in the replacement of *Relation* by *Table* and *Field* by *Column*. Moreover, primary keys are created and linked with the identification columns. At his moment, the database model is in conformance with the metamodel. However, a new rule (Rule 3 in Figure 5) has been included in the transformation mapping. Therefore, the database model is not consistent with the transformation specification. Hence, synchronization is necessary.

The synchronization process is based on the execution of the transformation specification. In the example, the second version is executed. Since the RHS metamodel has been modified (database metamodel), the transformation is performed from left to right (forward engineering). This results in the creation of the foreign keys that didn´t exist in the first version of the database model and the correspondent traceability links. After the synchronization, the database model becomes consistent in relation to the transformation specification. At this moment, the project is completely consistent.

IV. Related Work

The co-evolution of metamodels, models and transformation specification is a known research problem. The simplest approach is to create all the migration mappings manually [24]. However, this solution is tedious and mistakes can be made. Therefore, automatic co-evolution is necessary to reduce the effort and errors. Most of the works goes into this direction [11-15, 19], but considerable attention has been given to automate metamodel and model co-evolution [11-15]. However, transformation specification must also be considered. In [19] migration strategies (migration specifications) can be created for co-evolution of models and transformation specifications, but it is necessary to create distinct migration strategies for models and transformation specifications. In [18], correspondences are defined among two versions of transformation specifications. In the approach proposed in this paper, the same correspondence mapping is used to perform the migration of both artifacts.

In real MDD scenarios, models are generated from others. Therefore, the migration of models and transformation specifications are not enough to keep the consistency in MDD projects. Hence, synchronization is necessary after migration to keep consistency among models and transformation specifications and to keep traceability links up to date. Moreover, some BUCs can be solved during synchronization, reducing the effort to keep consistency in MDD projects. This is one of the objectives of our approach and works encompassing migration and synchronization together have not been found.

Our approach has some similarities to [13] since the change classification proposed by them are used in this work. Moreover, some of the tasks are the same. However, our approach has the following differences: (1) in [13], the changes are recorded in real time and the differences are calculated from this record; in our approach the differences are calculated from the previous version of the metamodel; (2) our approach takes into consideration the co-evolution of transformation specifications; (3) we analyze transformation specifications and models to verify if these artifacts really have breaking changes; and (4) we use synchronization to solve BUCs whenever possible. The last two contribute to the reduction of vertical-co-evolution effort.

A similar co-evolution concern is related to database schema evolution [25-26]. The schema and the database correspond to the metamodel and model, respectively. However, the co-evolution of schema and databases is a kind of vertical co-evolution. There is no concern related to transformation specifications and interrelated models.

Although there are more sophisticated solutions for detecting differences between two versions of an artifact [27-29], in our approach, we are focused on matching elements and identifying model structural changes. In the same way, there are also more sophisticated approaches related to traceability link detection and recovering, such as in [30]. In our approach, the objective is to keep existing traceability links between evolved model elements and generating them for new elements.

V. Conclusions

Consistency between different kinds of models (including transformation specifications and traceability links) is essential to the success of model-driven projects. Therefore, if metamodel is evolved, models and transformation specifications must be migrated to be in conformance with the new version of the metamodel. However, since models have a dependency relationship with other models, changes must be propagated between interrelated models. Therefore, if a model is migrated, interrelated models must be synchronized. Otherwise, the migrated model will be in conformance with the metamodel but not with other models. Moreover, traceability

links must be up to date, since this resource is essential to keep the evolution of MDD artifacts.

In this paper we have presented an approach to keep the consistency between metamodels, transformation specifications and models and to keep traceability links up to date. The correspondence model used for model migration can also be used to partially update transformation specifications. Moreover, we have demonstrated through an example that synchronization can help to solve BUCs, thus reducing the effort to migrate models.

One key factor to reduce the effort is the verification if models and transformation specifications really have breaking changes. The objective is to avoid the creation of migration mappings for metamodel changes that don´t have impact on models and transformation specifications. We consider scenarios where development organizations have their own domain specific languages and control the evolution of their development artifacts.

As future work, we intend to apply our approach in real MDD projects that use transformation specifications based on triple graph grammars. Another future work is to compare our approach to others to evaluate the reduction of the effort necessary to preserve consistency due to the metamodel evolution. Moreover, we used Emorf [5] as the graph based transformation tool. We intend to generalize the transformation migration to migrate any graph-based transformation specification.

REFERENCES

[1] A. Kepple, J. Warmer, and W. Bast, *MDA Explained: The Model Driven Architecture: Practice and Promice*: Addison-Wesley, 2002.

[2] G. Deng, G. Lenz, and D. C. Schimidt, "Addressing Domain Evolution Chalenges in Software Product Lines," in *Satellite events at the MoDELS 2005 Conference*, Montego Bay, Jamaica, 2005, pp. 247-261.

[3] N. Anquetil, *et al.*, "Traceability for Model Driven, Software Product Line Engineering " in *ECMDA Traceability Workshop Proceedings*, Berlin, Germany, 2008, pp. 77-86.

[4] J. Bézivin, "On the Unification Power of Models," *Software and System Modeling (SoSyM)*, vol. 4, pp. 171-188, 2005.

[5] L. Klassen and R. Wagner, "EMorF - A tool for model transformation," in *Proceedings of the 7th International Workshop on Graph Based Tools (GraBaTs 2012)*, Bremen, Germany, 2012, pp. 1-6.

[6] E. Biermann, C. Ermel, J. Schmidt, and A. Warning, "Visual Modeling of Controlled EMF Model Transformations using Henshin," in *Proceedings of the 4th International Workshop on Graph Based Tools*, Enschede, The Netherlands, 2010, pp. 1-14.

[7] J. Greenfield, K. Short, S. Cook, S. Kent, and J. Crupi, *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*: Wiley, 2004.

[8] D. Steiberg, F. Budinsky, E. Merks, and M. Paternostro, *Eclipse Modeling Framework*, 2nd ed.: Addison Wesley, 2009.

[9] S. A. Ajila and S. Alam, "Using a formal language constructs for software model evolution," Berkeley, CA, 2009, pp. 390-395.

[10] OMG, "Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification - Version 1," Object Management Group (OMG)2008.

[11] A. Cicchetti, D. D. Ruscio, R. Eramo, and A. Pierantonio, "Automating Co-evolution in Model-Driven Engineering," in *Proceedings of the 2008 12th International IEEE Enterprise Distributed Object Computing Conference*, Munich, Germany, 2008, pp. 222-231.

[12] G. Wachsmuth, "Metamodel Adaptation and Model Co-adaptation," in *21st European Conference on Object Oriented Programming (ECOOP 2007)*, Berlin, Germany, 2007, pp. 600-624.

[13] B. Gruschko, D. S. Kolovos, and R. F. Paige, "Towards Synchronizing Models with Evolving Metamodels," in *Proceedings of the International Workshop on Model-Driven Software Evolution*, Amsterdam, The Netherlands, 2007.

[14] M. Herrmanndoerfer, S. Benz, and E. Juergens, "COPE - Automating Coupled Evolution of Metamodels and Models," in *23rd European Conference on Object-Oriented Programming*, Genova, Italy, 2009, pp. 52-76.

[15] L. M. Rose, D. S. Kolovos, R. F. Paige, and F. A. C. Polack, "Model Migration with Epsilon Flock," in *Third International Conference on Theory and Practice of Model Transformations (ICMT 2010)*, Malaga, Spain, 2010, pp. 184-198

[16] S. Roser and B. Bauer, "Automatic generation and evolution of model transformations using ontology engineering space," *Journal on Data Semantics XI,* vol. 5383, pp. 32-64, 2008.

[17] D. Mendez, A. Etien, A. Muller, and R. Casallas, "Towards Transformation Migration After Metamodel Evolution," in *13th International Int.l Workshop on Models and Evolution (ME2010)*, Oslo, Norway, 2010, pp. 84-89.

[18] J. García, O. Diaz, and M. Azanza, "Model Transformation Co-evolution: A Semi-automatic Approach," *Lecture Notes in Computer Science,* vol. 7745, pp. 144-163, 2013.

[19] D. Wagelaar, L. Iovino, D. D. Ruscio, and A. Pierantonio, "Translational semantics of a co-evolution specific language with the EMF transformation virtual machine," in *Proceedings of the 5th International Conference ob Model Transformation (ICMT 2012)*, Prague, Czech Republic, 2012, pp. 192–207.

[20] C. K. F. Corrêa, T. C. Oliveira, and C. M. L. Werner, "An Analysis of Change Operations to Achieve Consistency in Model-Driven Software Product Lines," in *International Workshop on Model-driven Approaches in Software Product Line Engineering*, Munich, Germany, 2011.

[21] A. Leon, *Software Configuration Management Handbook*, 2 ed. Boston: Artech House, 2004.

[22] A. Schürr, "Specification of Graph Translators with Triple Graph Grammars," in *Proceeecings of 20th International Workshop on Graph-Theoretic Concepts in Computer Science*, Herrsching, Germany, 1994, pp. 151–163.

[23] C. Brun, L. Goubet, M. Barbero, and C. Notot. (2013, 02/05/2013). *EMF Compare*. Available: http://wiki.eclipse.org/EMF_Compare

[24] M. Paternostro and K. Hussey. (2006, Accessed 07 January 2013). Advanced features of EMF. *Tutorial at EclipseCon*. Available: http://www.eclipsecon.org/2006/Sub.do?id=171

[25] C. A. Curino, H. J. Moon, and C. Zaniolo, "Graceful database schema evolution: the PRISM workbench," presented at the 34th International Conference on Very Large Databases, Auckland, New Zealand, 2008.

[26] J. Andany, M. LConard, and C. Palisser, "Management Of Schema Evolution In Databases," in *17th International Conference on Very Large Data Bases* Barcelona, Catalonia, Spain, 1991, pp. 161-170.

[27] T. Apiwattanapong, A. Orso, and M. J. Harrold, "A Differencing Algorithm for Object-Oriented Programs," in *19th IEEE International Conference on Automated Software Engineering*, Linz, Austria, 2004, pp. 2-13.

[28] D. Jackson and D. A. Ladd, "Semantic Diff: A Tool for Summarizing the Effects of Modifications," in *International Conference on Software Maintenance* Victoria, BC, Canada, 1994, pp. 243-252.

[29] Z. Xing and E. Stroulia, "UMLDiff: An Algorithm for Object-Oriented Design Differencing," in *20th IEEE/ACM International Conference on Automated Software Engineering*, Long Beach, California, USA, 2005, pp. 54-65

[30] A. Egyed, "A Scenario-Driven Approach to Trace Dependency Analysis," *IEEE Transactions on Software Engineering,* vol. 29, pp. 116-132, 2003.

Measuring the Structural Similarity between Source Code Entities

Ricardo Terra*, João Brunet†, Luis Miranda*, Marco Túlio Valente*,
Dalton Serey†, Douglas Castilho*, and Roberto Bigonha*

*Universidade Federal de Minas Gerais, Brazil
Email: {terra,luisfmiranda,mtov,douglas.castilho,bigonha}@dcc.ufmg.br

†Universidade Federal de Campina Grande, Brazil
Email: {jarthur,dalton}@dsc.ufcg.edu.br

***Abstract*—Similarity coefficients are widely used in software engineering for several purposes, such as identification of refactoring opportunities and system remodularizations. Although the literature provides several similarity coefficients that vary on the computing strategy, there is a tendency among researchers to make habitual use of certain coefficients that others in their field are using. Consequently, some approaches might be using an inadequate coefficient for their purpose. In this paper, we conduct a quantitative study that compares 18 coefficients to identify which one is the most appropriate in determining where a class should be located. Our evaluation contemplates 111 open source systems from Qualitas Corpus, which totalizes more than 70,000 classes. As a result, we observed that Jaccard—one of the most used coefficients in our area—has not presented the best results. While Jaccard correctly indicated the suitable module to 22% of the classes, other coefficients were able to indicate 60%.**

I. Introduction

Similarity coefficient measures the degree of correspondence between two entities according to an established criterion. This concept is widely used in software engineering area for several different purposes, such as identification of refactoring opportunities [1]–[3] and system remodularizations [4], [5]. For example, Fokaefs et al. employ a well-known similarity coefficient—named *Jaccard* [6]—to measure similarity between methods of a given class in order to recommend a *Extract Class* refactoring for those methods with low similarity. As another example, Simon et al. also employed *Jaccard* coefficient to propose a set of cohesion metrics that help developers to identify refactoring opportunities, such as *Move Method* and *Move Attribute* [3].

