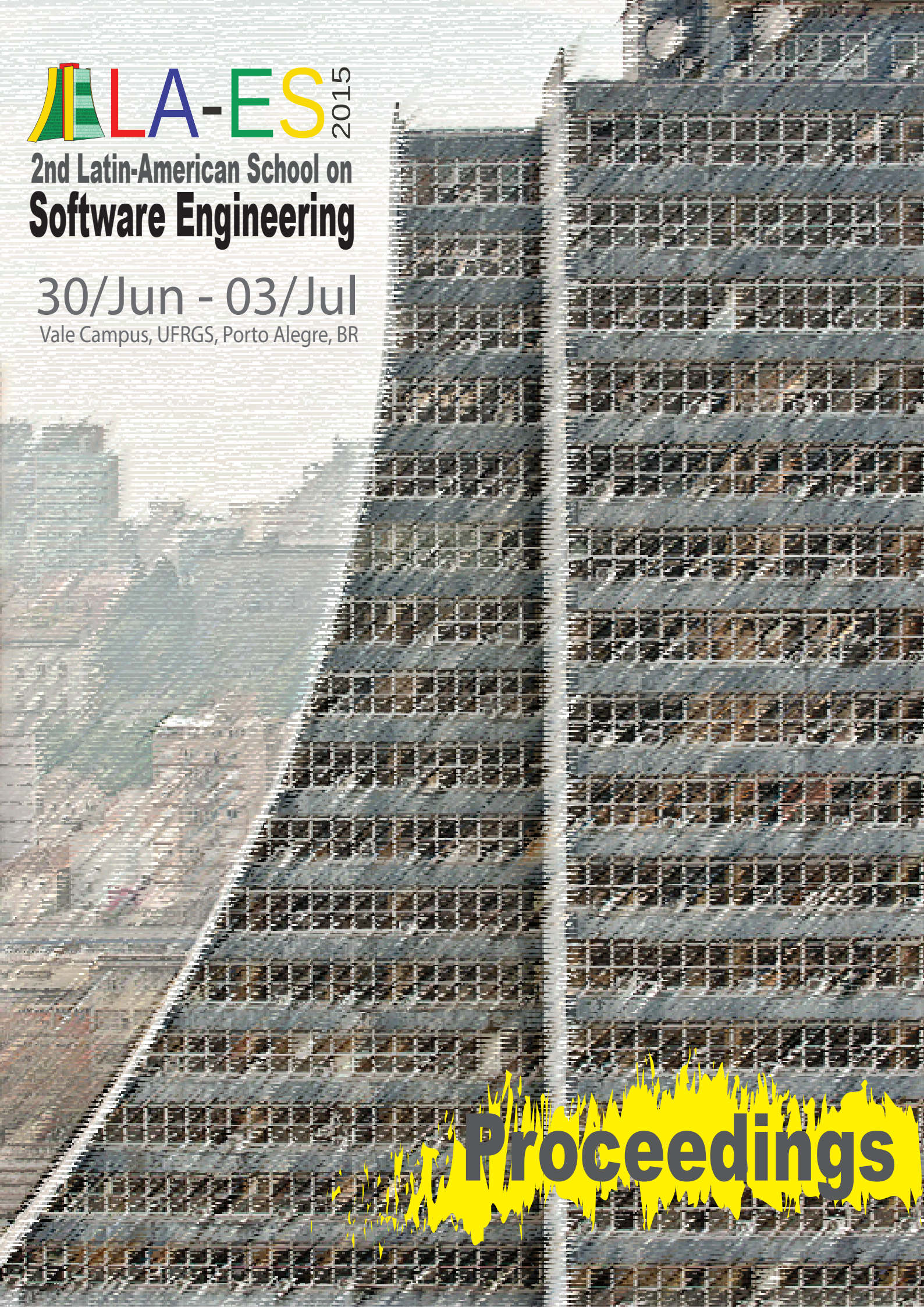


LA-ES 2015

2nd Latin-American School on Software Engineering

30/Jun - 03/Jul

Vale Campus, UFRGS, Porto Alegre, BR

Proceedings

ELA-ES 2015

II Latin-American School on Software Engineering

Ingrid Nunes and Francisco Dantas de Medeiros Neto

June 30th to July 3rd, 2015
Porto Alegre/RS – Brazil

PROCEEDINGS

Volume 01
1st Edition
ISBN: 978-85-88425-14-9

General Chairs
Ingrid Nunes
Francisco Dantas de Medeiros Neto

Realization
Instituto de Informática, Universidade Federal do Rio Grande do Sul (UFRGS)

Promoted by
Sociedade Brasileira de Computação (SBC)

Sponsors
Diamond sponsor: HP
Gold sponsors: ADP Labs, Dell, GE Research and Google

Support
CAPES, GE Research, UERN, PPGC-UFRGS, PROEXT-UFRGS, PROPG-UFRGS, PROPESQ-UFRGS

Publisher
UFRGS – Instituto de Informática

DADOS INTERNACIONAIS DE CATALOGAÇÃO-NA-PUBLICAÇÃO
(Porto Alegre, Brasil)

ELA-ES 2015 : II Latin-American School on Software Engineering (1. : 2015 : Porto Alegre, Rio Grande do Sul).
ELA-ES 2015 : [anais da] II Escola Latino Americana de Engenharia de Software [de] Ingrid Nunes e Francisco Dantas de Medeiros Neto – Porto Alegre: UFRGS – Instituto de Informática, 2015.
180 p. ; 21 cm.

Instituto de Informática, Universidade Federal do Rio Grande do Sul, Sociedade Brasileira de Computação.
Publicação composta por 1 volume, sendo este o volume 1.
ISBN : 978-85-88425-14-9.

1. Engenharia de software. I. coord. Ingrid Nunes. II. Francisco Dantas de Medeiros Neto. III. Anais.

CDD 620

Foreword

This proceedings records the contributions presented at the Second Latin American School on Software Engineering – ELA-ES, which took place in Porto Alegre, Rio Grande do Sul, from June 30th to July 3rd, 2015.

This second edition of the ELA-ES builds on the success of the previous event in this series, which took place in Rio de Janeiro (2013). ELA-ES provides intensive and exciting four-days of lectures on both basic and state-of-the-art themes of software engineering covering a wide range of topics, including software architecture, modularity, software testing, experimental software engineering. The school brings together undergraduate students, Master and PhD students and lecturers as well as other researchers and practitioners who are interested in software engineering and specifically the topics named above. The tutorials are given by renowned representatives of each domain of expertise. Each tutorial combines foundations, examples and advanced topics and some of them will present hands-on sessions when appropriate. Moreover, the event includes an industry panel, with discussions on applied research and the opportunities that are emerging in Latin America, with the creation of the Parque Científico e Tecnológico of UFRGS.

The program of the Latin American school also features lectures and activities that will address complementary issues that are relevant to young researchers: How to plan my research agenda in Software Engineering? What has been the evolution of Software Engineering in Latin America? What are the possible academic and industry careers after my studies?

The current edition of the workshop received a range of contributions. We received 10 technical paper submissions, from which we accepted 5 papers (50%). In addition, 3 of these papers were accepted as position papers (30%). We also received 28 position paper submissions, from which we accepted 22 papers (78.5%).

The success of ELA-ES 2015 was only possible because of the dedication and enthusiasm of many people. First, we would like to thank the authors for submitting their papers. We would also like to express our gratitude to the following people: the TPC members, for their dedication and great work when reviewing the submitted papers; the Organizing Committee, for their great help in giving shape to the event. In particular, we would like to thank the staff at Sociedade Brasileira de Computação (SBC), for their support.

We would also like to thank the invited speakers – Prof. Dr. Cláudia Werner (UFRJ, BR), Prof. Dr. Simone Barbosa (PUC-Rio, BR), Prof. Dr. Antónia Lopes (Universidade de Lisboa, PT), Prof. Dr. David Garlan (Carnegie Mellon University, USA), Prof. Dr. Don Batory (University of Texas, USA), Prof. Dr. Gail Murphy (University of British Columbia, CA), Prof. Dr. Gregor Engels (Universität Paderborn, DE), Prof. Dr. Guenther Ruhe (University of Calgary, CA) and Prof. Dr. Sebastián Uchitel (Universidad de Buenos Aires, AR and Imperial College London, UK) – and panelists for their valuable contribution to the event. Finally, we acknowledge CAPES and UFRGS for the financial support.

We hope ELA-ES goes ahead as a long-lasting school, thereby allowing a growing and inspiring sense of community and collaboration amongst researchers from Software Engineering community.

Porto Alegre, July 2015.

Ingrid Nunes
Francisco Dantas de Medeiros Neto
ELA-ES 2015 General Chairs

ELA-ES 2015 Chairs Short Biographies

Ingrid Nunes is a Professor Adjunto (Associate Professor) of the Instituto de Informática at the Federal University of Rio Grande do Sul (UFRGS), Porto Alegre, Brazil, and the head of the Prosoft research group. She completed her undergraduate studies in Computer Science at UFRGS (2006), obtained her Master's degree in Informatics at the Pontifical Catholic University of Rio de Janeiro (2009), and obtained her Doctor's degree in Informatics at the Pontifical Catholic University of Rio de Janeiro (2012). Her phd was in cooperation with King's College London (UK), under a sandwich Ph.D. programme of one year, and with University of Waterloo (Canada), with three three-month research visits. She was also a post-doc researcher at PUC-Rio in the Software Engineering Laboratory (LES) (2012), and has experience in the industry, where she worked as a software developer from 2005 to 2007. She is a section editor of the Scientific Initiation Magazine (REIC). Her main research areas are software engineering and artificial intelligence.

Francisco Dantas de Medeiros Neto is Professor Adjunto (Associate Professor) in the Computing Department at State University of Rio Grande do Norte, Brazil. He received his B.Sc. (2001) and M.Sc. (2004) degrees in Computer Science from Federal University of Rio Grande do Norte, Brazil. In 2013, he received the D.Sc. degree in Informatics from the Pontifical Catholic University of Rio de Janeiro, Brazil. His doctorate was developed in cooperation with Lancaster University (UK), under a sandwich Ph.D. programme of one year. Dantas' research interests include Advanced Techniques for Modular Programming, Product Lines, Software Metrics, Empirical Software Engineering and Software Architecture.