The literature is prolific and provides several similarity coefficients that vary on their computing strategy. Nevertheless, there are few works comparing similarity coefficients using structural dependencies as source of information [4]. This lack of knowledge may lead researchers to choose an inadequate coefficient to this purpose, once there is a tendency among researchers to make habitual use of certain coefficients that others in their field are using, even without sound scientific or empirical reasons [7].

In this paper, we conduct a quantitative study that compares 18 coefficients to identify which one is the most appropriate in determining where a class should be located. We computed the similarity between classes and packages of 111 open source systems from Qualitas Corpus [8]. From our results, we can point out three main findings:

1) Structural dependencies are indeed precise enough to determine where a class should be located. In our evaluation, we achieved an overall precision of 80% in indicating the correct package of a class up to rank 3.

2) Considering the dependency type or the multiplicity of dependencies does not improve the overall precision. Our results show that simply relying on the existence of dependencies between two entities—i.e., without considering the dependency type and multiplicity—achieves the best precision results.

3) *Jaccard*—one of the most used coefficients in our area— has not presented the best results. While *Jaccard* indicated the correct package to only 22% of the classes, other coefficients—such as *Relative Matching*, *Kulczynski*, and *Russell and Rao*—were able to indicate to slightly over 60%.

A usual problem that developers face during software refactoring or remodularization is to indicate the suitable package in which a particular class should be located. In this context, our findings might improve software engineering approaches that need to determine the suitable package of a class. As aforementioned, *Jaccard* is not the most precise coefficient in the context of measuring the similarity between classes and packages using structural dependencies as source of information.

The remainder of this paper is structured as follows. Section II provides a description of similarity coefficients. Section III describes strategies for extracting structural dependencies from a class. Section IV presents and discusses results on comparing our strategies and coefficients in 111 real-world systems. Finally, Section V presents related work and Section VI concludes the paper.

II. Similarity Coefficients

Table I shows 18 similarity coefficients that we have evaluated to determine the most appropriated one in the context of measuring similarity among classes [7], [9]. To calculate these coefficients, we assume that a given source code entity (method, class, or package) is represented by the dependencies it establishes with other types. Therefore, the measure of the structural similarity between two source code entities i and j (i.e., S_{ij}) considers the following variables:

- $\boldsymbol{a}$ = the number of dependencies on both entities,
- $\boldsymbol{b}$ = the number of dependencies on entity $\boldsymbol{i}$ only,
- $\boldsymbol{c}$ = the number of dependencies on entity $\boldsymbol{j}$ only, and
- $\boldsymbol{d}$ = the number of dependencies on neither of the entities.

For instance, *Jaccard*—one of the simplest and most used coefficient in our field—is defined by:

$$S_{ij} = \frac{a}{a+b+c} \quad (1)$$

Basically, *Jaccard* indicates maximum similarity when two entities have identical dependencies, i.e., when $b = c = 0$ and thus $S_{ij} = 1.0$. On the other hand, it indicates minimum similarity when there are no dependencies in common, i.e., when $a = 0$ and thus $S_{ij} = 0.0$.

```
class Bar extends X {
    A a;
    B b;

    exampleBar(D d){
        a.f();
        d.g();
    }
}
```

```
class Foo extends X {
    B b;
    G g;

    exampleFoo(E e){
        e.j();
        new A().f()
    }
}
```

Code 1. Hypothetical classes to explain the measurement of similarity

As an illustrative example, Code 1 presents two hypothetical classes. In order to measure the similarity between `Bar` and `Foo`, we first determine the value of the variables a, b, c, and d. In this example: $a = 3$ since both classes rely on `A`, `B`, and `X`; $b = 1$ since only `Bar` relies on `D`; $c = 2$ since only `Foo` relies on `E` and `G`; and $d = 3$ since none establishes dependencies with three other classes of the system (namely `C`, `F`, and `Y`). Next, we choose a similarity coefficient and solve the formula. For example, the similarity between `Bar` and `Foo` using *Jaccard* results in 0.5, whereas using *Phi* decreases to 0.35 or using *Kulczynski* increases to 0.675.

Each coefficient has a unique property that differs it from others. For example, while *Jaccard* does not consider what the both entities do not have in order to compute their similarity (variable d), *Simple matching* and 10 other coefficients contemplate it. The *Yule* and *Hamann* coefficients are mathematically related. Although both have the same variables in their numerators and denominators, *Hamann* relates the variable by addition whereas *Yule* relates them by multiplication.

As another example, *Sorenson*[1] gives twice the weight to what the entities have in common (variable a), while the *Rogers and Tanimoto* coefficient gives twice the weight to what each entity has independently (variables b and c). Except for the d variable in the denominator, *Russell and Rao* resembles *Jaccard*. On the other hand, the *Sokal and Sneath* coefficient, which is quite similar to *Simple matching*, reduces the importance of what each entity has independently (variables b and c) by half.

Kulczynski and *Sokal and Sneath 4* are based on conditional probability. *Kulczynski* assumes that a characteristic is present in one item, given that it is present in the other, whereas the *Sokal and Sneath 4* coefficient assumes that a characteristic in one item matches the value in the other. Finally, *Relative Matching* considers a set of similarity properties such as no mismatch, minimum match, no match, complete match, and maximum match.

TABLE I
General Purpose Similarity Coefficients

Coefficient	Definition S_{ij}	Range
1. Jaccard	$a/(a+b+c)$	0–1*
2. Simple matching	$(a+d)/(a+b+c+d)$	0–1*
3. Yule	$(ad-bc)/(ad+bc)$	-1–1*
4. Hamann	$[(a+d)-(b+c)]/[(a+d)+(b+c)]$	-1–1*
5. Sorenson	$2a/(2a+b+c)$	0–1*
6. Rogers and Tanimoto	$(a+d)/[a+2(b+c)+d]$	0–1*
7. Sokal and Sneath	$2(a+d)/[2(a+d)+b+c]$	0–1*
8. Russell and Rao	$a/(a+b+c+d)$	0–1*
9. Baroni-Urbani and Buser	$[a+(ad)^{\frac{1}{2}}]/[a+b+c+(ad)^{\frac{1}{2}}]$	0–1*
10. Sokal binary distance	$[(b+c)/(a+b+c+d)]^{\frac{1}{2}}$	0*–1
11. Ochiai	$a/[(a+b)(a+c)]^{\frac{1}{2}}$	0–1*
12. Phi	$(ad-bc)/[(a+b)(a+c)(b+d)(c+d)]^{\frac{1}{2}}$	-1–1*
13. PSC	$a^2/[(b+a)(c+a)]$	0–1*
14. Dot-product	$a/(b+c+2a)$	0–1*
15. Kulczynski	$\frac{1}{2}[a/(a+b)+a/(a+c)]$	0–1*
16. Sokal and Sneath 2	$a/[a+2(b+c)]$	0–1*
17. Sokal and Sneath 4	$\frac{1}{4}[a/(a+b)+a/(a+c)+d/(b+d)+d/(c+d)]$	0–1*
18. Relative Matching	$[a+(ad)^{\frac{1}{2}}]/[a+b+c+d+(ad)^{\frac{1}{2}}]$	0–1*

The symbol "$*$" denotes the maximum similarity.

III. Strategies

In order to measure the similarity between two source code entities, we assume that a given source code entity (a class or a package, in this paper) is represented by the *structural dependencies* it establishes with other types. We also distinguish the type of the dependency, i.e., whether a given dependency was established by accessing methods and fields (`access`), declaring variables (`declare`), creating objects (`create`), extending classes (`extend`), implementing interfaces (`implement`), throwing exceptions (`throw`), or using annotations (`useannotation`). Structural dependencies are extracted from the source code using a function named $Deps(E, S)$. Basically, this function returns `E`'s dependencies according to a strategy `S`. A strategy is a pair $[\texttt{C}, \texttt{D}]$ that defines the *collection*[2] and the *data information* to be employed in the extraction. The collection `C` can assume one of the following values:

[1] *Sorenson* is also referred on the literature as *Czekanowski* or *Dice*.

[2] We use the generic term "*collection*" when we do not need to be specific about the kind of structure (set or multiset) under consideration.

1) *set*: a collection that contains no duplicated elements. In other words, if a class establishes more than one dependency to `java.sql.Statement`, it considers only one.

2) *multiset*: a generalization of the notion of set in which elements are allowed to appear more than once. For instance, if a class establishes three dependencies to `java.sql.Statement`, we actually consider all of them.

The data information `D` can assume one of the following values:

1) *target type* (*tt*): in this case the extraction function returns a collection of target types that the entity establishes dependencies with. Thus, an element is a single $[\texttt{T}]$, denoting the existence of at least one dependency between the entity under analysis and `T`.

2) *target and dependency type* (*dtt*): in this case the extraction function returns a collection whose elements are pairs $[\texttt{dt}, \texttt{T}]$, denoting the existence of a dependency of type `dt` between the class under analysis and `T`.

```
public class Bar {
    public void foo (Date d){
        if (d == null){
            d = new Date();
        } else {
            new Date()
        }
    }
}
```

Code 2. An example class that explains our strategies

As an illustrative example, consider the class presented in Code 2. The collection returned by function $Deps(\texttt{Bar}, \texttt{S})$ differs according to the strategy `S` employed. More specifically, the following calls (and respective results) are possible:

$$Deps(\texttt{Bar}, [\texttt{set}, \texttt{tt}]) = \{\texttt{Date}\}$$
$$Deps(\texttt{Bar}, [\texttt{set}, \texttt{dtt}]) = \{[\texttt{declare}, \texttt{Date}], [\texttt{create}, \texttt{Date}]\}$$
$$Deps(\texttt{Bar}, [\texttt{mset}, \texttt{tt}]) = \{\texttt{Date}, \texttt{Date}, \texttt{Date}\}$$
$$Deps(\texttt{Bar}, [\texttt{mset}, \texttt{dtt}]) = \{[\texttt{declare}, \texttt{Date}], [\texttt{create}, \texttt{Date}], [\texttt{create}, \texttt{Date}]\}$$

Strategies that rely on *target and dependency type* may be particularly important when the classes of the system rely on types differently according to their location. For example, a factory method that creates a `DTO` (Data Transfer Object) and a logic presentation method that handles a `DTO` may not be similar. As another example, a class that implements `java.io.Serializable` and a method that declares `java.io.Serializable` may also not be similar. Although this strategy clearly performs better in particular cases, our evaluation is concerned with the overall precision.

Last but not least, the set of dependencies of a package `Pkg` is calculated by the union of the set of the dependencies of its classes as follows:

$$Deps(\texttt{Pkg}, \texttt{S}) = \bigcup_{C_i \in Pkg} Deps(\texttt{C}_\texttt{i}, \texttt{S})$$

IV. Evaluation

A. Research Questions

We designed a study to address the following overarching research questions:

RQ #1 – Are structural dependencies precise enough to indicate whether a class is located in the correct package?

RQ #2 – Considering the multiplicity of dependencies—i.e., a multiset rather than a set—improves the overall precision?

RQ #3 – Considering the dependency type—i.e., representing a target dependency as a pair $[\texttt{dt}, \texttt{T}]$ rather than only a single type $[\texttt{T}]$—improves the overall precision?

RQ #4 – Which coefficient is the most suitable to measure the similarity among classes of object oriented systems?