Technical Committees

General Chairs
Ingrid Nunes (UFRGS)
Francisco Dantas de Medeiros Neto (UERN)

Organizing Team
Daniela Brauner (UFPel)
Daltro Nunes (UFRGS)
Carlos Lucena (PUC-Rio)

Local Organizers
Fernando dos Santos (UFRGS)
Jacob de Quadros Stein (UFRGS)
Jhonny Marcos Acordi Mertz (UFRGS)
João Guilherme Faccin (UFRGS)
Lucas Lazzari Tomasi (UFRGS)
Matheus Medeiros Dias (UFRGS)
Vanius Zapalowski (UFRGS)

Program Committee
Adenilso Simao (USP, Brazil)
Alejandra Garrido (Universidad Nacional de la Plata, Argentina)
Álvaro Moreira (UFRGS, Brazil)
Ana Paula Terra Bacelo (PUC-RS, Brazil)
Baldoino Santos Neto (UFAL, Brazil)
Beatriz Marín (Universidad Diego Portales, Chile)
Claudia Pons (Universidad Nacional de La Plata, Argentina)
Cláudia Werner (UFRJ, Brazil)
Claudio Sant'Anna (UFBA, Brazil)
Daniel Lucrédio (UFSCar, Brazil)
Daniela Godoy (ISISTAN Research Institute, Argentina)
Diego Vallespir (Universidad de la República, Uruguay)
Edson A. Oliveira Junior (UEM, Brazil)
Eduardo Almeida (UFBA, Brazil)
Eduardo Figueiredo (UFMG, Brazil)
Eduardo Guerra (INPE, Brazil)
Eduardo Piveta (UFSM, Brazil)
Elder Cirilo (UFSJ, Brazil)
Erika Cota (UFRGS, Brazil)
Eugenio Scalise (UCV, Venezuela)
Fabiano Ferrari (UFSCar, Brazil)
Fabio Silveira (UNIFESP, Brazil)
Fernando Castor (UFPE, Brazil)
Gaston Mousques (Universidad ORT, Uruguay)
Guilherme Travassos (UFRJ, Brazil)
Gustavo Rossi (LIFIA, Argentina)
Hernán Astudillo (Universidad Técnica Federico Santa María, Chile)
Isela Bertran (GE Global Research Center, Brazil)
Jair Leite (UFRN, Brazil)
Jean Cheiran (UNIPAMPA, Brazil)
Juan Pablo Carvallo (Universidad del Azuay, Ecuador)

Judith Barrios (ULA, Venezuela)
Leandro Wives (UFRGS, Brazil)
Leila Ribeiro (UFRGS, Brazil)
Leopoldo Teixeira (UFPE, Brazil)
Lucineia Thom (UFRGS, Brazil)
Lucio Duarte (UFRGS, Brazil)
Luis Lamb (UFRGS, Brazil)
Marcelo Pimenta (UFRGS, Brazil)
Marco Tulio Valente (UFMG, Brazil)
María Cecilia Bastarrica (Universidad de Chile, Chile)
Nelio Cacho (UFRN, Brazil)
Norah Villegas (Icesi, Colombia)
Paulo Masiero (USP, Brazil)
Ricardo Choren (IME/RJ, Brazil)
Ricardo Terra (UFLA, Brazil)
Ricardo Soto (Pontificia Univ. Católica de Valparaíso, Chile)
Rodrigo Bonifacio (UnB, Brazil)
Rodrigo Paes (UFAL, Brazil)
Romain Robbes (University of Chile, Chile)
Rosângela Penteado (UFSCar, Brazil)
Roxana Giandini (Universidad Nacional de la Plata, Argentina)
Thais Batista (UFRN, Brazil)
Tiago Massoni (UFCG, Brazil)
Toacy Oliveira (UFRJ, Brazil)
Uirá Kulesza (UFRN, Brazil)
Valter Camargo (UFSCar, Brazil)
Vinicius Garcia (UFPE, Brazil)

Additional Reviewers

Alcemir Santos
Boris Almonacid
Carolina Valverde
Daniel Perovich
Italo Silva
Kattiana Constantino
Luiz Paulo Coelho Ferreira
Rodrigo Olivares

Table of Contents

Technical Papers

Position Papers

A Tentative Agenda and Perspectives for Software Engineering Young Researchers

Cláudia Werner[1]

[1]Federal University of Rio de Janeiro (UFRJ), BR

werner@cos.ufrj.br

1. Abstract

Becoming a new professor at a university or a researcher at an industrial research lab is a challenge. They are typically under tremendous pressure to teach/train new software engineers, supervise graduate students/subordinates, collaborate with industry/academia, raise research funds, be leaders in their field, and/or publish journal/conference papers/technical reports. Moreover, there is a tremendous shortage of Software Engineering (SE) faculty/professionals in many countries around the world. In Latin American the situation is not different. Thus, it is important to help young software engineering researchers survive in academia or industry in their early careers. Questions such as: "How to plan a research agenda in SE?", "What has been the evolution of SE in Latin America?", "What are the possible academic and industry careers after the studies?" and "How to balance career and personal life?" need to be handled. This talk aims to discuss a tentative agenda and perspectives for SE young researchers, providing ideas on practical guidelines for having a successful and fulfilling academic/industry career. We also present the trajectory of research in Brazil/Latin America and some challenges to overcome in this field.

2. Short bio

Cláudia Maria Lima Werner received her D.Sc. from COPPE/UFRJ (1992) (the Graduate School of Engineering of the Federal University of Rio de Janeiro, Brazil) and since 1994 is a Professor of the Computer Science Department, being the leader of the Software Engineering group, at COPPE/UFRJ. She is also a CNPq and FAPERJ (Cientista do Nosso Estado) researcher, having experience in Software Engineering for more than 20 year, with emphasis in Software Reuse, Software Engineering Education, Software Visualization and Ecosystems. She has over 200 technical papers published in national and international conferences and journals, besides book chapters. She is a member of the Brazilian Society of Computer Science (SBC) and the program committee of various national and international conferences, and also co-editor-in-chief of the Springer Journal of Software Engineering Research and Development (JSERD).

Tutorial 1: Formal Aspects of Software Architecture

Antónia Lopes [1]

[1]Universidade de Lisboa, PT

mal@di.fc.ul.pt

1. Abstract

In the last two decades, Software Architecture has become an important sub-discipline of Software Engineering. Architecture is now a widely-accepted conceptual basis for the development of large and complex software systems, the importance of architectural decisions in the ability of a system to meet its non-functional requirements being widely recognised. At an architectural level, the design of a system involves deciding on how the system is to be structured in terms of components and connectors. Representing those decisions, even if using informal box-and-line drawings, contributes to understanding the system and to reasoning about runtime quality attributes such as performance, reliability, availability and security. Such architectural descriptions become more useful and relevant when expressed in a formal notation equipped with a semantics that enables formal analysis. In this talk I will discuss the role of software architecture, and describe the progress that has been made in formal modelling and analysis of software architectures as well as on formal foundations of software architecture.

2. Short bio

Antónia Lopes is Associate Professor in the Department of Informatics of the University of Lisbon, Faculty of Science, Portugal, since March 2006. She received a Ph.D. in Informatics at the University of Lisbon in 1999. Her research interests are in the area of formal methods for software engineering. These include mathematically based techniques for the specification, modelling and analysis of various types of software intensive systems, namely service-oriented systems and self-adaptive systems. She was the program committee co-chair of Fundamental Approaches to Software Engineering 2007 (FASE 2007) and 11th IFIP WG 6.1 International Conference on Formal Techniques for Distributed Systems (FMOODS/FORTE 2009) and ASAAS 2011 (First Workshop on Assurances for Self-Adaptive Systems). She is member of the Editorial Board of Academic Editors for PeerJ Computer Science.

Tutorial 2: A Theory of Modularity for Automated Software Design

Don Batory [1]

[1]University of Texas, USA

dsb@cs.utexas.edu

1. Abstract

Automated Software Development (ASD) uses technologies to develop customized programs automatically and compositionally from modules. The foundations of ASD are domain-specific algebras, where each program in the target domain maps to a unique expression, and modules are expression terms. Programs are optimized automatically using algebraic identities among module compositions. This tutorial traces the history of ASD and presents a general theory of modularity for ASD that follows from its tenets.

2. Short bio

Don Batory holds the David Bruton Centennial Professorship in the Department of Computer Science at The University of Texas at Austin. He received a B.S. (1975) and M.Sc. (1977) degrees from Case Institute of Technology, and a Ph.D. (1980) from the University of Toronto. He was a faculty member at the University of Florida in 1981 before he joined the University of Texas in 1983. He was Associate Editor of IEEE Transactions on Software Engineering (1999-2002), Associate Editor of ACM Transactions on Database Systems (1986-1992), member of the ACM Software Systems Award Committee (1989-1993; Committee Chairman in 1992), Program Co-Chair for the 2002 Generative Programming and Component Engineering Conference. He is a proponent of Feature Oriented Software Development (FOSD) and with colleagues (and former students) has recently authored a textbook on the topic. Since 1993, he and his students have written 11 Award Papers for their work in automated program development. He and Lance Tokuda were awarded the Automated Software Engineering 2013 Most Influential Paper Award on their work on program refactorings.

Tutorial 3: Model-driven Development

Gregor Engels [1]

[1]Universität Paderborn, DE

engels@uni-paderborn.de

1. Abstract

Model-driven resp. model-based approaches have become the de facto standard to develop, to evolve, to migrate, or to modernize complex software systems. In order to do so, modeling languages are used, which might be general-purpose modeling languages (as e.g. UML) or domain-specific modeling languages (DSML). The syntax of those modeling languages is defined by meta-modelling techniques, while the semantics is defined directly by transformation techniques or indirectly by translating models into appropriate semantic domains. During the construction process, models are stepwise refined. This can be done manually by a modeler or automatically by model transformations. These are formally specified by using a (domain-specific) model transformation language (as e.g. QVT, ATL). High quality of models can be achieved by e.g. deploying design patterns during a forward construction process, by mining techniques during a backward construction process, or by model checking techniques during an afterwards analytical process. The talk gives an overview on the state-of-the-art of model-driven development, its industrial acceptance as well as an overview on open research questions.