B. Target Systems

Our evaluation relies on the Qualitas Corpus[3], which is a collection of software systems intended to be used for empirical studies of code artifacts [8]. In its current version, the corpus includes the source code of many popular systems, such as JRE, Eclipse, NetBeans, and Apache Tomcat. Table II summarizes information about our data set.

TABLE II
Qualitas Corpus

# systems	111
# total of packages	6,841
# total of analyzed classes	71,823

It is worth noting that our data set is large and heterogeneous, ranging from text processors and small frameworks to complete IDEs and virtual machines.

C. Major Assumption

We made the following assumption due to the infeasibility in obtaining a 100% accurate oracle for thousands of classes.

"In order to conduct our experiment, we assume that every class under analysis is in its right package."

Therefore, similarity coefficients should indicate the *current* package of the class as its *most suitable one*.

D. Methodology

To provide answers to our research questions, we performed the following tasks:

1) *Setup*: First, we have set all system up, i.e., we imported and compiled the 111 projects from Qualitas Corpus.

2) *Data Extraction*: Second, we extracted the structural dependencies of classes using the four possible strategies described in Section III, i.e., $[\texttt{set}, \texttt{tt}]$, $[\texttt{set}, \texttt{dtt}]$, $[\texttt{mset}, \texttt{tt}]$, and $[\texttt{mset}, \texttt{dtt}]$.

[3]Qualitas Corpus v20120401. Available at: http://qualitascorpus.com.

3) *Comparative Analysis*: Third, we have measured the similarity using all coefficients described in Section II. The coefficients were applied to measure the similarity between pairs $[\texttt{class}, \texttt{package}]$ from our corpus.

4) *Qualitative Analysis*: Last, we have conducted a qualitative analysis in order to answer our research questions.

E. Experimental Setup

In order to conduct this experiment, the following policies have been proposed:

1) We have disregarded the class under analysis while searching for its right location. For example, when measuring the similarity between a class `A` and its package `Pkg`, we actually consider its own package `Pkg` as being $\texttt{Pkg} - \{\texttt{A}\}$.

   Thereupon, we have not sought the suitable location of classes whose package contains only such class. For example, assume that package `Pkg` contains only class `A`. The measure of similarity will be unfair because $Deps(\texttt{Pkg} - \{\texttt{A}\}, \texttt{S}) = \phi$.

2) We have disregarded a particular class $\texttt{C}_\texttt{i}$ when $|Deps(\texttt{C}_\texttt{i}, \texttt{S})| < 5$, i.e., we have not evaluated classes that establish less than five dependencies. These classes contain too little information to make any inference based on their structural dependencies.

3) We have not evaluated test classes, since most of the systems organize their test classes on a single package. Consequently, the test package contains classes related to different parts of the system—i.e., they are not structurally related—which certainly reduces the precision of any approach based on structural dependencies.

4) We have filtered trivial dependencies, such as those established with primitive and wrappers types (e.g., `int` and `java.lang.Integer`), `java.lang.String`, and `java.lang.Object`. Since virtually all classes establish dependencies with these types, they do not actually contribute for the measure of similarity. This decision is quite similar to text retrieval systems that exclude stop words because they are rarely helpful in describing the content of a document.

F. Results

Figure 1 illustrates the overall precision for each coefficient regarding the four analyzed strategies. The overall precision is defined by the ratio between the number of classes that have their location (package) correctly predicted by the similarity coefficient and the total number of analyzed classes. We have also provided the Top 1, 2, and 3 ranking, which stands for the position of the correct package of a class. As an example, considering strategy $[\texttt{set}, \texttt{tt}]$, the *Relative Matching* precision has reached 60% on Top 1, 72% on Top 2, and 78% on Top 3. In other words, it means that *Relative Matching* located the correct package of a class 60% on the first position of its ranking, 12% on the second position, and 6% on the third position.

Before we provide answers to our research questions, it is worth noting that many similarity coefficients presented very similar (mostly identical) results. In fact, the Spearman correlation among these coefficients was very close to 1, which allowed us to group them. The multiple correlation among *Simple Matching*, *Hamann*, *Rogers and Tanimoto*, *Sokal and Sneath*, and *Sokal Binary Distance* presented lowest correlation value of 0.999994. Similarly, *Jaccard*, *Sorenson*, *Dot-product*, and *Sokal and Sneath 2* presented lowest correlation value of 0.999999. Finally, *Ochiai* and *PSC* presented correlation equal to 0.998251. These results explain why there is no variance in the ranks within the same group.

Next, we answer our research questions based mainly on Figure 1. In all answers, our data interpretation always considers the Top 3 ranking—when not stated differently.

RQ #1: *Are structural dependencies precise enough to indicate whether a class is located in the correct package?*

Yes. As can be observed in Figure 1, there are coefficients that achieved a high precision to determine the package where a class should be located. In particular, *Relative Matching*, *Kulczynski*, *Russell and Rao*, and *Sokal and Sneath 4* indicate, in the worst scenario, over than 70% of precision.

RQ #2: *Considering the multiplicity of dependencies—i.e., a multiset rather than a set—improves the overall precision?*

No. Figure 1 shows that strategies that use the traditional set ($[\texttt{set}, \texttt{tt}]$ and $[\texttt{set}, \texttt{dtt}]$) perform better than an equivalent multiset-based strategy for all coefficients. The only exception is the *Russell And Rao* coefficient, which presented results slightly better for the $[\texttt{mset}, \texttt{tt}]$ strategy. More important, if we consider only Top 1, a traditional set-based strategy performs better than multiset-based one for all coefficients.

RQ #3 – *Considering the dependency type—i.e., representing a target dependency as a pair* $[\texttt{dt}, \texttt{T}]$ *rather than only a single type* $[\texttt{T}]$*—improves the overall precision?*

No. On one hand, Figure 1 shows that multiset-based data (i.e., $[\texttt{mset}, \texttt{tt}]$ and $[\texttt{mset}, \texttt{dtt}]$) presents very similar results for all coefficients. It is expected since the extracted collection is very similar. For instance, assume a collection `A` extracted using strategy $[\texttt{mset}, \texttt{dtt}]$ and a collection `B` extracted using strategy $[\texttt{mset}, \texttt{tt}]$. If $\texttt{A(i)} = [\texttt{access}, \texttt{Foo}]$ for an index `i`, then $\texttt{B(i)} = [\texttt{Foo}]$.

On the other hand, analyzing set data, we can observe that $[\texttt{set}, \texttt{tt}]$ provides better results for all coefficients, except for *Russell and Rao* and *Sokal and Sneath 4* that presented results slightly better using the dependency type ($[\texttt{set}, \texttt{dtt}]$).

From now on, our discussion only considers strategy $[\texttt{set}, \texttt{tt}]$, since we have demonstrated that the use of multiset

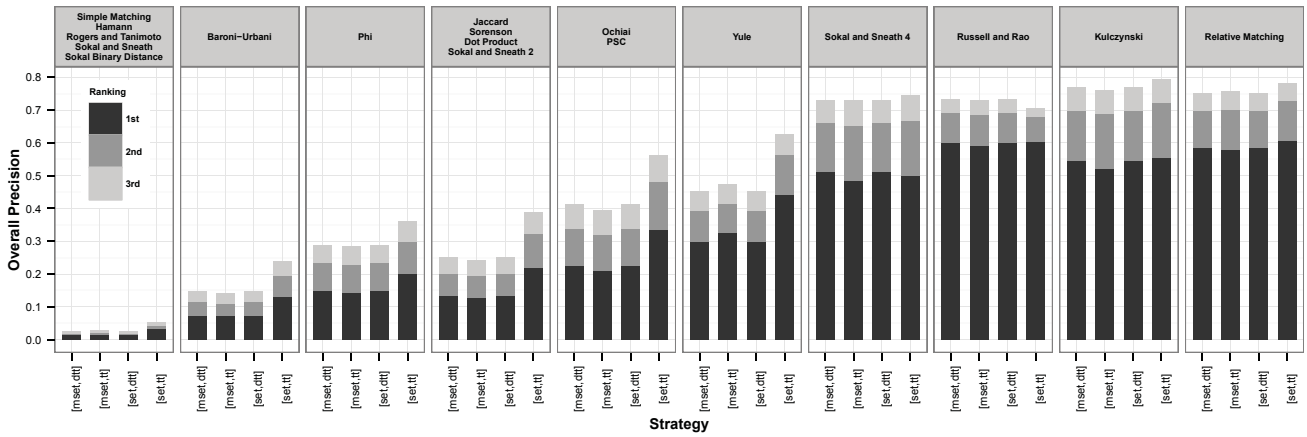


Fig. 1. Top 3 ranking of similarity coefficients using all strategies

(`mset`) and dependency type (`dtt`) does not actually improve the overall precision.

RQ #4: *Which coefficient is the most suitable to measure the similarity among classes of object oriented systems?*

Relative Matching*, *Kulczynski*, and *Russell and Rao. These coefficients have reached the highest precision values in our study. As can be observed in Figure 1, *Relative Matching* (60.83%) and *Russell and Rao* (60.27%) achieved the highest similarity values of the Top 1, and *Relative Matching* (72.78% and 78.18%) and *Kulczynski* (72.15% and 79.23%) of the Top 2 and 3.

As the central finding of our study, these three coefficients significantly outperform *Jaccard*—one of the most used similarity coefficients. While *Jaccard* indicated the correct package to 22% (Top 1) and 39% (Top 3) of the classes, *Relative Matching*, *Kulczynski* and *Russell and Rao* were able to indicate the correct package to 60% (Top 1) and 79% (Top 3).

To better explain this behavior, we anecdotally analyzed some systems to understand the influence of the variables a, b, c, and d (see Section II) on their precision. We performed this analysis by plotting each variable against the ranking.[4] Our major finding regards to the fact that large packages negatively influence *Jaccard* and other coefficients that presented very low precision (e.g., *Simple Matching*). Usually, large packages imply a large difference between c and d, which negatively impacts the precision of certain coefficients like *Jaccard*. On the other hand, by their nature, this scenario does not influence *Relative Matching*, *Kulczynski*, and *Russell and Rao*. It explains why these coefficients have presented the best results on both small and large packages.

G. Supplementary Results

Figure 2 illustrates the Top N ranking of every coefficient using strategy [`set`, `tt`]. In contrast to Figure 1 that displays only the Top 3, it displays the full distribution of the ranks

[4]Due to space constraints, we have not graphically presented this analysis.

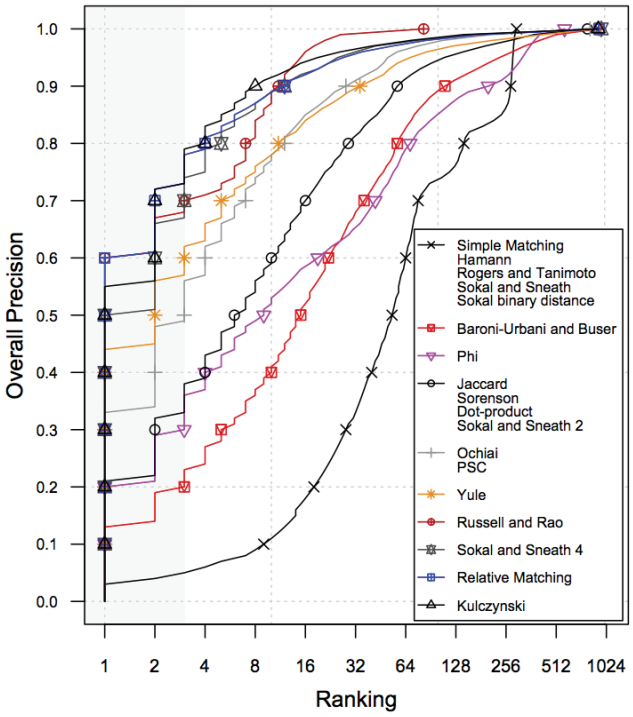


Fig. 2. General ranking using strategy [`set`, `tt`]

until full coverage (i.e., precision of 1.0). As a second relevant finding from our study, Figure 2 demonstrates that there is no coefficient that drastically improves its precision right after the top 3 ranking (e.g., Top 4 or Top 5). This behavior reinforces our decision in using Top 3. As can also be observed in Figure 2, *Russell and Rao* achieved precision of 1.0 in the rank 79. It means that the suitable package of a class was detected, in the worst case, on its 79th position. This result is quite relevant, since the other coefficients only achieved precision of 1.0 from the rank 347.

Since our analysis so far has considered the overall precision, we also analyzed the results of each coefficient per

system. Figure 3 summarizes the number of systems in which a particular coefficient has presented the best result (i.e., better identified the correct package of a class). For instance, *Relative matching* has better determined correct modules to *Eclipse* classes, whereas the *Russell and Rao* coefficient has behaved better to *ArgoUML* classes. Furthermore, we have not graphically presented some coefficients (e.g., *Simple Matching* and *Jaccard*) because they have not presented the best result for any system.

As can be observed in Figure 3, *Relative Matching*, *Kulczynski*, and *Russell and Rao* have presented the best results for most systems, which reinforces our claim that these coefficients are the most suitable ones to measure the similarity among classes in object-oriented systems.

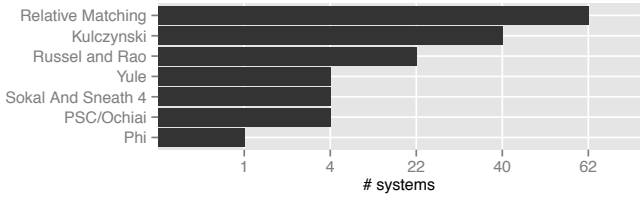


Fig. 3. # systems in which a particular coefficient presented the best result

H. Threats to Validity

We must state at least one major threat to the conclusion validity of the reported evaluation. Our experiment assumes that every class is in the right location. Although there might be misplaced classes, we rely on a stable and trustworthy collection of systems.

V. Related Work

There are few research works that compares similarity coefficients using structural dependencies as source of information [4]. Despite this lack of knowledge, similarity coefficients have been widely used for several different purposes [1]–[3], [5]. For example, the *Jaccard* distance between a method and a class is employed to support the automated identification of Feature Envy bad smells [1], [2]. Similarly, Simon et al. employ *Jaccard* distance to analyze similarity between classes and to identify refactoring opportunities [3]. They have proposed a cohesion metric based on *Jaccard* distance in order to suggest refactorings that improve the measurements of the metric. Our results suggest that the precision of the aforementioned approaches may be improved by using other coefficients that outperform *Jaccard* (e.g., *Relative Matching* and *Kulczynski*).

Fokaefs et al. employed *Jaccard* coefficient to develop a clustering method to suggest *Extract Class* refactorings for those entities with low level of similarity [5]. However, it is not clear in their paper how the authors handle structural dependencies to measure similarity among classes.

VI. Conclusion

First, we take the position that the choice of a similarity coefficient should not continue to be made without well-founded reasons. To address this shortcoming, we conducted a quantitative study that compares 18 coefficients to identify which one is the most appropriate in determining where a class should be located. As the major result, we observed that *Jaccard*—one of the most used coefficients in our area—has not presented the best results. While *Jaccard* indicated the correct package to only 22% of the classes, other coefficients—such as *Relative Matching*, *Kulczynski*, and *Russell and Rao*—were able to indicate to slightly over 60%.

Next, we have observed that the simplest strategy to extract structural dependencies from a class—set with only types ($[\texttt{set}, \texttt{tt}]$)—is indeed the best one. Stated differently, considering multisets of dependencies (`mset`) or considering also the dependency type (`dtt`) does not improve the overall precision.

Plans for future work include: (i) a sensitivity analysis of the factors `a`, `b`, `c`, and `d` in the ranking to statistically explain the behavior of each coefficient; (ii) an investigation of the impact on the results when measuring similarity of each pair $[\texttt{class}, \texttt{class}]$ and hence the similarity between a class `C` and a package `Pkg` will be calculated considering the average of the resulting similarity between `C` and `Pkg`'s classes; (iii) the extension of our comparative study to determine the most suitable class for a method; and (iv) the development of a tool that points out misplaced methods or classes.

Furthermore, we also have plans to use the main findings of the present study in the implementation of `ArchFix` [10], the recommendation system we are currently proposing to help developers to reverse software architecture erosion.

Acknowledgments: Our research has been supported by CAPES, FAPEMIG, and CNPq.

References

[1] N. Tsantalis and A. Chatzigeorgiou, "Identification of move method refactoring opportunities," *IEEE Transactions on Software Engineering*, vol. 99, pp. 347–367, 2009.

[2] ——, "Identification of extract method refactoring opportunities for the decomposition of methods," *Journal of Systems and Software*, vol. 84, no. 10, pp. 1757–1782, 2011.

[3] F. Simon, F. Steinbruckner, and C. Lewerentz, "Metrics based refactoring," in *5th European Conference on Software Maintenance and Reengineering (CSMR)*, 2001, pp. 30–38.

[4] R. Naseem, O. Maqbool, and S. Muhammad, "Improved similarity measures for software clustering," in *15th European Conference on Software Maintenance and Reengineering (CSMR)*, 2011, pp. 45–54.

[5] M. Fokaefs, N. Tsantalis, E. Stroulia, and A. Chatzigeorgiou, "Jdeodorant: identification and application of extract class refactorings," in *33rd International Conference on Software Engineering (ICSE)*, 2011, pp. 1037–1039.

[6] I. H. Moghadam and M. Ó. Cinnéide, "Automated refactoring using design differencing," in *15th European Conference on Software Maintenance and Reengineering (CSMR)*, 2012, pp. 43–52.

[7] H. C. Romesburg, *Cluster Analysis for Researchers*. Lifetime Learning Publications, 1981.

[8] E. Tempero, C. Anslow, J. Dietrich, T. Han, J. Li, M. Lumpe, H. Melton, and J. Noble, "The Qualitas Corpus: A curated collection of Java code for empirical studies," in *17th Asia Pacific Software Engineering Conference (APSEC)*, 2010, pp. 336–345.

[9] B. S. Everitt, S. Landau, M. Leese, and D. Stahl, *Cluster Analysis*, 5th ed. Wiley, 2011.

[10] R. Terra, M. T. Valente, K. Czarnecki, and R. Bigonha, "Recommending refactorings to reverse software architecture erosion," in *16th European Conference on Software Maintenance and Reengineering (CSMR), Early Research Achievements Track*, 2012, pp. 335–340.

On Use Case Identification

David Kung
Department of Computer Science and Engineering
The University of Texas at Arlington
Arlington, Texas 76019-0015, USA

Abstract

Identifying the real requirements is the single hardest part of system development. While requirements specify the capabilities of the software system, use cases specify how to deliver these capabilities. It is important to identify real use cases because wrong use cases make it difficult to design, implement, test and maintain the software system. This increases effort and costs, and reduces software quality. Unfortunately, identifying real use cases is not easy for many software engineers, due to the lack of a clear definition. In this paper, we provide a definition and present a use case derivation methodology. Experiments and experiences show that the proposed method improves the correctness and completeness of the use cases identified.

Keywords Software engineering, requirements, use case modeling, deriving use cases, software quality, use case metrics, case study and experience.

1. Introduction

Brooks pointed out that "the hardest single part of building a software system is deciding precisely what to build — i.e., the requirements." [6]. Use cases provide a partial solution to this problem. While requirements specify the capabilities or WHAT the software system must deliver, use cases specify HOW the software system will deliver the capabilities. Since the inception of the notion of a use case [9], numerous books and articles on use case modeling have been published. Today, use cases are widely used in agile as well as plan-driven development. Despite these efforts, identifying use cases remains a challenge for many software engineers. It is not uncommon that some real use cases are not identified and wrong use cases are identified. This increases development costs and reduces software quality because efforts are required to correct errors.

One reason that use cases are not identified correctly is due to the lack of a clearly defined use case concept. Another reason is the lack of a use case derivation methodology. This paper is aimed to fill these gaps. In particular, section 2 presents a use case definition that is easy to understand and use. It also clarifies related notions that often lead to identifying wrong use cases. Section 3 discusses use cases in the software life cycle to justify the importance of identifying correct use cases. In Section 4, we present a use case derivation methodology. Assessments of the methodology are presented in Section 5. In particular, we define the correctness and completeness metrics, and show that the methodology led to significant improvement of these metrics. Related work is presented in Section 6 and Section 7 presents the summary and future work.

2. Basic Concepts

In this paper, use cases are identified using the following definition:

Definition 1. A *use case* is a business process. It begins with an actor, ends with the actor, and accomplishes a business task for the actor.

An *actor* is a (business) role played by and on behalf of a set of (business) entities or stakeholders that are external to the system and interact with the system. Consider, for example, an ATM has use cases such as *Deposit Money, Check Balance, Withdraw Money*, and *Transfer Money* because these are business processes of a bank. Moreover, these processes begin and end with the ATM customer, and accomplish these business tasks for the customer.

Business processes are often confused with steps, operations, and actions of a business process. For example, "contact bank server," and "enter password" are sometimes identified as use cases but in fact the former is a step and the latter is an action of a business pro-

Application	Use Case	Step/Operation	Action
Text Editor	Edit Report	Open Report	click File, select Open, navigate to appropriate directory, select file, click OK button
		Make Changes	add, delete and modify texts and graphics (these are editor specific editing actions)
		Save Report	click File, select Save
		Exit Editor	click File, select Exit
Class Diagram Editor	Edit Diagram	Open Diagram	click File, select Open, navigate to appropriate directory, select file, click OK button
		Add Class	right click in canvas, select Add Class, fill in class information, click OK button
		Save Diagram	click File, select Save
		Exit Editor	click File, select Exit
ATM	Deposit Money / Withdraw Money	Start	insert card
		Authenticate	enter password, press Enter key
		Do transaction	select transaction type, enter deposit/withdraw amount, insert cash/take cash, take deposit/withdraw slip
		Finish	press Exit button, take ejected card

Figure 1. Use cases, steps, operations and actions

cess. Therefore, it is desirable to distinguish a business process with related concepts, as illustrated in Figure 1:

1. A *business process* is a series of information processing steps to complete a business task. Identifying business processes is the most important in use case identification.

2. An *operation* is a series of acts or instructions to carry out a step of a business process. For example, "get balance from database" is a step of a business process.

3. An *action* is an indivisible act, movement, or instruction that is performed during the performance of an operation. Examples are "enter password" and "click Submit button." Sometimes, an operation is an action, and vice versa.

3. Use Case in Life Cycle

The importance of identifying real use cases is explained in the context of a software life cycle, as shown in Figure 2. Use cases are derived in the early stage of the life cycle and affect architectural design, actor-system interaction modeling (ASIM), and object interaction modeling (OIM). For example, the functional cohesions of the subsystems represented by the use case diagrams affect the quality of the architectural design. For each use case, ASIM specifies how a user interacts with the system to carry out the business process. The actor-system interaction behavior is used to prepare user's manual and conduct use case based acceptance testing during testing and deployment. Actor requests needing internal computation are analyzed by OIM, which specifies how objects collaborate to produce the desired responses. These are represented by sequence diagrams. During maintenance, use cases and architectural design are useful for functional enhancement, error correction, and re-engineering. For example, requirements change may introduce additional business processes, add features to existing use cases. These show that use case derivation has significant impact on software quality, and effort to perform subsequent development activities. In particular, if steps, operations, or actions are identified as use cases, then the number of so-called use cases is large. This could lead to more work in subsequent activities and more effort to correct errors.

4. Methodology

Use cases should be derived from requirements and satisfy the requirements. This ensures that no use cases are redundant and all requirements are fulfilled by some use cases. The methodology for deriving use cases consists of the following steps:

1. Examine each functional requirement, look for or infer a domain-specific verb-noun phrase that represents an business process.