2. Short bio

Gregor Engels received his PhD in Computer Science in 1986 from the University of Osnabrück, Germany. Between 1991 and 1997 he held the position of Chair of Software Engineering and Information Systems at the University of Leiden, The Netherlands. Since 1997, he is Professor of Informatics at the University of Paderborn, Germany. Currently, he is also director of two technology transfer labs at the University of Paderborn, the C-LAB, a joint venture together with ATOS, and the s-lab – Software Quality Lab, where overall more than 50 PhD students do joint research with industrial partners. His research interests are in the area of model-driven software development, software architecture, and software quality assurance. He has published more than 200 papers in scientific journals, as book contributions or articles at international conferences and workshops. He teaches in Bachelor, Master and PhD programmes since more than 30 years. He also gives regularly tutorials and seminars on recent technology topics at scientific conferences and industrial events.

Tutorial 4: Abstractions for Validation

Sebastián Uchitel [1]

[1]Universidad de Buenos Aires, AR and Imperial College London, UK

[2]Department of Computer Science – University of Durham
Durham, U.K.

suchitel@dc.uba.ar

1. Abstract

Validating (as opposed to verifying) software artefacts is notoriously difficult. In this talk I will discuss the general problem of validation and then focus on a specific kind of artefacts, those that define a set of operations and impose restrictions to the ordering on which they have to be invoked. I will discuss Enabledness Preserving Abstractions (EPAs) and show that they are a concise representations of the behaviour space for such artefacts. I will also show how these abstractions can be built and how they may be used to support some programming tasks. Finally, I will discuss limitations of these abstractions and opportunities for improvement.

2. Short bio

Sebastian Uchitel is a professor at University of Buenos Aires and Imperial College London. He currently also sits on the board of the national argentine oil company, YPF. He received his undergraduate computer science degree from University of Buenos Aires and his Phd in Computing from Imperial College London. His research interests are in behaviour modelling, analysis and synthesis of requirements and design for software-intensive systems. He was associate editor of IEEE Transactions on Software Engineering and is currently associate editor of the Requirements Engineering Journal and the Science of Computer Programming Journal. He was program co-chair of the 32nd IEEE/ACM International Conference on Software Engineering (ICSE 2010) and will be general chair of the same conference in 2017, when it will be held in Buenos Aires, Argentina.

Tutorial 5: Experimental Software Engineering – The Pathway for Achieving Evidence

Guenther Ruhe[1]

[1] University of Calgary, CA

ruhe@ucalgary.ca

1. Abstract

As for any technology, the same is also true for software engineering technologies (tools, techniques, processes): When and where is it applicable and when and where is it not? How good does it work? and how much time and effort is needed to run it? There is no easy way to accumulate this knowledge. In this mini-tutorial, the empirical research paradigm is motivated and explained as a means to answer the above questions. Key steps of conducting experiments are described: (i) experimental context, (ii) experimental design, (iii) run the experiment, (iv) analysis of results, (v) presentation and (vi) interpretation of results. As series of examples is given to illustrate the value added for researchers and users of an empirically validated software technology.

2. Short bio

Günther Ruhe is a Professor at the University of Calgary in Canada. He received a doctorate rer. nat. degree in Mathematics with emphasis on Operations Research from Freiberg University and a doctorate habil. nat. degree (Computer Science) from University of Kaiserslautern (Germany). From 1996 until 2001, he was the deputy director of the Fraunhofer Institute for Experimental Software Engineering Fh IESE. Since 2007, he serves as an Associate Editor of the Journal of Information and Software Technology, published by Elsevier. His main research interests are in the areas of Product and Project Management, Data Analytics, Decision Support in Requirements Engineering and Empirical Software Engineering. Günther is the Founder and CEO of Expert Decisions Inc., a University of Calgary spin-off company created in 2003.

Tutorial 6: Self-Adaptive Systems

David Garlan [1]

[1] Carnegie Mellon University, USA

garlan@cs.cmu.edu

1. Abstract

The increasing use of computing systems in every facet of our everyday lives raises a number of challenges for software engineering. In particular, one of the most important requirements for today's systems is high availability – even in the presence of faults, changing environmental conditions, and attacks. To address these requirements we need to be able to build systems that take more control over their own dependability, security, and usefulness – automating many of the tasks that now lead to system failures and that require computing experts and administrators to manage. This has led to a new sub-field of software engineering and systems design, sometimes termed Autonomic Computing, Self-healing Systems, or Self-Adaptive Systems. In this talk I describe this emerging field and recent advances that allow us to address various engineering challenges, including (a) the ability to support self-healing through architectural models and automated repair, (b) new techniques for diagnosing faults at run-time with applications to manufacturing control systems, (c) the ability to support self-securing systems, and (d) the ability to reason about human-in-the loop systems.

2. Short bio

David Garlan is a Professor of Computer Science and Director of Software Engineering Professional Programs in the School of Computer Science at Carnegie Mellon University. His interests include software architecture, self-adaptive systems, formal methods, and cyber-physical systems. He is considered to be one of the founders of the field of software architecture, and, in particular, formal representation and analysis of architectural designs. He is a co-author of two books on software architecture: "Software Architecture: Perspectives on an Emerging Discipline", and "Documenting Software Architecture: Views and Beyond." In 2005 he received a Stevens Award Citation for "fundamental contributions to the development and understanding of software architecture as a discipline in software engineering." In 2011 he received the Outstanding Research award from ACM SIGSOFT for "significant and lasting software engineering research". He is a Fellow of the Association for Computing Machinery (ACM) and the Institute of Electrical and Electronic Engineers (IEEE).

Tutorial 7: Recommender Systems for Software Engineering

Gail Murphy [1]

[1]University of British Columbia, CA

murphy@cs.ubc.ca

1. Abstract

Software developers must interact with large amounts of different types of information and perform many different activities to build a software system. To ease the finding of information and hone workflows, there has been growing interest in building recommenders that are intended to help software developers work more effectively. Building an effective recommender requires a deep understanding of the problem that is the target of a recommender, analysis of different aspects of the approach taken to perform the recommendations and design and evaluation of the mechanisms used to deliver recommendations to a developer. In this lecture, I will discuss these steps using case studies of recommendation systems to support software engineering activities.

2. Short bio

Gail Murphy is a Professor in the Department of Computer Science and an Associate Dean (Research & Graduate Studies) in the Faculty of Science at the University of British Columbia. She is also a co-founder and currently Chief Science Offer at Tasktop Technologies. She received a B.Sc. from the University of Alberta and M.Sc. and Ph.D. degrees from the University of Washington. Her current research interests are in reducing the information overload and improving the work days of knowledge workers, including software developers.

Why, When, and How to Write up Your Research Work

Simone Barbosa [1]

[1] Pontifical Catholic University of Rio de Janeiro (PUC-Rio), BR

simone@inf.puc-rio.br

1. Abstract

Academia communicates its advances through scientific publications. It is through those papers that we learn who is doing what, why, and how, i.e., the scientific discussions revolving around important societal and technological issues. It is essential for all researchers to participate in this discussion, and the prime means to do so is by writing scientific papers. In today's fast-paced world, we strive to find balance between rushing to write half-baked ideas and alienating ourselves from the scientific discussions for too long because we have not yet achieved the desired results. In this event, I will present some thoughts on academic writing from prominent scholars to promote discussions on why and how to write up your work before, during, and after you conduct your research.

2. Short bio

Simone Diniz Junqueira Barbosa is Associate Professor of the Department of Informatics of the Pontifical Catholic University of Rio de Janeiro (PUC-Rio), where she teaches, advises and conducts research on Human-Computer Interaction (HCI). Since 2012, she has taught a graduate course on Research Design and Academic Writing. Level 2 researcher in CNPq (National Council for Scientific and Technological Development in Brazil), she has obtained research funding from several national and international agencies, such as CNPq, FAPERJ, Microsoft Research, and Hewlett-Packard. She has coordinated or participated in the program committee of both national and international conferences, including CHI, INTERACT, EICS, and IHC. She was the Brazilian Computer Society's representative in IFIP TC 13 from 2008 to 2013, when she became an expert member and later Vice-chair for Working Groups and Special Interest Groups. In 2009 she has joined the Editorial Board of Interacting with Computers (formerly published by Elsevier, now by Oxford University Press); in 2011 the Advisory Board of IxD&A; in 2012 the editorial boards of Springer's HCI and CCIS book series; and in 2013 the editorial board of Springer's OpenAccess Journal of Interaction Science. Since 2014 she is the chair of the Special Commission for HCI of the Brazilian Computer Society. She has authored, together with Bruno Santana da Silva (UFRN), the book "Interação Humano-Computador," published by Elsevier in the Brazilian Computer Society series.

Software Engineering Research at UFRGS

1. Abstract

In this panel, groups of professors of the Graduate Program in Computer Science (PPGC) at UFRGS that work on Software Engineering will present an overview of their current research work and discuss views on the future needs in their fields. Also, indirectly, it serves as a mean for students to gain more detailed knowledge of a set of software engineering topics, such as software architecture, software visualization, software verification, formal methods, business process management, and agent-oriented software engineering. Prof. Dr. Daltro Nunes

2. Panelists

Daltro José Nunes is the head of the Secretariat of Institutional Assessment (SAI) and an invited professor at UFRGS. He has a doctor degree in Informatic from Institut für Informatik, Universität Stuttgart, Germany, a M.Sc. degree in Informatics from PUC, Rio de Janeiro, Brazil, and graduated in Electric Engineering at Escola de Engenharia, UFRGS, Porto Alegre, Brazil. His main research areas are formal methods and theory of computation.

Érika Cota is an associate professor in the Computer Science Department at Federal University of Rio Grande do Sul (UFRGS). Her research interests include software testing, testing and design for test of embedded systems, and hardware testing. Dr. Cota has a BS in computer science from the Federal University of Minas Gerais (UFMG), and an MS and a PhD in computer science from Federal University of Rio Grande do Sul.

Ingrid Nunes is a Professor Adjunto (Associate Professor) of the Instituto de Informática at the Federal University of Rio Grande do Sul (UFRGS), Porto Alegre, Brazil, and the head of the Prosoft research group. She completed her undergraduate studies in Computer Science at UFRGS (2006), obtained her Master's degree in Informatics at the Pontifical Catholic University of Rio de Janeiro (2009), and obtained her Doctor's degree in Informatics at the Pontifical Catholic University of Rio de Janeiro (2012). Her phd was in cooperation with King's College London (UK), under a sandwich Ph.D. programme of one year, and with University of Waterloo (Canada), with three three-month research visits. She was also a post-doc researcher at PUC-Rio in the Software Engineering Laboratory (LES) (2012), and has experience in the industry, where she worked as a software developer from 2005 to 2007. She is a section editor of the Scientific Initiation Magazine (REIC). Her main research areas are software architecture, agent-oriented Software Engineering, and model-driven Development.

Leandro Krug Wives – Associate professor at Federal University of Rio Grande do Sul (UFRGS), where he develops research in the fields of information retrieval, recommender systems, (semantic) web services discovery, integration and matchmaking. Leandro is particularly interested in Architectures and Frameworks for Web 2.0 Application Development. He got a Ph.D degree in Computer Science from

(UFRGs). He participated in many research projects in collaboration with France, Spain and Portugal, the most recent involving Context-Awareness, Business Process Management, Ontology, Web services and Cloud Computing.

Leila Ribeiro is a Full Professor at the Department of Theoretical Informatics of INF/UFRGS. She received the Bachelor 1988 and MSc Degrees in Computer Science at UFRGS in 1888 and 1991, respectively, and the PhD Degree in Computer Science at the Technical University of Berlin in 1996. She coordinated several scientific projects involving research institutions from Brazil and abroad, and is the leader of the VeriTeS Group (Group on Verification, Validation and Test of Computational Systems). She is a member of the IFIP Working Group 1.3 (Foundations of System Specification).

Her main research interests are modelling and analysis of complex systems, models of computation, formal specification and verification, concurrent systems and bioinformatics.

Lucinéia Heloisa Thom is an associate professor with Informatics Institute at Federal University of Rio Grande do Sul, UFRGS (Brazil). From 2010 to 2011 she was a visiting Scientist at the University of Grenoble. Before, she was a visiting Scientist at the University of Ulm (2007-2009). She received her Bachelor's in Computer Science from the University of Santa Cruz do Sul, Brazil (1999); her Master's in Computer Science from UFRGS (2002); and her PhD in Computer Science from UFRGS (2006). She developed part of her thesis research at the Institute for Parallel and Distributed Systems of the University of Stuttgart (2004-2005). Her research interests are in the area of Business Process Management and workflow with a special focus on workflow patterns, process design, IT support for healthcare processes and ontology. In these fields she has many published papers and is involved in several PCs. She has also participated in the organization of several events such as the Brazilian largest workshop in Business Process Management (WBPM 2013) and the 5th International Workshop on Process Model Collections: Management and Reuse (PMC-MR'14) from the most important conference in Process Management the BPM 2013. She is co-chair of the First Latin American School on Business Process Management (LAS-BPM 2015).

Lucio Mauro Duarte is a Senior Lecturer at the Department of Theoretical Computer Science of the Institute of Informatics of the Federal University of Rio Grande do Sul (UFRGS), where he teachs Algorithms and Software Verification. He holds a Ph.D. degree in Computing (Imperial College London, University of London) and his main research areas are Validation and Verification of Systems, Software Testing, and Software Modelling. His web page is www.inf.ufrgs.br/~lmduarte.

Luis Lamb is Professor and Dean of the Institute of Informatics (2011-2015), Federal University of Rio Grande do Sul. He was Deputy Dean of the Institute of Informatics at UFRGS from August 2006 to October 2011. He holds a Ph.D. in Computing Science from Imperial College London (2000), the Diploma of the Imperial College, MSc by research (1995) and BSc in Computer Science (1992) from the Federal University of Rio Grande do Sul, Brazil. In 2010 he received the MIT Executive Certificate in Strategy and Innovation and in 2014 he received the MIT Executive Certificate in Management and Leadership. He is Honorary Visiting Fellow at the Department of Computing, City University London and was Visiting Research Fellow, Abductive Systems Group, Department of Philosophy, University of British Columbia. His

research interests include Logic in Computer Science, Social Computing and Formal Methods in Embedded Software.

Marcelo S. Pimenta is an Associate Professor at Institute of Informatics, Federal University of Rio Grande do Sul (UFRGS), in Brazil. He received his PhD in Informatique at Université Toulouse 1, France, in 1997 and the bachelor and master's degree in Computer Science at UFRGS in 1988 and 1991, respectively. Since 1998, he is member of a multidisciplinary research group at UFRGS working with topics in Human-Computer Interaction, Software Engineering, and Computer Music with emphasis in the integration of these areas. Member and founder of the Ubiquitous Music Group (g-ubimus), currently his research focuses on ubiquitous music, collaborative design, adaptive interfaces, digital governance and user-centered software engineering.

Rodrigo Machado is a Professor Adjunto at the Institute of Informatics of the Federal University of Rio Grande do Sul since 2012. Rodrigo received its Doctorate from the Postgraduate Program in Computer Science / UFRGS. His research interests include functional programming, semantics of programming languages, type systems and formal methods in software development, in particular algebraic graph rewriting. He is currently interested in the development of theory and tools to specify software evolution, in particular for rule-based software models.

Perspectives, Opportunities and Challenges of Software Engineering in the Industry

1. Abstract

The development of modern software systems is a challenging task, given that software is nowadays everywhere and its complexity never stops growing. In this panel, representatives of internationally recognised companies that produce software and face software engineering challenges will share their view in this context, discussing challenges and opportunities. Attendees will learn about the current state of software industry, including technologies, challenges, and best practices.

2. Panelists

Antônio Gomes (Chief Architect, HP Brazil R&D) is Chief Architect at HP Brazil R&D, with 33+ years of experience in R&D and Consulting, where he is responsible to provide overall technical leadership and promoting innovation across the organization. He received the Electrical Engineering degree from Universidade Federal do Rio Grande do Sul (UFRGS) and an Executive MBA, General Business Management, from ESPM Porto Alegre.

Diego Nobre (Software Architect, ADP Labs): Software Architect at ADP Labs with 18 years of experience developing enterprise-scale systems in varied industries such as Banking, Communications, Media, Computer Manufacture and HCM. Bachelor's degree, Computational and Applied Mathematics – Universidade Federal do Rio Grande do Sul / UFRGS.

Marcelo Blois (CoE leader, GE Research) has over 18 years of experience on research in different aspects of software engineering, including software process improvement, software architectures and middleware, software engineering for multi-agent systems, and software reuse. He joined GRC Rio in December 2011 as CoE leader, helping the startup of the new technology center in the area of Systems Integration. Marcelo woked as a professor in Pontifical Catholic University of Rio Grande do Sul (PUCRS) for 10 years where he coordinated the Intelligent Systems Engineering Research group and the Systems Engineering Research Center. During this time Marcelo advised over 20 Master students and 3 PhD students. He managed different research initiatives with local and global companies being responsible for budgeting negotiation and project management. From 2002 to 2011 Marcelo worked in applied research projects with Dell Computers helping the company in different process improvement initiatives in its Software Development Facility in Brazil. Prior to that Marcelo had a software development company in Rio where he played the role of Executive Director. He received his PhD and MS in computer science from the Pontifical Catholic University of Rio de Janeiro (PUC-Rio) and his BS in computer science from the Federal University of Rio de Janeiro (UFRJ).

Roberto Petry (IT Director, Dell) works as IT Director at Dell where he is globally responsible for Infrastructure Management Project Delivery leading a team distributed in several countries. He has Master Degree in Computer Science at UFRGS, with 25+

years' experience in IT with focus in Database, Project Management, Software Development and IT Governance. He is PMP and ITIL Foundations certified. He teaches at Graduation and Post-Degree courses at Unilasalle, Ulbra and PUC University. He was the former President of the Open Technology Committee at American Chamber of Commerce in Porto Alegre and past Regional and National President of User Association Group (SUCESU), acting now as council at PMI-RS & SUCESU-RS.

Software Engineer: Industry or Academia?

1. Abstract

Students interested in software engineering have many alternatives in their professional careers. They can work in the industry after they get a bachelor degree in Computer Science (or related courses). They can pursue a Master or Doctorate degree. They can become research scientists in leading companies or professors at universities. In this panel, panelists will discuss about these different alternatives, and what is required for each of them. Examples of questions to be discussed are: is a Bachelor degree enough to work in the Industry? Those who get a Master or Doctorate degree should stay in academia or there are positions available in Latin America where research can be conducted?

2. Panelists

Daniel Wobeto (TRE-RS): CIO at the Tribunal Regional Eleitoral do Rio Grande do Sul since 2007, Computer Science Bachelor at UFRGS in 1993, Law Bachelor in 2001, TI Administration MBA in 2009, member of Urna Eletronica's Ecosystem Workgroup, responsible for requirements specification of systems related to voting process in Brazilian elections.

Eduardo Arruda (VP of ASSESPRO-RS) received the BSc and MSc degrees in Computer Science at the Universidade Federal do Rio Grande do Sul (UFRGS). He is CEO of Bluetterfly, which is company dedicated to develop innovative business. Nowadays it is involved with the conception of a cloud services plataform with a focus on secure storage of digital content. He is a professor at the Faculty of Informatics of PUC-RS since 1994. He was an associated and Director of Bussines Development of uMov.me, a company of mobile technology considered one of the five most innovative of Brazil by Gartner Group. He was also CIO of Justice Law Departament of Rio Grande do Sul, coordinating the project of infomatization of this court, the first in the country to use digital certification in the signment of judgments. Nowadays, he is Vice-President of Articulation of ASSESPRO-RS, President of the Deliberative Board of SUCESU-RS, Director of Market Relationship of SEPRORGS and member of the Administrative Board of SOFTSUL. He is also dedicated to actions to encourage innovative entrepreneurship.