2. Answer the following questions for the verb-noun phrase identified: (1) Is it a complete business process of the application? (2) Does it begin with an actor? (3) Does it end with the actor? (4) Does it accomplish a business task useful for the actor?

3. If all the answers to above questions are "yes," then continue — the verb-noun phrase is a use case, else goto Step 1 — examine the next functional requirement.

4. Assign a unique identifier to the use case and enter the relationship that the requirement derives the use

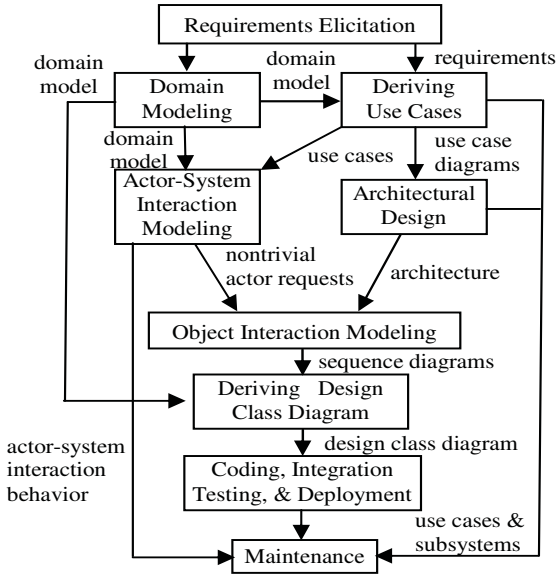


Figure 2. Use cases in the software life cycle

R1. The LIS must allow a patron to check out documents

R2. The LIS must allow a patron to return documents

Verb-Noun Phrase	Checkout Documents	Return Documents	Allow a Patron
Is it a business process?	Y	Y	N
Does it begin with an actor?	Y	Y	N
Does it end with the actor?	Y	Y	N
Does it accomplish a business task for the actor?	Y	Y	N
Is it a use case?	*Y*	*Y*	*N*
Actor	Patron	Patron	NA
Subsystem	LIS	LIS	NA

Legend: oval=verb-noun phrase underline=actor rectangle=system

Figure 3. Identify use cases for an LIS

case into a requirement-use case traceability matrix (RUCTM).

5. If the appropriate actor has not been identified in Step 2, then from the requirement or its context, look up or infer the subject, who initiates the business process, or the object for which the business process is initiated. This identifies the actor for the use case.

6. Similarly, look up or infer the organizational unit, context, system, or subsystem within which the business process is performed. This identifies the system or subsystem for the use case.

7. Repeat above steps until all functional requirements are examined.

Figures 3 shows how use cases, actors and subsystems are identified for a library information system (LIS). Sometimes, the verb-noun phrase cannot be literally identified from a requirement. Inference is needed to identify the functionality and then the business process. These are shown in Figure 4, which is a part of a project we performed for the Office of International Education (OIE) of our university.

Use cases are also derived from nonfunctional requirements such as security requirements. For example, an authentication requirement implies that the software system must provide functions to authenticate users. From these, one may derive login and logout use cases. Nonfunctional requirements may be affected by a use case. In this case, the relationship is entered into the RUCTM.

Subsystems should achieve high cohesion. Low cohesion may occur when a subsystem contains too many use cases. High cohesion can be obtained by partitioning the use cases and assign them to separate subsystems. In particular, the following rules are repeatedly applied, in the order listed:

1. *Role-Based Partition.* Use cases for a common actor tend to exhibit role-specific business processes. Therefore, use cases can be partitioned according to their common actors.

2. *Communicational Partition.* Use cases that operate on a common object tend to perform object-specific tasks. For example, use cases of a UML diagram editor can be partitioned according to the objects they process, resulting in a project-specific partition containing use cases such as *New Project, Edit Project (properties), Open Project, Close Project*, and *Delete Project*.

3. *Type-based Partition.* Sometimes, the object that modifies the noun of the use case can be used to partition the use cases. For example, *Edit Class Diagram* and *Edit State Diagram* should belong to different partitions.

Inheritance relationships are useful for arranging the use cases among subsystems. For example, students,

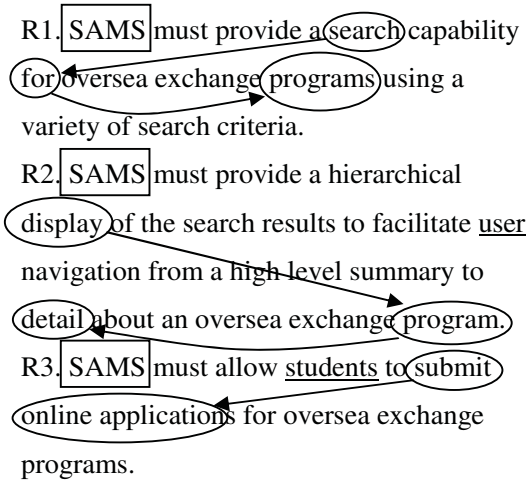


Figure 4. Identify use cases for a Study Abroad Management System (SAMS)

OIE staff members, and OIE advisors are account users, who can login and logout. Thus, *Login* and *Logout* can be assigned to an Authentication subsystem. The actor-specific subsystems do not need to show these use cases. To reduce the number of use cases of a subsystem, use cases of an actor subclass can be separated from use cases of the actor superclass to form a subsystem of their own. This is in fact role-based partition applied in the inheritance context. Use cases of an actor subclass can be merged with use cases of an actor superclass to reduce the number of subsystems if desired. For example, SAMS has 26 use cases, partitioned into 7 subsystems as follows, where "A → B" means A is a subclass of B and inherits the use cases of B:

1. **Web User (Actor: Web User):** *Search for Programs, Display Program Detail, View Feedback.*

2. **Account-User (Actor: Account User → Web User):** *Edit Preferences, Logon, Log-off.*

3. **Administrator (Actor: Admin → Account User):** *Create User Account, Delete User Account, Update User Account, Edit System Settings, Startup System, Shutdown System.*

4. **Student (Actor: Student → Account User):** *Submit Online Application, Edit Online Application, Check Application Status, Post Feedback.*

5. **OIE Advisor (Actor: OIE Advisor → Account User):** *Review Online Application, Notify Student, Notify Academic Advisor*

6. **Academic Department (Actor: Faculty → Account User, Academic Advisor → Account User):** *Submit Recommendation Letter, Approve Course Equivalency Form, View Application Review Results.*

7. **OIE Staff (Actor: OIE Staff → Account User):** *Add Program, Delete Program, Update Program, Upload Programs.*

5. Assessment

To evaluate the usefulness of the proposed use case derivation method, we define two metrics — *correctness* and *completeness*. Let RUC be the set of real use cases implied by the requirements. In practice, RUC can be determined jointly by domain experts, use case modeling experts and users. In our experiments, the RUCs are determined by the instructor and the graduate teaching assistants using Definition 1. Let UC denote the set of use cases identified by an experiment participant. The correctness and completeness metrics for UC are defined as follows:

$$\text{Correctness } Q_{cr}(UC) = \frac{|UC \cap RUC|}{|UC|}$$

$$\text{Completeness } Q_{cp}(UC) = \frac{|UC \cap RUC|}{|RUC|}$$

That is, correctness is the percentage of all use cases identified by the participant that are real use cases. Completeness is the percentage of all real use cases identified by the participant. For example, let $uc_1, uc_2, ..., uc_5$ be the real use cases and $uc_1, uc_2, uc_3, uc_6, uc_7, uc_8$ be the use cases identified by a participant. Then the Q_{cr} and Q_{cp} of the identified use cases are 0.50 and 0.60, respectively. Note: UC-RUC is the set of non-use cases identified, and RUC-UC is the set of use cases missed. It should be pointed out that a use case may be named differently, such as "Edit Diagram" and "Update Diagram." Therefore, we renamed the use cases, if needed, before calculating $UC \cap RUC$ and $UC - RUC$ in our experiments.

The first experiment involved six teams that applied the use case derivation methodology and six teams that did not. The case study required the teams to derive use cases for a portion of a domain modeling tool. It had three use cases: *Derive Domain Model Concepts, Display Domain Model*, and *Show Context-Dependent Help*. The case study was scaled down so the teams could complete the tool in one semester. The results are given in Figure 5, which shows that the methodology leads to significantly better correctness average (71.03% versus 26.49%), and completeness average (88.89% versus 61.11%). The average number of use cases ($|\ UC\ |$)

Teams Applying Methodology (DM Tool)

	|UC · RUC|	|UC|	|UC-RUC|	Q_{cr}	Q_{cp}
T1	3	3	0	100.00%	100.00%
T2	3	3	0	100.00%	100.00%
T3	2	3	1	66.67%	66.67%
T4	3	6	3	50.00%	100.00%
T5	3	7	4	42.86%	100.00%
T6	2	3	1	66.67%	66.67%
Avg	2.67	4.17	1.5	71.03%	88.89%

Teams Not Applying Methodology (DM Tool CG)

	|UC · RUC|	|UC|	|UC-RUC|	Q_{cr}	Q_{cp}
G1	2	5	3	40.00%	66.67%
G2	1	4	3	25.00%	33.33%
G3	0	7	7	0.00%	0.00%
G4	2	6	4	33.33%	66.67%
G5	3	11	8	27.27%	100.00%
G6	3	9	6	33.33%	100.00%
Avg	1.83	7.00	5.17	26.49%	61.11%

|RUC|=3

Figure 5. Results of modeling tool case study

and non-use cases ($| UC - RUC |$) identified by the control group are significantly higher (7 versus 4.15, and 5.17 versus 1.50). This means many non-use cases, such as steps and operations, were identified. It confirms our observation that without a methodology students tend to include steps and operations as use cases. The completeness average for the control groups looks better than its correctness counterpart. This is because the control groups tend to include everything as well as the real use cases.

The second case study was performed in a different semester and required five teams to design and implement a part of a large online *office patient medical record* (OPMR) management system. The system had 30 use cases but the case study involved only 5 (i.e., RUC=5): *Record Phone Call, Add Special Note, Edit Special Note, Delete Special Note*, and *Generate Reports*. All teams were required to apply the use case derivation methodology. Figure 6 shows that the teams achieved similar performance although the correctness average is slightly better (79.05% versus 71.03%). Probably, this is due to that the OPMR case study is easier to understand and less confusing.

Thirty-eight subjects participated in the third case study, which was conducted in a third semester. It required the subjects to individually derive use cases from requirements for a National Trade Show Service (NTSS) system, which had 14 real use cases. Because steps, operations and actions are often misidentified as use cases, this case study required the subjects to classify domain-specific verb-noun phrases into business processes, steps, operations, or actions before deriving the

Team	|UC · RUC|	|UC|	Q_{cr}	Q_{cp}
1	5	6	83.33%	100.00%
2	5	6	83.33%	100.00%
3	5	7	71.43%	100.00%
4	4	7	57.14%	80.00%
5	3	3	100.00%	60.00%
Average	4.4	5.80	79.05%	88.00%

|RUC|=5

Figure 6. Results of Office Patient Medical Record case study

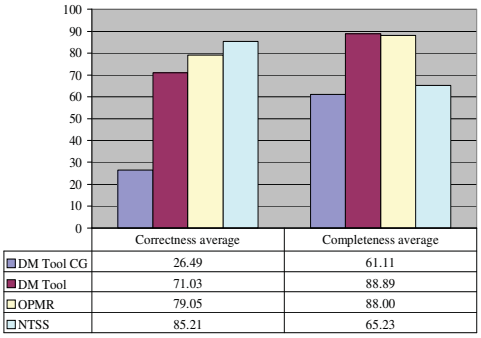


Figure 7. Comparing results of case studies

use cases. The correctness and completeness averages were 85.21% and 65.23%, respectively. Noticeably, 15 subjects or 39.47% achieved 100% correctness. Figure 7 compares the two metrics for the case studies. It shows that the methodology results in significantly higher correctness averages than the control group. The completeness averages are also significantly higher except for the NTSS case study, which is still better than the control group. The NTSS completeness average is not significantly higher because it had more requirements and use cases. It was more likely for a subject to miss a real use case and result in lower completeness measurement.

6. Related Work

Use cases are widely discussed in the literature, e.g. [4, 5, 7, 9, 10, 11, 14, 16]. Usefulness of use case in the software life cycle is discussed in various papers including [1, 2, 3, 13]. However, few publications provide a clear, easy to use definition of the notion of a use case. Definition 1 given in Section 2 fills this gap. A practically useful methodology for identifying use cases is also lacking in the literature. In particular, one that can be used by beginners to identify real use cases. One related work is found in [12], where

a context-free language is proposed for specifying requirements, from which use cases and actors can be derived automatically. The limitations are that the syntax is very restrictive, and it is a challenge for a software engineer to use a formal language. In comparison, our methodology works with requirements specified using the natural language. Another related work is found in [15], which describes a method for generating use case diagrams from use cases and architectural designs. Our methodology generates use case diagrams directly and use case diagrams are used to derive an architectural design. It is beyond the scope of this paper to discuss the differences further.

7. Summary and Future Work

We define the notion of a use case and distinguish it from other related concepts such as steps, operations and actions. We present a methodology for use case derivation. Experiments show that the methodology greatly improves the correctness and completeness of use cases identified. A tool supporting the methodology has been prototyped. It works in two modes: (1) manual derivation, and (2) automatic derivation of use cases from requirements. In manual derivation, the user highlights and identifies the use cases, actors and subsystems. The tool presents the four questions stated in Definition 1 and asks the user to verify. In the automatic mode, the tool derives the use cases automatically and the user verifies them. In either mode, use case diagrams are generated automatically. Future work is to improve, and evaluate the tool.

References

[1] Bente Anda, Dag Sjberg, "Towards an inspection technique for use case models," Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering, SEKE '02, July 15-19, 2002. pp. 127-134.

[2] Bente Anda and Dag I.K. Sjberg, "Investigating the role of use cases in the construction of class diagrams," Empirical Software Engineering, vol. 10, no. 3, September 2005 pp. 285-309.

[3] Flvia A. Barros, Las Neves, rica Hori, Dante Torres, "The ucsCNL: A controlled natural language for use case specifications," Proceedings of the 23rd International Conference on Software Engineering and Knowledge Engineering, SEKE '11, pp. 250-253.

[4] Michael R. Blaha and James R Rumbaugh, "Object-Oriented Modeling and Design with UML (2nd Ed.)," Prentice Hall, 2004.

[5] G. Booch, J. Rumbaugh and I. Jacobson, "The Unified Modeling Language User Guide (2nd Ed.)," Addison Wesley, 2005.

[6] F. P. Brooks, Jr. "The Mythical Man-Month (2nd Ed.)," Addison-Wesley Professional, 1995.

[7] Bernd Bruegge and Allen H. Dutoit, "Object-Oriented Software Engineering: Using UML, Patterns, and Java (3nd Ed.)," Prentice Hall, 2009.

[8] Stefania Gnesi, "Use case-based testing of product lines," Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering, 2003. pp. 355-358.

[9] Ivar Jacobson, "Object-Oriented Software Engineering: A Use Case Driven Approach" Reading, MA: Addison-Wesley, 1992.

[10] Ivar Jacobson, James Rumbaugh and Grady Booch, "Unified Software Development Process," Addison-Wesley, 1999.

[11] Craig Larman, "Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Ed.)," Prentice Hall, 2005.

[12] Dipankar Majumdar, Sabnam Sengupta, Ananya Kanjilal, Swapan Bhattacharya, "Adv-EARS: A Formal Requirements Syntax for Derivation of Use Case Models," in Advances in Computing and Information Technology Communications in Computer and Information Science Vol. 198, 2011. pp. 40-48.

[13] Martyn Ratcliffe and David Budgen, "The application of use cases in systems analysis and design specification," Information and Software Technology, vol. 47, no. 9, 2005. pp. 623-641.

[14] J. Rumbaugh, I. Jacobson and G. Booch, "The Unified Modeling Language Reference Manual," Addison-Wesley, 1999.

[15] Maribel Yasmina Santos and Ricardo J. Machado, "On the derivation of class diagrams from use cases and logical software architectures," Proceedings of 5th International Conference on Software Engineering Advances, ICSEA 2010. pp. 107-113.

[16] Stephen R. Schach, "Object-Oriented Software Engineering," McGraw-Hill Companies, Inc. 2007.

ScubAA: A Human Plausible Reasoning Approach to Agent Trust Management

Sadra Abedinzadeh
Dept. of Computer Science,
University of Regina,
Regina, SK, Canada
sadra@cs.uregina.ca

Samira Sadaoui
Dept. of Computer Science,
University of Regina,
Regina, SK, Canada
sadaouis@uregina.ca

***Abstract*— Agent Trust Management (ATM) is becoming very important because of the growth of the number of autonomous service providers in open systems such as the Web. By modeling open systems with agent technology, ATM methods are able to identify a set of the most trusted service agents. The goal of this paper is to highlight important features of ScubAA, an ATM framework that is based on the Human Plausible Reasoning (HPR) theory. Thanks to HPR, ScubAA is able to evaluate and manage the agents' trust by considering several features, such as the context of the user's request, third-party references from both trustees (i.e. users) and trustors (i.e. service agents) and the hierarchy of relations the trustees may have, in one single framework.**

Keywords - Agent Trust Management; Human Plausible Reasoning; Multi Agent Systems.

I. OBJECTIVES

In this paper, we highlight major features of ScubAA, a general purpose ATM framework based on the theory of HPR [1]. We discuss how these features are important in addressing ATM problems [1, 2] as well as provide the results of our evaluation of ScubAA based on the statistical method named ANOVA [2].

II. AN OVERVIEW OF SCUBAA

For each submitted request, ScubAA first recommends to the user a ranked list of the most trusted service agents, associated to the same context of the request, and then forwards the request to those trusted services only. ScubAA divides the agents into trustors (users) and trustees (service agents). It makes use of HPR to relate a user to their trusted service agents. For this purpose, it maintains a Knowledge Base (KB) that includes the hierarchies representing all the agents in the society as well as the relations between them. Furthermore, ScubAA utilizes the HPR transformation functions to generate new relations between trustors, between trustees, and between the agents in the hierarchy of trustors and trustees. Each relation in KB is bound to a certainty parameter γ which states the degree of certainty that the corresponding statement is true. ScubAA uses this parameter as the degree of trust. Moreover, by identifying the context of the user's request, it is able to associate a context to each statement in KB. This context is then used to retrieve those agents which are able to provide trust references within the same context. ScubAA takes into account the values of trust provided by related agents as well as the degree of certainty of the identified context. Finally, to update the trust relations in KB, our system gets the values of interaction rating from the user and by applying several aggregation functions (such as mathematical average and Dempster-Shafer Theory [5]), it is able to calculate the trust for different service agents.

III. MAJOR FEATURES OF SCUBAA

Table 1 summarizes the major features of ScubAA that are important to manage the trust of trustees. ScubAA considers different metrics w.r.t. the user's request, system inputs and outputs as well as the properties of both trustors and trustees.

TABLE 1. MAJOR FEATURES OF SCUBAA

Category	**Feature**
User's request	Context
Trustor Agent	Third Party References
	History of Interactions
Trustee Agent	Third Party References
	Current Value of Trust
	User's Interaction Rating Value
Other Features	Degree of Certainty
	Tuning Parameters
Output	Ranked List of Most Trusted Service Agents

1. Context: ScubAA identifies the context of the request w.r.t to the properties of both the user's profile (such as the history of interactions) and request (such as its domain). The context plays an important role in ATM in the sense that a trustor evaluates trustees within a specific context. For instance, you may trust your physician in the context of diagnosing diseases, but not in the context of repairing a watch.

2. Third-Party References from Trustors: ScubAA uses third-party references from other trustors. By applying the HPR transformation functions, i.e. generalization, specialization and similarity, it is able to find related trustors to the trustor of interest. Our system determines the related entities only in the

context of the request. Third-party references are important as they provide more trust evidences for trustees that are already trusted by a trustor. Further, they deliver valuable information regarding the trustees for which the trustor does not have any trust information, and as a result, they can extend the trust network of trustors.

3. History of Interactions: ScubAA utilizes the history of trustors' requests in order to find the degree of similarity between them in a specific context. This value is then used by ScubAA to update the certainty parameters of the similarity relations between trustors in the KB.

4. Third-Party References from Trustees: ScubAA also considers the references from other trustees. It uses the HPR transformation functions to determine the related trustees in the context of the request.

5. Current Value of Trust: To evaluate the trust for each trustee, ScubAA aggregates several factors to obtain a single quantitative value. Current value of trust of a trustee is the one of these factors. The aggregation functions can be customized according to the ATM applications.

6. Interaction Rating Value: ScubAA employs the interaction rating value provided by the user to update its KB. This complies with the fact that the trust is subjective, and consequently the user's opinion plays an important role in the evaluation process of trust.

7. Degree of Certainty: ScubAA associates with each statement in KB (hierarchy, relation and context) a certainty value γ. Then, it uses these values, by employing several aggregation functions, to calculate the trust for each trustee.

8. Tuning Parameters: There are a number of tuning parameters introduced to ScubAA to make it possible to adjust how the trust evaluation process is performed in different ATM scenarios and applications.

9. Ranked List of Most Trusted Agents: In ScubAA, each agent in the list of trusted service agents is associated with a degree of trust, a value between 0 and 1. This implies that the trusted agents are sorted based on their degree of trust.

IV. Implementation and Evaluation

The main components of ScubAA are the HPR engine and the service agent system. We implemented a generic engine for the HPR theory. Moreover, a friendly GUI allows ScubAA to be used by any ATM application. To assess the accuracy of ScubAA, we conducted an experiment on the domain of Web search in order to manage trust for an agent-based system consisting of three search engines, Yahoo!, Bing, and Google. We employed CAPNET [3], an agent development framework, to implement the service agent system. Then, using a set of 40 queries, we compared the results of ScubAA with the actual values of trust returned by the three search engines. These results are computed with the statistical model ANOVA [2]. As illustrated in Figure 1, the analysis reveals that there are no statistically significant difference between the actual values of trust and the values of trust returned by ScubAA.

ANOVA						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Sample	1706.667	1	1706.667	9.33576	0.002508	3.881505
Columns	3483.775	2	1741.888	9.528424	0.000105	3.034414
Interaction	22.35833	2	11.17917	0.061152	0.940695	3.034414
Within	42777.45	234	182.8096			

Figure 1. Result of ANOVA

V. Comparison

We compare ScubAA to three other ATM systems, AFRAS [4], FIRE [5] and TRAVOS [6], from a theoretical point of view w.r.t. the features they utilize to address ATM problems. This comparison reveals that ScubAA takes into account several features within the same framework to provide a more accurate value of trust. However, the other three methods lack such characteristic since ScubAA contains all the features that each of them expose separately. For instance, FIRE uses references from trustees but does not consider the context of the request and references from trustors.

VI. Future Works

To better assess ScubAA, we can apply it to another domain (e-commerce for instance) in order to compare its results with other ATM systems in that domain. Customizing ScubAA with other aggregation functions, such as Fuzzy OWA [7], and comparing the results is another future research work. Due to the importance of context in ScubAA, one can extend this work by improving the accuracy of the process of identifying the context. ScubAA only uses the certainty parameter γ amongst nine different parameters available in HPR. A future extension of this work would be to incorporate those parameters in order to improve the accuracy of trust values.

VII. References

[1] A. Collins and R. Michalski, "The Logic of Plausible Reasoning: A Core Theory," *Cognitive Science*, vol. 13, 1989, pp. 1-49.