Luigi Carro (Head of the Graduation Program in Computer Science (PPGC), UFRGS, BR) was born in Porto Alegre, Brazil, in 1962. He received the Electrical Engineering and the MSc degrees from Universidade Federal do Rio Grande do Sul (UFRGS), Brazil, in 1985 and 1989, respectively. From 1989 to 1991 he worked at ST-Microelectronics, Agrate, Italy, in the R&D group. In 1996 he received the Dr. degree in the area of Computer Science from Universidade Federal do Rio Grande do Sul (UFRGS), Brazil. He is presently a full professor at the Applied Informatics Department at the Informatics Institute of UFRGS, in charge of Computer Architecture and Organization courses at the undergraduate levels. He is also a member of the Graduation Program in Computer Science at UFRGS, where he is co-responsible for courses on

Embedded Systems, Digital signal Processing, and VLSI Design. His primary research interests include embedded systems design, validation, automation and test, fault tolerance for future technologies and rapid system prototyping. He has advised more than 20 graduate students, and has published more than 150 technical papers on those topics. He has authored the book Digital systems Design and Prototyping (2001-in Portuguese) and is the co-author of Fault-Tolerance Techniques for SRAM-based FPGAs (2006-Springer), Dynamic Reconfigurable Architectures and Transparent optimization Techniques (2010-Springer) and Adaptive Systems (Springer 2012). In 2007 he received the prize FAPERGS – Researcher of the year in Computer Science. His most updated resume is located in http://lattes.cnpq.br/8544491643812450.

Luís Lamb (Dean of the Institute of Informatics, UFRGS, BR) is Professor and Dean of the Institute of Informatics (2011-2015), Federal University of Rio Grande do Sul. He was Deputy Dean of the Institute of Informatics at UFRGS from August 2006 to October 2011. He holds a Ph.D. in Computing Science from Imperial College London (2000), the Diploma of the Imperial College, MSc by research (1995) and BSc in Computer Science (1992) from the Federal University of Rio Grande do Sul, Brazil. In 2010 he received the MIT Executive Certificate in Strategy and Innovation and in 2014 he received the MIT Executive Certificate in Management and Leadership. He is Honorary Visiting Fellow at the Department of Computing, City University London and was Visiting Research Fellow, Abductive Systems Group, Department of Philosophy, University of British Columbia. His research interests include Logic in Computer Science, Social Computing and Formal Methods in Embedded Software.

Marcelo Blois (CoE leader, GE Research) has over 18 years of experience on research in different aspects of software engineering, including software process improvement, software architectures and middleware, software engineering for multi-agent systems, and software reuse. He joined GRC Rio in December 2011 as CoE leader, helping the startup of the new technology center in the area of Systems Integration. Marcelo woked as a professor in Pontifical Catholic University of Rio Grande do Sul (PUCRS) for 10 years where he coordinated the Intelligent Systems Engineering Research group and the Systems Engineering Research Center. During this time Marcelo advised over 20 Master students and 3 PhD students. He managed different research initiatives with local and global companies being responsible for budgeting negotiation and project management. From 2002 to 2011 Marcelo worked in applied research projects with Dell Computers helping the company in different process improvement initiatives in its Software Development Facility in Brazil. Prior to that Marcelo had a software development company in Rio where he played the role of Executive Director. He received his PhD and MS in computer science from the Pontifical Catholic University of Rio de Janeiro (PUC-Rio) and his BS in computer science from the Federal University of Rio de Janeiro (UFRJ).

Introdução ao Planejamento e à Análise Estatística de Experimentos em Engenharia de Software

Lisiane Selau [1]

[1] Departamento de Estatística, Universidade Federal do Rio Grande do Sul (UFRGS)
Porto Alegre, BR

lisianeselau@gmail.com

1. Resumo

Grande parte do conhecimento científico é desenvolvido por meio de evidências empíricas. Tais evidências, em geral, são oriundas dos resultados de experimentos devidamente planejados e analisados. Experimento é uma pesquisa planejada em que mudanças relevantes são feitas nas variáveis de entrada de um processo de modo a identificar as razões para mudanças na resposta (saída do processo), tendo por objetivo tomar decisões (fazer uma recomendação). Para um experimento ser o mais eficiente possível, um procedimento científico para planejá-lo deve ser empregado. Ao planejar um experimento o pesquisador tem que ter bem claro quais são os objetivos da pesquisa, as questões a serem respondidas, e as hipóteses a serem testadas. Nesse sentido, devem ser empregados procedimentos estatísticos de análise de dados apropriados aos objetivos do experimento, que sejam consistentes e coerentes com o planejamento experimental adotado e com o correspondente modelo estatístico estabelecido.

2. Biografia Resumida

Professora Adjunta do Departamento de Estatística da Universidade Federal do Rio Grande do Sul (UFRGS). Também é professora convidada do Programa de Pós-Graduação em Fitotecnia da UFRGS. É Bacharel em Estatística pela UFRGS (2000), Licenciada em Estatística pela UFRGS (2002), Mestre em Engenharia de Produção pela UFRGS (2008) e Doutora em Administração na área de Sistema de Informação e Apoio à Decisão pela UFRGS (2012). Tem experiência na área de Modelagem para Gestão do Risco de Crédito, Planejamento e Análise de Experimentos e Educação Estatística.

Vivencial da Metodologia Ágil SCRUM

Pablo Schoeffel [1]

[1] Departamento de Sistemas de Informação, Universidade do Estado de Santa Catarina (UDESC), BR

pablo.schoeffel@udesc.br

1. Resumo

O mini curso tem o objetivo de demonstrar os princípios e valores ágeis, aplicando o processo e técnicas da metodologia SCRUM. Com isso, os participantes poderão, além de conhecer os conceitos, vivenciá-los em dinâmicas e práticas para compreendê-los melhor.

2. Biografia Resumida

Pablo Schoeffel, é mestre em Computação Aplicada na UNIVALI (SC) na área de Engenharia de Software. Possui graduação em Ciências da Computação (FURB) e Especialização em Desenvolvimento Web e E-Commerce (ICPG). Atua desde 2002 na área de desenvolvimento de software, como: programador, analista de sistemas, analista de negócio, gerente de projetos e coordenador de equipe. Atua como professor em cursos de tecnologia desde 2007. Atualmente é professor efetivo do Departamento de Engenharia de Software da Universidade do Estado de Santa Catarina (UDESC) e consultor na área de Engenharia de Software e Gerência de Projetos. Incentivou e iniciou a utilização de processos ágeis, e atuou como SCRUM Master em algumas empresas em que trabalhou. Leciona a disciplina de Métodos Ágeis para cursos de pós-graduação desde 2011.

DUPLICIDADE DE INFORMAÇÃO E FERRAMENTAS PARA LIMPEZA DOS DADOS

Carlos Eduardo O. Santos[1], Sergio Martins Fernandes[1]

[1]Departamento de Sistemas e Computação - Universidade Salvador – (UNIFACS) – Salvador – BA - Brasil

eduardo.maceio@yahoo.com.br, sergio.martins@pro.unifacs.br

Abstract. *The purpose of this paper is to discuss about the quality of information stored in databases, regarding the issue of duplicates. To promote quality assurance of information, some techniques and tools are created from criteria, characteristics and attributes that direct the obtaining of this quality. This dimension will be the non-duplicated data, exposing the solutions found for the data cleaning process. Thus it is proposed deduplication technique and some existing tools that perform this process in cleaning and organization of data, aiming to improve the quality of information.*

Resumo. *O objetivo deste artigo é discutir sobre a qualidade da informação armazenada em bancos de dados, referente à questão das duplicidades. Para promover a garantia da qualidade da informação, algumas técnicas e ferramentas são criadas a partir de dimensões, critérios, características e atributos que direcionam a obtenção dessa qualidade. A dimensão tratada neste trabalho será a de não duplicidade de dados, expondo soluções encontradas para o processo de limpeza dos dados. Para tal é proposto a técnica de deduplicação e algumas ferramentas existentes que realizam este processo na limpeza e organização dos dados, objetivando a melhoria da qualidade da informação.*

1. INTRODUÇÃO

Os problemas na aquisição da informação podem gerar danos irreparáveis. Faz-se necessário para os ambientes corporativos ter um conhecimento aprofundado da importância do gerenciamento deste conteúdo e das tecnologias disponíveis que minimizam essa suscetibilidade de erros, de forma que a qualidade da informação que circula no seu ambiente, não seja passível de vulnerabilidades, visto que afeta diretamente no processo de tomada de decisão, como também no gerenciamento da qualidade do serviço oferecido por essa organização.

Questões envolvendo a qualidade de dados e informações podem variar desde dificuldades de natureza técnica, por exemplo, integração de fontes de dados diferentes, até dificuldades não técnicas, por exemplo, a falta de uma estratégia integrada em toda a organização para assegurar o direito das partes interessadas de acessar a informação certa, no formato certo, na hora e lugar certo (MADNICK et al, 2009).

Os dados são os elementos que servem de base para a formação de juízos ou servem para a resolução de problemas. Um dado é apenas um índice, um registro, uma manifestação objetiva, passível de uma análise subjetiva, isto é, existe a interpretação da pessoa para a sua manipulação. Em si, os dados têm pouco valor. Todavia, quando classificados, armazenados e relacionados entre si, os dados permitem a obtenção da informação.

A informação não é simplesmente um dado, sequências de números, listas de endereços, ou resultados de testes armazenados num computador. A informação é o produto de processos de organização dos dados. Assim, segundo Chiavenato (2008), os dados isolados não são significativos e não constituem informação. Já a Informação, apresenta significado e intencionalidade, aspectos que a diferenciam do conceito de dado.

A informação é um ativo (NBR ISO/IEC 27002, 2005) que, como qualquer outro é essencial para os negócios da organização e, consequentemente, necessita ser adequadamente protegida, ser refinada e analisada para que decisões importantes sejam tomadas, por todos que a utilizem, nos mais diversos níveis da hierarquia da empresa. Entende-se por ativos de informação, todos os tipos de informação que uma empresa possui, como arquivos e sistemas, que possuam valor, demandando necessidades em termos de proteção (NBR ISO/IEC 27002, 2005).

Para promover a garantia da qualidade da informação, algumas técnicas e ferramentas são criadas a partir de critérios, características e atributos que dimensionam a obtenção dessa qualidade. A ideia a ser exposta nesse artigo está direcionada a uma questão especifica que é a duplicidade dos dados.

Para tal é apresentado na segunda seção uma exposição mais abrangente sobre a qualidade da informação (QI), problemas causado pela má qualidade, dando ênfase a necessidade da limpeza dos dados, mais especificadamente nas ocorrências de duplicidade. A terceira seção traz a abordagem da técnica de deduplicação, introduzindo algumas ferramentas existentes que realizam o processo de limpeza e organização dos dados, objetivando a melhoria da QI.