[2] D.C. LeBlanc, *Statistics: Concepts and Applications for Science.*, Jones & Bartlett Publishers, 2004.

[3] E. German and L. Sheremetov, "An agent framework for processing FIPA-ACL messages based on interaction models," *Proceedings of the 8th international conference on Agent-oriented software engineering.*, Springer-Verlag, 2008, pp. 88-102.

[4] J. Carbo, J. Molina and J. Davila, "Trust management through fuzzy reputation," *International Journal of Cooperative Information Systems*, vol. 12, no. 01, 2003, pp. 135-155.

[5] H. Trung Dong, R.J. Nicholas and R.S. Nigel, "An integrated trust and reputation model for open multi-agent systems," *Autonomous Agents and Multi-Agent Systems*, vol. 13, no. 2, 2006, pp. 119-154.

[6] W.T.L. Teacy, J. Patel, N.R. Jennings and M. Luck, "Travos: Trust and reputation in the context of inaccurate information sources," *Autonomous Agents and Multi-Agent Systems*, vol. 12, no. 2, 2006, pp. 183-198.

[7] J.M. Merigó and M. Casanovas, "The fuzzy generalized OWA operator and its application in strategic decision making," *Cybernetics and Systems: An International Journal*, vol. 41, no. 5, 2010, pp. 359-370.

Attribute-Value based Reconfiguration Model for Sensor Network Environment

Hyunjun Jung, Sukhoon Lee, Doo-Kwon Baik†
Dept. of Computer and Radio Communications Engineering, Korea University, Seoul, Korea
E-mail: {darkspen, leha82, baikdk}@korea.ac.kr

***Abstract*— Software reconfiguration is indispensable in Wireless Sensor Networks (WSNs) environment. A sensor network system requires reconfiguration to approve energy efficiency, change actions of sensor, and set alternative network routing. This paper proposes a reconfiguration model in WSNs. For attribute-value based reconfiguration, we define attributes of sensor network and reconfiguration processes. The attributes are used for updating attribute values of sensor network nodes. And the reconfiguration processes describe actions for the nodes. The proposed model is low cost and needs minimal user intervention.**

Keywords - reconfiguration model; attribute based reconfiguration;

I. INTRODUCTION

Wireless Sensor Networks (WSNs) have gained worldwide attention in recent years. The sensor nodes used are small and have limited processing and computing resources. Sensor nodes are typically deployed in the natural environment to sense data such as temperature, humidity, illumination, sound, vibration, pressure, motion, and pollution. The sensor nodes are also embedded in part of the infrastructure such as buildings, forests, and machinery, and are used in uncontrollable environments. Therefore sensor nodes are needed to be changed for adapting variable situations [1].

Nodes comprising WSNs have attribute values. The nodes reconfigure attribute values themselves to adapt changeable situation. The attribute-value based reconfiguration is able to approve energy efficiency, change actions of sensor, and set alternative network routing.

This paper proposes an attribute-value based reconfiguration model in WSNs. For attribute value based reconfiguration, we define attributes and reconfiguration process. For an evaluation, we compare attribute value based model to full image based model and module based model.

II. ATTRIBUTE-VALUE BASED RECONFIGURATION MODEL FOR SENSOR NETWORK

Figure 1 gives an overview of the reconfiguration system. The WSNs application system comprises sensor nodes, sink nodes, routers, and actuator nodes. Sensor nodes play the role of sensors and sense data that they then transmit to a coordinator node. Router nodes act as both routers and coordinators, and they control sub-networks and accept data from other nodes. A sink node has both a sink role and a Personal Area Network (PAN) coordinator role. It controls the entire network. A sink node collects data from other nodes in the sensor network. An actuator node, which plays the role of actuator, controls devices [2]. The measured data store is an ontology registry and ontology uses reasoning for situation awareness. The W3C Semantic Sensor Network (SSN) Incubator group (the SSN-XG) produced SSN ontology to describe sensors and observation [3]. A sink node uses SSN ontology for reasoning. This is necessary for it to adapt to situations. The sink node creates update code using inferred policy and sensed data. Update code is transferred to target nodes. The sensor nodes that receive the update code carry out the update process.

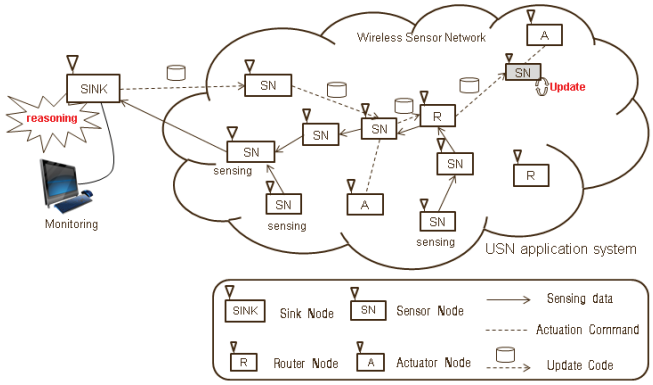


Figure 1. Overview of Attribute-value based Reconfiguration Model

TABLE I. ATTRIBUTES CLASSIFICATION AND FEATURES

Attribute classification	*Feature*
Network	Network comprises communication attributes shared with other nodes.
	E.g., NodeID, SensingPeriod, MACAddr, AdjacentActuatorNodeID, NextHopRoutingFirstNodeID, NextHopRoutingSecondNodeID, RFChannel, ScanChannel, Zigbee RF
Scheduler	Scheduler is scheduler selection.
	E.g., Scheduler_Enable
Device driver	Device driver comprises functional attributes.
	E.g., EEPROM_Enable, UART, FlashMemory_Enable, RSSI_Enable
Sensor type	Sensor type comprises sensing attributes.
	E.g., Sensor_Temperature_Enable, Sensor_Light_Enable, Sensor_Gas_Enable, Sensor_Humidity_Enable

This research was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology (2012M3C4A7033346).

† Corresponding author

III. ATTRIBUTE CLASSIFICATION FOR RECONFIGURATION

Attributes represent the capability of a node. They are used to determine the type of a node. The reconfiguration attribute is a functional capability of the USN application. It comprises network, scheduler, device driver, sensor type, and so on. USN application systems have various reconfiguration attributes.

Table 1 lists our proposed attribute classification and features. Self-reconfiguration attributes are classified into four categories: network, scheduler, device driver, and sensor type. Figure 2 shows examples of self-reconfiguration attribute settings. In the examples, the four classifications include attributes for a Gas and Light Monitoring System [4].

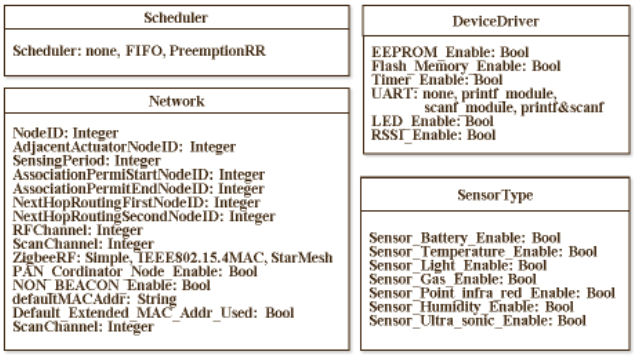


Figure 2. Examples of attribute-value pairs

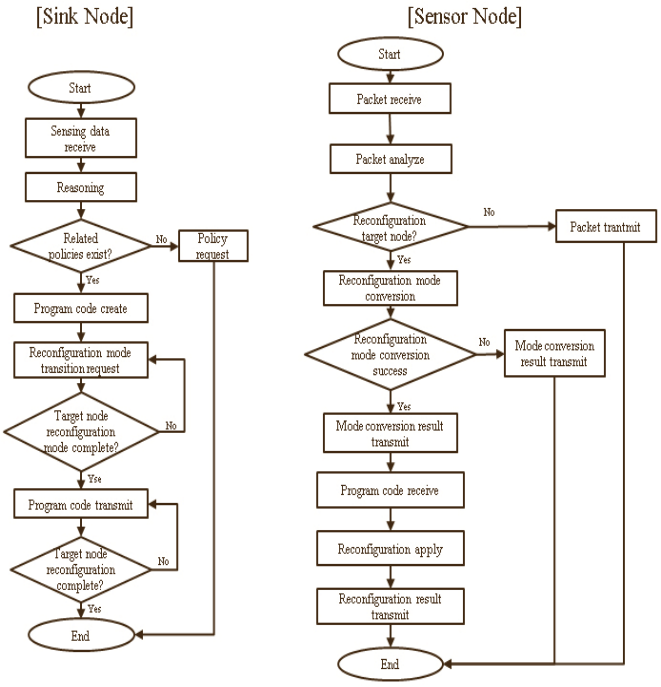


Figure 3. Process for attribute-value based reconfiguration

IV. ATTRIBUTE-VALUE BAESD RECONFIGURATION PROCESS

Attribute-value based reconfiguration process comprises two main processes: the sink node process and the sensor node process.

Figure 3 outlines the attribute-value based reconfiguration procedure. The sink node receives sensing data from the sensor nodes, which it saves in the store. The sink node needs situational awareness and reasoning based on ontology. It decides on the reconfiguration policy from the policy manager. If there is a relevant policy, it creates program code based on that policy. If there is no relevant policy, it requests a policy from the user. The sink node requires the target nodes to be in reconfiguration mode. Finally, it transmits the program code. If a sensor node receives a reconfiguration mode packet from the sink node, it changes to reconfiguration mode. (Otherwise, the program code is transmitted to other nodes.) The sensor node then waits to receive the program code, and carries out reconfiguration after receiving it. Finally, the sensor node transmits the reconfiguration results to the sink node.

V. EVALUATION

We conducted a qualitative evaluation. The evaluation factors are a reconfiguration cost and a unit size. Table 2 shows the results of our evaluation.

TABLE II. A QUALITATIVE EVALUATION

System	Reconfiguration cost	Unit size
Attribute-value based model (proposal)	Low	Small
Full image based model [5]	High	Large
Module based model [6]	Middle	Medium

VI. CONCLUSION

This paper addressed the problem of reconfiguration in sensor network environments and proposed a reconfiguration middleware model. The model uses attribute classification and reconfiguration mechanisms, and needs minimal user intervention. We also defined a middleware structure and described and analyzed the process of sensor network self-reconfiguration. Future work in this area will include the implementation of our proposed model.

REFERENCES

[1] B. Greenstein, A. Pesterev, C. Mar, E. Kohler, J. Judy,S. Farshchi, and D. Estrin, "Collecting high-rate data over low-rate sensor network radios," Technical report, CENS Technical Report 55, UCLA, 2005.

[2] R. Balani,C.-C. HAN, R. Rengaswamy, M. Srivastava, "Multi-level software reconfiguration for sensor networks," In Proceedings of the 6th ACM & IEEE International Conference on Embedded Software (EMSOFT'06). ACM, 2005, pp.112–121.

[3] Compton M et al. (2012) The SSN Ontology of the W3C Semantic Sensor Network Incubator Group. Journal of Web Semantics. 2012.

[4] Lee, W.J., Kim J.I., Kang J. M., "Automated Construction of Node Software Using Attributes in a Ubiquitous Sensor Network Environment", Sensors 2010, 10, p. 8664.

[5] J. W. Hui and D. Culler, "The dynamic behavior of a data dissemination protocol for network programming at scale," In Proceedings of the second internation conference on Embedded Networked Sensor Systems, 2004.

[6] A. Dunkels, B. Gr¨onvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors. In Proceedings of the First IEEE workshop on Embedded Networked Sensors, 2004.