2. QUALIDADE DA INFORMAÇÃO

A informação pode ser vista como um bem, com dimensões (atributos) de qualidade que podem ser medidas. Conforme Wand e Wang (1996), a qualidade da informação é um conceito multidimensional, e assim como um produto físico tem dimensões de qualidade associadas, um produto de informação também tem dimensões de qualidade da informação. Uma vez identificados os atributos, a qualidade da informação pode ser gerenciada (MILLER, 2001).

Inúmeras classificações referentes às dimensões de qualidade, são encontradas na literatura. É possível apontar um conjunto comum a todos das dimensões de qualidade da informação (QI), incluindo acuracidade (acuracy), integridade (completeness), consistência (consistency) e temporalidade (timeliness), que constituem o foco da maioria dos autores(CATARCI AND SCANNAPIECO, 2002). Entretanto não existe qualquer consenso geral sobre qual conjunto de dimensões definem a QI, ou sobre o significado exato de cada dimensão. Dentre estas, a referência adotada é colocada pelos autores WANG(1996), PIPINO(2002), BATINI(2009) e

GAMBLE(2011), que utilizam as dimensões de Acurácia, Credibilidade (ou Confiabilidade), Completude, Temporalidade, Atualidade, Precisão, Ausência de Dados Duplicados e Facilidade de Acesso, para propiciar a medição da qualidade da informação utilizada nos bancos de dados. Mais especificadamente ao interesse deste trabalho, a dimensão tratada será a de **não duplicidade de dados**. Este foco não desfavorece as outras dimensões, nem tampouco sua escolha se justifica por nível de importância, mas se torna relevante a continuidade da pesquisa do mestrado que tem como objetivo a deduplicação de dados, entende-se também, que mesmo restringindo a só esta dimensão, outras estão implicadas no processo. Dessa forma vamos expor considerações referentes à duplicidade de informação e ferramentas existentes no processo de limpeza dos dados.

2.1. Problemas da Qualidade da Informação

Problemas relacionados à qualidade da informação de uma organização, não necessariamente estão ligados a particularidades dos sistemas ou ao tempo de uso, como também a linguagem utilizada. Eles podem ser causados pelos próprios usuários, quando da inserção dos dados, como também em processos de análises mal sucedidas que são realizadas a partir de dados repetidos. Mesmo que esses erros sejam perceptíveis pelos usuários que lidam com o sistema, é difícil perceber a dimensão dessa problemática e, por conseguinte até que ponto isto afeta a organização, tanto no caráter financeiro, como em sua credibilidade. Esses problemas são indicativos da má qualidade dos dados.

Para ter dados precisos é necessário um programa formal de garantia da qualidade de dados com um componente específico dedicado à precisão(OLSON, 2003). Normalmente, isso envolve processos de atualização, padronização, detecção e limpeza dos registros, por meio de ferramentas específicas, para criar uma visão única dos dados, mesmo se ele estiver armazenado em vários sistemas distintos.

2.2. Limpeza dos dados

Não importa o quão eficiente seja o processo de entrada de dados, os erros ainda irão ocorrer e, por conseguinte, os dados sobre validação e correção não podem ser ignorados. A detecção de erros, validação e limpeza desempenham papéis importantes, especialmente com dados legados[1]. E, portanto, a prevenção de erros e limpeza de dados deve ser incorporada em uma política de gestão de dados.

Um ponto importante de limpeza de dados é a identificação das causas fundamentais dos erros detectados e usar essa informação para melhorar o processo de entrada de dados para evitar novas recorrências desses erros(RAHM and DO, 2000).

Uma abordagem de limpeza de dados deve satisfazer várias exigências. Em primeiro lugar, deve detectar e remover todos grandes erros e inconsistências, tanto em fontes de dados individuais e ao integrar múltiplas fontes.

A ocorrência dos problemas com a qualidade dos dados se dá tanto no nível de esquema como no nível de instância. Quanto ao nível de esquema, os problemas são

[1] Dados armazenados ao longo do tempo.

decorrentes da falta de um modelo adequado ou da aplicação de restrições de integridade específicas, como por exemplo as limitações do modelo de dados ou a má concepção do esquema. Quanto ao nível de instância dizem respeito aos erros e inconsistências que não podem ser prevenidos no nível de esquema como, por exemplo, erros de escrita, falta de valores, referências incorretas, entre outros. Fica exposto desta forma, a existência da necessidade de correção e limpeza destes dados em diversas tabelas dos bancos de dados, inclusive as tabelas da área financeira.

2.3. Duplicação de dados.

Lwin(2010) descreve a detecção de duplicação, como uma sub-tarefa importante de limpeza de dados, sendo, portanto, a tarefa de identificar múltiplas representações de um mesmo objeto do mundo real e necessário para melhorar a qualidade dos dados. Dessa forma entendemos por duplicidade como sendo uma medida de duplicação indesejada existentes num ou em vários sistemas para um determinado campo, registro ou conjunto de dados, gerados por erro de armazenamento da informação, ocasionando a existência da baixa qualidade dos dados.

Há muitos custos ocultos associados com registros duplicados. Assim, dados de má qualidade custam aos negócios de várias formas: desperdício e retrabalho, perda de oportunidades de receita, perda de negócios, etc.

Essa duplicação provoca sérios problemas na evolução do sistema como um todo, tais como, incoerência de uma mesma informação armazenada por dados duplicados, passíveis de ser alterados individualmente, o que pode provocar uma inconsistência entre os mesmos, e o custo de manutenção, que aumenta devido ao fato de uma mesma tarefa estar sendo realizada em dois ou mais processos distintos.

Este problema de detectar e remover entradas duplicadas em repositórios de dados é conhecido como deduplicação de registros (KOUDAS et al, 2006), mas também é denominado na literatura de limpeza de dados[2] (CHAUDHURI et al, 2003), pareamento de registros[3] (BHATTACHARYA, 2004); (FELLEGI and SUNTER, 1969); (KOUDAS et al, 2006), e casamento de registros[4] (VERYKIOS, 2003).

Cecchin (2010) traz as denominações deduplicação, resolução de referências e reconciliação. Mais especificamente, a deduplicação de registros em repositórios de dados consiste na identificação e remoção de registros que se referem ao mesmo objeto ou entidade do mundo real, ainda que apresentem estilos de escrita, grafias, tipos de dados ou esquemas diferentes.

A importância desse processo fica clara ao analisar o processo de integração de dados, este consiste em combinar diferentes representações de um objeto do mundo real em uma representação única, neste caso, a garantia da não duplicidade dos dados, é fator relevante.

A detecção de duplicidade é um problema difícil e não pode ser resolvido usando apenas casamentos exatos de atributos, pois há o problema de identificação,

[2] Do inglês *Data cleasing*.
[3] Do inglês *Record linkage*.
[4] Do inglês *Record matching*.

onde diferentes representações referem-se à mesma entidade. Desse modo, vamos utilizar o termo deduplicação para abordar essa técnica de limpeza de dados.

3. DEDUPLICAÇÃO DE DADOS

Esta técnica tem evoluído durante esta década recente, recebendo uma atenção ampla da academia e da indústria. Algumas pesquisas se concentram na abordagem pela qual dados redundantes possam ser mais reduzidos e outras investigam como fazer a deduplicação de dados em alta velocidade. A técnica consiste em eliminar dados duplicados, reduzir o espaço utilizado pelas réplicas durante o armazenamento dos dados, das cópias de dados para armazenamento secundário, ou no contexto de armazenamento de dados, melhorando, assim, a qualidade dos dados e a integração.

O termo deduplicação foi criado há vários anos pelos administradores de banco de dados, como uma maneira de descrever o processo de remoção de registros duplicados de um banco de dados, após a união de dois bancos de dados. Segundo Tavares (2003), a deduplicação de dados significa identificar registros duplicados em uma base de dados. Após a identificação desses registros é possível eliminar ou marcar o dado duplicado para controle. Para Dbdireto (2011) a deduplicação é um processo para a verificação, marcação e exclusão de registros com valores iguais em um banco de dados.

3.1. Técnicas de deduplicação

A primeira técnica utilizada de deduplicação executava em apenas uma única instância, por isso ficou conhecida como SIS (Single-Instance Storage), como objetivo dessa técnica é diminuir a quantidade de dados repetidos e melhorar o desempenho das aplicações, ela parte da ideia de manter apenas uma instância do arquivo e criar ponteiros para ser acessado pelos demais usuários sem que seja necessário ter uma cópia do arquivo para um usuário distinto, porém tal tecnologia possuía limitações, já que se o arquivo fosse modificado iria exigir que outro arquivo fosse armazenado com a alteração realizada, já que, essa tecnologia apenas é executada em nível de arquivo (DORNALA et al, 2010).

O algoritmo de deduplicação passou por uma evolução e começou a trabalhar com os dados em nível de blocos de dados, como pode ser visualizado na **Figura 1**, e não mais em nível de arquivo. A comparação feita em nível de blocos é mais especifica do que a feita em nível de arquivos, pois tem a possibilidade de analisar uma sequência de dados em baixo nível, podendo encontrar sequências idênticas, sendo capaz de eliminar vários gigabytes de dados repetidos do sistema de armazenamento.

A eliminação permite que o usuário execute o sistema mais rápido e mais eficiente já que não estará sobrecarregado com dados extras. Além de uma melhora perceptível relacionada ao tráfego de dados do sistema e consequentemente o aumento de espaço livre.

Os sistemas de grande porte se beneficiam dessa técnica para a limpeza e redução de espaço de armazenamento, já que possuem um crescimento no percentual de dados não estruturados gerados por aplicativos de colaboração, virtualização de servidores, imagens e demais aplicativos. Com o crescimento dos dados não

estruturados em ritmo exponencial o tempo necessário para uma ampliação do sistema de armazenamento é reduzido, assim como os custos de armazenagem aumentam. Por esse motivo, é notado o crescimento da utilização do algoritmo de deduplicação em ambientes corporativos.

Dessa forma, a deduplicação pode ser resumida em um processo de segmentação de cada pedaço dos dados que é processado, cada segmento é identificado e confrontado com os dados que já estão no sistema, caso o dado seja único então ele é armazenado em um disco, caso o dado esteja duplicado uma referência é criada em seu lugar apontando para o primeiro dado idêntico a este que teve entrada no sistema. Muitas vezes, os mesmos dados podem ser armazenados em mais de 50 locais diferentes em um sistema de armazenamento. Se cada dado tiver um byte de espaço, a deduplicação irá reduzir o espaço no armazenamento de cinquenta bytes para apenas um byte.

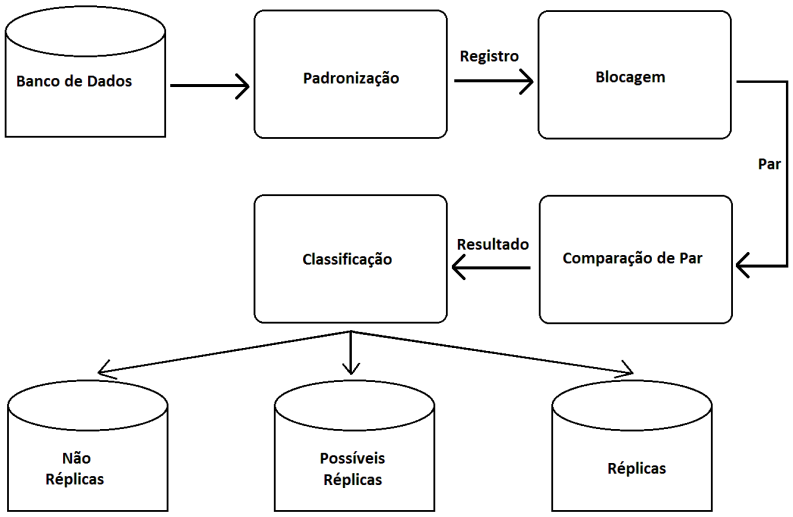


Figura 1: Processo de deduplicação

3.2. FERRAMENTAS DE DEDUPLICAÇÃO DE DADOS

Algumas ferramentas que realizam o processo de deduplicação de dados foram selecionadas e expostas abaixo. Não reduzindo com isso a existência de muitas outras soluções. As ferramentas apontadas são referenciadas nas fontes bibliográficas com uma maior frequência e também funcionam como indicativos para a construção de novas soluções. Também está demonstrado na **Tabela1** um comparativo de visão geral destas ferramentas.

3.2.1. FRIL

O Fine-grained Record Integration and Linkage (FRIL) (JURCZYK et al, 2008) é uma ferramenta de Record Linkage, desenvolvida pela Universidade de Emory (Emory University)[5], localizada no estado da Geórgia (EUA). Essa ferramenta de código aberto (open source) tem como proposta associar técnicas tradicionais de Record Linkage com um rico e configurável conjunto de parâmetros.

[5] http://www.emory.edu/home/index.html

3.2.2. BIGMATCH

O programa BigMatch (YANCEY, 2004, 2007) é um Record Linkage desenvolvido pelo departamento de censo dos Estados Unidos (U.S. Bureau of Census)[6], cujo objetivo é extrair combinações plausíveis de fontes de dados de grande volume. Ele permite que sejam configurados diferentes critérios de blocagem. Esse software também é utilizado para descobrir duplicações em um arquivo único.

3.2.3 PARALLEL FEBRL

Febrl[7] - Freely Extensible Biomedical Record Linkage (CHRISTEN, 2008) é uma das mais completas ferramentas de deduplicação disponíveis como software livre. Ferramenta de open-source para o data cleaning, que usa as seguintes métricas para a detecção de duplicados: edit distance e q-gram distance. Para a detecção de nomes aproximadamente duplicados é usada a codificação fonética (Soundex e Double Metaphone);

É importante notar que novas funcionalidades podem ser anexadas ao Febrl sem alterações em sua estrutura, criando uma plataforma ideal para novos experimentos em deduplicação de dados.

3.2.4. FERAPARDA/ PAREIA

FERAPARDA é uma ferramenta de relacionamento probabilístico capaz de detectar com sucesso réplicas em grande conjuntos de dados sintéticos com agilidade, por meio de algoritmo de deduplicação paralela, objetivando limitar o número de comparações. Inicialmente a ferramenta foi denominada por Feraparda e posteriormente recebeu o nome de Pareia. Santos et al (2007) descreve a ferramenta Feraparda, que é baseada no conceito de filtros-fluxo, implementada sobre a plataforma Anthill (FERREIRA et al, 2005).

3.2.5. SWOOSH

Resolução de entidade (ER)[8] é um problema que surge em muitas aplicações de integração de informações. ER (também conhecido como deduplicação ou merge-purge) usa duas funções, combinar e fundir. O Processo de ER identifica registros duplicados que se referem à mesma entidade do mundo real (processo de correspondência), e deriva informação consolidada sobre a entidade (processo de fusão). Além disso, o registro mesclado pode coincidir com outros registros de forma recursiva.

Foram propostos vários sistemas paralelos de deduplicação para melhorar o desempenho, desenvolvidos a partir do modelo ER. Os algoritmos da família Swoosh desenvolvido pelo grupo Infolab (TALBURT, 2008), se encontram nesse modelo. O algoritmo D-Swoosh (BENJELLOUN et al, 2006) para arquiteturas de processadores distribuídos e o algoritmo P-Swoosh (KAWAI et al, 2006) para arquiteturas paralelas.

3.2.6. PROGRAMAÇÃO GENÉTICA

A programação genética faz parte de uma área denominada “computação evolutiva”, que tem como base de inspiração a Teoria da Evolução de Charles Darwin. É uma área recente, surgida nos anos 50, fazendo parte das pesquisas de Inteligência

[6] http://www.census.gov/
[7] http://sourceforge.n«t/projects/febrl
[8] Entity Resolution

artificial (IA). Porém JOHN KOZA (1992), se tornou o responsável pela sua popularização, seu algorítmo de Programação Genética, foi aplicado a uma grande variedade de problemas incluindo controle, robótica, games, classificadores, etc.

Os indivíduos que serão evoluídos, na Programação Genética, são os programas de computador, representados por estruturas de árvores sintáticas. Estas árvores possuem funções e terminais, que determinam suas características e definem seu comportamento no ambiente. Cada função é um ramo da árvore, e cada terminal é uma folha. As funções podem ser, por exemplo, operações lógicas ou matemáticas, ou funções que provocam iteração, e os terminais podem ser variáveis, constantes, ou funções que não recebem argumentos

A linguagem LISP foi originalmente utilizada para implementar os algoritmos da programação genética, por ter características que facilitam a implementação de árvores. A linguagem utiliza o princípio da prefixação, o que faz com que as expressões simbólicas representem a árvore do programa. Atualmente as implementações em programação genética são desenvolvidas em linguagem C.

Carvalho et al. (2008) apresentaram uma abordagem inovadora para a identificação de registros duplicados em repositórios de dados, recorrendo a Programação Genética. Através dessa abordagem, registros são deduplicados utilizando-se evidências extraídas do conteúdo dos dados para criar funções de similaridade, genericamente denominadas de funções de deduplicação, capazes de apontar quais registros do repositório são réplicas.

Tabela 1: Quadro comparativo de ferramentas de deduplicação

SOLUÇÃO	ANO	DISPONIBILIDADE NA WEB		LINGUAGEM DE PROGRAMAÇÃO	SISTEMA OPERACIONAL	MODELO
		FONTES	MANUAL			
FRILL	2008	ABERTO	_	JAVA		GRAFO
BIGMATCH	2004, 2007	Disponível por solicitação	SIM - Web	C	Windows, UNIX e VAX	
FEBRL	2004	ABERTO		Python+ MPI		MASTER / SLAVE
FERAPARDA / PAREIA	2007	ABERTO		C++/PV M		PIPELINE
D-SWOOSH P-SWOOSH	2006 2006			JAVA		GRAFO / TAREFA MASTER / SLAVE
PROGRAMAÇÃO GENÉTICA	1992			LISP		ARVORES E GRAFOS

4. CONSIDERAÇOES FINAIS

O volume de dados que são operados nos sistemas computacionais hoje, seja na exploração, armazenamento ou integração de sistemas legados, oferece uma vasta oportunidade para extrair novos conhecimentos e, ao mesmo tempo, impulsiona a demanda por novas soluções.

Para atingir tais objetivos, as informações devem estar armazenadas em fontes de dados concisas e sem erros, ou seja, com uma boa qualidade. No entanto, muitas fontes de dados sofrem da baixa qualidade devido a anomalias ou impurezas ocasionadas principalmente pela inserção despadronizada destes dados. No contexto da deduplicação, os principais desafios giram em torno dos métodos para detectar defeitos que comprometem os critérios de avaliação da qualidade da informação. Dados imprecisos e inconsistentes são fontes de problemas e impulsionam os esforços de desenvolvimento das técnicas de deduplicação.

Dentre as ferramentas pesquisadas, pode-se inferir que algumas técnicas estão voltadas para características distintas que dão validação a qualidade dos dados, nota-se também a existência de técnicas voltadas para grandes volumes de dados, e técnicas voltadas para pequenos conjuntos de dados, explorando características distintas no contexto da deduplicação.

5. REFERÊNCIAS

Associação Brasileira de Normas Técnicas - ABNT. Norma ABNT NBR ISO/IEC 27002, 2005.

BATINI, C., CAPPIELLO, C., FRANCALANCI, C., & MAURINO, A. Methodologies for data quality assessment and improvement. ACM Computing Surveys (CSUR). 2009

BENJELLOUN, O. GARCIA-MOLINA, H. KAWAI, H. LARSON, T. MENESTRA, D. THAVISOMBOON, S. D-Swoosh: A Family of Algorithms for Generic, Distributed Entity Resolution. Stanford University Technical Report, 2006.

BHATTACHARYA I. AND L. GETOOR, “Iterative Record Linkage for Cleaning and Integration,” Proc. Ninth ACM SIGMOD Workshop Research Issues in Data Mining and Knowledge Discovery, pp. 11-18, 2004.

CARVALHO, M.G. DE, LAENDER, A.H.F, GONC¸ALVES, M.A. AND SILVA, A.S. DA, “Replica Identification Using Genetic Programming,” Proc. 23rd Ann. ACM Symp. Applied Computing (SAC), pp. 1801-1806, 2008.

CATARCI, T., AND SCANNAPIECO, M. Data quality under the computer science perspective. Archivi Computer 2. 2002.