DuSE-MT: From Design Spaces to Automated Software Architecture Design

Sandro S. Andrade[†] [§]
sandros@ufba.br

[§] GSORT Distributed Systems Group
Federal Institute of Education, Science
and Technology of Bahia (IFBa)
40110-150. Salvador-Ba. Brazil

Raimundo José de A. Macêdo[†]
macedo@ufba.br

[†] Distributed Systems Laboratory (LaSiD)
Department of Computer Science
Federal University of Bahia (UFBa)
40170-110. Salvador-Ba. Brazil

Abstract

The design of effective software architectures which fulfill complex and conflicting demands for stringent quality attributes is usually a challenging task. Despite the availability of some forms of structured architectural knowledge, like styles catalogs and reference architectures, effective tool support for systematic representation of domain-specific design knowledge and well-informed trade-off decisions is still missing. This extended abstract presents DuSE-MT: a supporting tool for the DuSE automated architecture design process. Major DuSE-MT features currently available include: i) the definition of domain-specific architecture design spaces; ii) the definition of related quality evaluation metrics; and iii) a multi-objective optimization mechanism which elicits a set of Pareto-optimal candidate architectures and supports well-informed trade-off analysis. DuSE-MT has been fully implemented in C++/Qt and a case study regarding the design of architectural self-adaptation capabilities in a cloud-based media encoding service has been undertaken.

Keywords: *software architecture design, self-adaptive systems, feedback control, model-driven software engineering.*

1. Introduction

The well-orchestrated use of refined experience, good aesthetics, and right balance when making trade-off decisions is imperative if we are to design effective architectures for complex software-intensive systems. Over the past years, software engineering researchers have been urged to drive their efforts towards an engineering discipline for software [4], which mostly involves the prospection of theories and organization of knowledge for routine use [3].

For that purpose, the development of supporting tools which systematically captures highly specialized design knowledge, supports effective solution space exploration, and helps when making well-informed trade-off decisions is mandatory. Our approach – DuSE[1] [1, 2] – integrates model-based software engineering and search-based architecture design mechanisms to address the aforementioned aspects. DuSE-MT, the tool we present here, is intended to fully support the automated design process provided by DuSE and depicted in Figure 1.

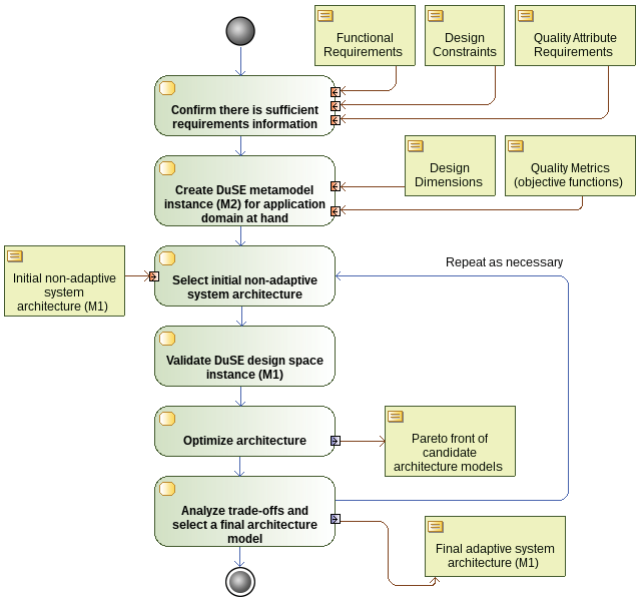


Figure 1. DuSE architecture design process.

DuSE defines a domain-independent metamodel (language) for design spaces representation which can be instantiated in order to specify degrees of freedom for specific application domains. Currently, a DuSE instance representing a design space for architecting self-adaptive systems is available (SA:DuSE).

This research is partially supported by grant 006/2012/PRPGI from the IFBa's Research and Innovation Supporting Program.

[1] http://duse.sf.net

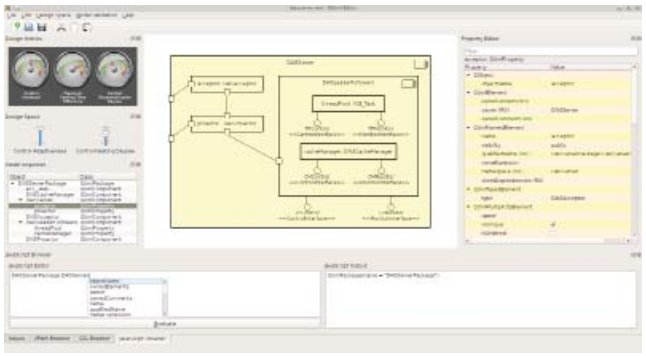


Figure 2. DuSE-MT: the DuSE supporting tool.

Our approach assumes an initial architectural model is fed into DuSE-MT (shown in Figure 2), which then enables the automated redesign of such initial model when navigating through the design space. Each resulting candidate architecture can be evaluated according with the defined metrics and usually denote a solution favoring a specific quality attribute.

In order to tackle the issue of navigating through huge design spaces, a search-based mechanism based on evolutionary multi-objective optimization has been implemented. The outcome of such mechanism is a set of Pareto-optimal candidate architectures: solutions for which it is impossible to make any one architecture better off without make at least another one worse.

2. DuSE-MT Requirements and Architecture

From its outset, DuSE-MT has been aimed to extend basic modeling capabilities towards a higher level representation of design expertize and enhanced elicitation of architectural trade-offs. With this in mind, the core foundations of DuSE-MT were designed to provide a metamodel-agnostic platform for defining architecture design spaces (representing alternative solutions in a given application domain) and quality evaluation metrics.

Figure 3 presents a simplified Component&Connector architectural view of DuSE-MT. DuSE builds upon *Qt*[2] and *Qt Modeling Framework*[3] technologies, responsible for basic UML model manipulation. A plugin-based architecture easily supports the addition of new design space instances (DuSE instance plugins) and provides the underlying mechanism for solution space exploration and optimization.

DuSE and DuSE-MT have currently been applied to the particular domain of self-adaptive systems. In [2] we provide details about such specific instance of DuSE and the findings of using our approach when designing a self-adaptive cloud-based media encoding service.

[2]http://www.qt-project.org

[3]http://qt-project.org/wiki/QtModeling

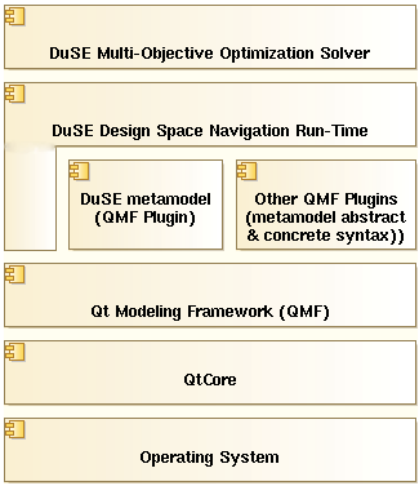


Figure 3. DuSE-MT C&C architectural view.

3. Conclusion and Future Work

This extended abstract presents an overview of DuSE-MT: a supporting tool for automated architectural design. We have presented its major requirements, current available features, and some structural aspects of its architecture. Future work include the implementation of effective metric visualization, sharing DuSE artifacts over a network, and mapping from design space navigation traces to higher level design theories.

References

[1] Sandro S. Andrade and Raimundo José de A. Macêdo. Architectural design spaces for feedback control concerns in self-adaptive systems. In *Proceedings of the 25th International Conference on Software Engineering and Knowledge Engineering*, SEKE 2013, New York, NY, USA, 2013. ACM.

[2] Sandro S. Andrade and Raimundo José de A. Macêdo. A search-based approach for architectural design of feedback control concerns in self-adaptive systems, May 2013. Technical Report. Available at http://dusearchitects.files.wordpress.com/2013/05/duse-techrep.pdf.

[3] Muhammad Ali Babar, Torgeir Dingsyr, Patricia Lago, and Hans van Vliet. *Software Architecture Knowledge Management: Theory and Practice*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[4] Mary Shaw. Research toward an engineering discipline for software. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*, FoSER '10, pages 337–342, New York, NY, USA, 2010. ACM.

Author's Index

Reviewer's Index

A

Silvia Teresita Acuna
Taiseera Albalushi
Edward Allen
Omar El Ariss

B

Doo-hwan Bae
Ebrahim Bagheri
Hamid Bagheri
Rami Bahsoon
Xiaoying Bai
Purushotham Bangalore
Fevzi Belli
Ateet Bhalla
Swapan Bhattacharya
Alessandro Bianchi
Borzoo Bonakdarpour
Ivo Bukovsky

C

Gerardo Canfora
Jaelson Castro
Raul Garcia Castro
Peggy Cellier
Keith Chan
Kuang-nan Chang
Ned Chapin
Shu-Ching Chen
Wen-Hui Chen
Zhenyu Chen
Stelvio Cimato
Peter Clarke
Esteban Clua
Nelly Condori-fernandez
Fabio M. Costa
Maria Francesca Costabile
Jose Luis Cuadrado
Juan J. Cuadrado-gallego

D

Aldo Dagnino
Jose Luis De La Vara
Massimiliano Di Penta
Scott Dick
Junhua Ding
Jing Dong
Weichang Du
Philippe Dugerdil

E

Christof Ebert
Ali Ebnenasir
Raimund Ege
Magdalini Eirinaki

F

Davide Falessi
Behrouz Far
Scott D. Fleming
Liana Fong
Renata Fortes
Ellen Francine Barbosa
Fulvio Frati

G

Jerry Gao
Felix Garcia
Ignacio Garcia Rodriguez De Guzman
Itana Gimenes

Swapna Gokhale
Wolfgang Golubski
Desmond Greer
Eric Gregoire
Christiane Gresse Von Wangenheim
Katarina Grolinger

H

Hao Han
Xudong He
Miguel Herranz

J

Clinton Jeffery
Jason Jung
Natalia Juristo

K

Selim Kalayci
Eric Kasten
Taghi Khoshgoftaar
Jun Kong
Nicholas Kraft
Aneesh Krishna
Vinay Kulkarni
Gihwon Kwon

L

JJeff Lei
Bixin Li
Ming Li
Tao Li
Yuan-Fang Li
Zhi Li
Shih-hsi Liu
Xiaodong Liu
Yi Liu
Hakim Lounis
Joan Lu

M

Marcelo de Almeida Maia
Antonio Mana
Vijay Mann
Riccardo Martoglia
Hong Mei
Hsing Mei
Ali Mili
Alok Mishra

N

Kia Ng
Allen Nikora
Amjad Nusayr

O

Edson A. Oliveira Junior

P

Erick Passos
Xin Peng
Oscar Pereira
Antonio Piccinno
Alfonso Pierantonio
Daniel Plante

R

Rick Rabiser
Filip Radulovic
Damith C. Rajapakse
Rajeev Raje
Jose Angel Ramos
Henrique Rebelo
Marek Reformat
Robert Reynolds

Daniel Rodriguez
Ivan Rodero

S

Samira Sadaoui
Masoud Sadjadi
Claudio Sant'Anna
Salvatore Alessandro Sarcia
Andreas Schoenberger
Tony Shan
Michael Shin
Qinbao Song
George Spanoudakis
Jing Sun
Yanchun Sun
Gerson Sunye

T

Jeff Tian
Genny Tortora
Mark Trakhtenbrot
Peter Troeger
T.h. Tse

V

Giorgio Valle
Sylvain Vauttier
Silvia Vergilio
Akshat Verma
Sergiy Vilkomir
Arndt Von Staa

W

Gurisimran Walia
Huanjing Wang
Jiacun Wang
Linzhang Wang
Hironori Washizaki
Victor Winter
Guido Wirtz
Eric Wong
Franz Wotawa

X

Dianxiang Xu
Frank Xu
Haiping Xu

Y

Chi-lu Yang
Hongji Yang
Jijiang Yang
Huiqun Yu

Z

Cui Zhang
Du Zhang
Hongyu Zhang
Yong Zhang
Zhenyu Zhang
Hong Zhu
Xingquan Zhu
Eugenio Zimeo

Poster/Demo Presenter's Index

SEKE

2013

Boston
June 27-29

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

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, 2013
ISBN 1-891706-33-0 (paper)
ISSN 2325-9000 (print)