CECCHIN, F. UM MODELO PARA RESOLUÇÃO DE CONFLITOS SOBRE REPOSITÓRIO DE DADOS XML. Dissertação de Mestrado do Programa de Pós-Graduação em Informática, UFPr, Curitiba. 2010.

CHAUDHURI, S. GANJAM, K. GANTI, V AND MOTWANI, R. “Robust and Efficient Fuzzy Match for Online Data Cleaning,” Proc. ACM SIGMOD Int’l Conf. Management of Data, pp. 313-324, 2003

CHIAVENATO, Idalberto. Gestão de Pessoas. 3ª edição, Editora Elsevier – Campus, 2008.

CHRISTEN P. Febrl: A freely available record linkage system with a graphical user interface. http://crpit.com/confpapers/CRPITV80Christen.pdf, 2008. [Acessado em 19-07-2014].

DBDIRETO. Higienização de Banco de Dados, 2011. Disponível em: < dbdireto.com.br/Higienizacao-de-dados.html>. Acesso em 16 out. 2014.

DORNALA, R. AKINGBEHIN, K. YOON, D. Data De-duplication in Storage Management. Int'l Conf. Internet Computing and Big Data - ICOMP'13. p. 10 – 14, 2013.

FERREIRA, R. MEIRA JR, W. GUEDES, D. DRUMMOND, L. COUTINHO, B. TEODORO, G. TAVARES, T. ARAUJO, R. FERREIRA, G. "Anthill: A scalable run-time environment for data mining applications". In Proc. of the 17th International Symposium on Computer Architecture and High Performance Computing, Rio de Janeiro, RJ, 2005.

FELLEGI, I.P. AND SUNTER, A.B. "A Theory for Record Linkage," J. Am. Statistical Assoc., vol. 66, no. 1, pp. 1183-1210, 1969.

GAMBLE M.; GOBLE C. "Quality, Trust, and Utility of Scientific Data on the Web: Towards a Joint Model". ACM International Conference on Web Science, pp. 1-8, 2011.

JURCZYK, P., LU, J., XIONG, L., CRAGAN, J., & CORREA, A. FRIL: A tool for comparative record linkage. In Proceedings of the American Medical Informatics Association Conference (AMIA 2008). 2008

KAWAI, H. GARCIA-MOLINA, H. BENJELLOUN, O. MENESTRINA, D. WHANG, E. AND GONG. H. "Pswoosh: Parallel algorithm for generic entity resolution". Technical Report, Stanford Info-Lab, 2006.

KOUDAS, N. SARAWAGI, S. AND SRIVASTAVA, D. "Record Linkage: Similarity Measures and Algorithms," Proc. ACM SIGMOD Int'l Conf. Management of Data, pp. 802-803, 2006

KOZA, J.R. Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, 1992.

LWIN, T & NYUNT, T. T. S. An Efficient Duplicate Detection System for XML Documents. 2nd International Conference on Computer Engineering and Applications. IEEE. 2010.

MADNICK, S.E., WANG, R.Y., LEE, Y.W. AND ZHU, H. Overview and Framework for Data and Information Quality Research. ACM Journal of Data and Information Quality 1, 1, Article #2. 2009.

MILLER, B., et al. Towards a framework for managing the information environment. Information and Knowledge Systems Management, v. 2. 2001.

OLSON, J. E. Data Quality – The accuracy dimension. San Francisco, CA: Morgan Kaufmann Publishers, 2003. ISBN: 1-55860-891-5

PIPINO, L. L.; LEE, Y. W.; WANG, R. Y. Data quality assessment. Communications of the ACM, New York, v. 45, n. 4, p. 68-73, Apr. 2002.

RAHM, E.; AND DO, H. H. "Data Cleaning: Problems and Current Approaches," Bulletin of the Technical Committe on Data Engineering – Special Issue on Data Cleaning, vol. 23, no. 4, pp. 3-13, 2000.

SANTOS. W, TEIXEIRA, T. MACHADO, C. MEIRA, W. SILVA, A. FERREIRA, R. GUEDES, D. "A Scalable Parallel Deduplication Algorithm". In 19th International Symposium on Computer Architecture and High Performance Computing 2007.

TALBURT, John R. Entity resolution and information quality. Elsevier, 2011.

TAVARES, Rossano Soares. Bancos de Dados Qualificados Podem Reduzir Perdas e Aumentar os Ganhos em CRM, 82 pg. (Monografia (MBA) – Pontifícia Universidade Católica de São Paulo, São Paulo. 2003.

VERYKIOS, V.S. MOUSTAKIDES, G.V. AND ELFEKY, M.G. "A Bayesian Decision Model for Cost Optimal Record Matching," The Very Large Databases J., vol. 12, no. 1, pp. 28-40, 2003.

WAND, Y.; WANG, R. Anchoring data quality dimensions in ontological foundations. Communications of the ACM, v. 39, n. 11. 1996.

YANCEY, W.E. An adaptive string comparator for record linkage RR 2004-02, US Bureau of the Census, February. 2004.

YANCEY, W.E. BigMatch: A Program for Extracting Probable Matches from a Large File. www.census.gov/srd/papers/pdf/rrc2007-01.pdf, 2007. [Acessado em 19-07-2014].

Conceptual Framework to Support Sampling Activities in Software Engineering Surveys

Rafael M. de Mello, Guilherme H. Travassos

Programa de Engenharia de Sistemas e Computação (PESC), COPPE/UFRJ
Universidade Federal do Rio de Janeiro

Caixa Postal 68.511 – 21.9451-970 – Rio de Janeiro – RJ – Brasil

{rmaiani, ght}@cos.ufrj.br

Abstract. *Although questionnaire-based surveys have been frequently applied in Software Engineering research, specialized literature shows many issues on establishing representative samples. Consequently, it is hard to generalize or even to interpret the survey results. The lack of sources for establishing adequate sampling frames and the lack of systematic activities to support sampling in Software Engineering surveys contribute to this adverse scenario. This paper presents an ongoing research aiming at establishing a conceptual framework to support researchers on conducting sampling activities in Software Engineering surveys. Based on the lessons learned through the conduction of preliminary studies, a first version of the conceptual framework was designed. Then the framework was evolved, with the inclusion of a set of activities and evidence based recommendations for supporting its use.*

1. Introduction

Surveys (questionnaire-based surveys) are one of the most frequently research methods used in Software Engineering (SE) research, allowing researches to perform descriptive large-scale investigations without the rigorous control level requested by controlled (*quasi*) experiments. Surveys are primary studies that shall support repetition through several executions and allow the aggregation of observed results. In this context, SE literature often presents surveys in which their research plans are clearly designed in many aspects but having sampling frames typically established by convenience [de Mello and Travassos, 2015]. Consequently, even after exhaustive effort on recruitment, the interpretation of the survey results is significantly limited since the sampling process could not be repeated in further replications.

In comparison with another research areas (i.e. social sciences, nursing, medicine,…), identifying representative sources of population available in SE research is a great challenge, motivating researchers to work with alternative sources, such as logs from open-source projects [Bettenburg et al., 2008], discussion groups [Nugroho and Chaudon, 2008] and social networks [França e da Silva, 2009; Martínez-Fernández et al., 2010]. However, the *ad hoc* use of such technologies *per se* for increasing sample sizes is not sufficient to evolve the samples quality, since sample size is just one part of samples' representativeness challenge [Kruskal and Mosteller, 1979]. In the context of our research, a *representative sample* consists on subset of units of analysis, *randomly* retrieved from a *heterogeneous* population from the point view of the survey target audience. Such heterogeneity can be measured through the attributes used in a study for characterizing each unit [de Mello et al., 2015-2]. For instance, in a survey having

Brazilian SE research groups as target audience, the unit of analysis is the research group and a representative population could be retrieved through the research group directory from CNPq (http://lattes.cnpq.br/web/dgp). In addition, it is also important to emphasize that establishing representative samples may be not sufficient if their individuals (unit of observation) do not effectively participate. In this context, Smith et al. (2013) argue that individual participation may be stimulated through persuasive factors applied in the survey recruitment. Thus, the following research questions emerge:

- *How to identify and assess potentially relevant sources of population for conducting surveys in SE?*
- *How to deal with the limitations on retrieving relevant information from the source of population available for a survey?*
- *How to stimulate the participation of individuals out from the convenience samples in SE surveys?*
- *How to systematize the whole sampling process in order to make it reusable?*

In order to answer the presented questions, it has been observed that surveys guidelines available in SE literature do not provide enough guidance [Pfleeger and Kitchenham, 2001; Kasunic, 2005; Pfleeger and Kitchenhan, 2008]. Thus, this paper presents the ongoing research in the context of a Doctoral Thesis aiming at establishing a conceptual framework for supporting researchers on establishing representative samples in SE surveys. Such framework includes a set of activities and evidence-based recommendations in order to guide the use of its concepts.

Section 2 discusses the related works. Section 3 presents the research proposal, delimitating its scope and describing the expected contributions to SE research. Section 4 presents the research progress and related publications, including each step performed until the submission of this paper. Section 5 presents the conclusions.

2. Related Works

Kasunic [2005] presents a hands-on set of guidelines for conducting SE surveys, describing the survey process through seven steps. Four of these steps are composed by planning activities (Figure 1), being the second and the third steps directly related with the research presented in this paper. In this context, it was observed that challenges and issues regarding the establishment of the target audience and the sampling design in SE surveys are barely discussed. Also, no discussion is presented regarding how to stimulate the participation in SE surveys.

In a series of five short papers discussing the design of SE surveys, Pfleeger and Kitchenham [Pfleeger and Kitchenham, 2001] devotes one paper for presenting population and sampling concepts in general. Again specifically issues and recommendations for SE issues were not observed, which is also observed in a more recent work [Pfleeger and Kitchenham, 2008].

In addition, specialized literature presents a couple of papers can be considered as guidelines due to the initiative of their authors on detailed reporting their own experiences on conducting large-scale surveys in SE [Ciolkowski et al., 2003; Conradi et al., 2005]. Ciolkowski et al. [2003] present a comprehensive work reporting the

authors' experience on conducting three SE surveys. However, although they present the composition of each survey sample, sampling issues in SE are barely discussed. Conradi et al. [2005] reported in depth how they established the target audience for an international survey and how they obtained a representative sample through an exhaustive process of gathering organizations' data from three countries. The authors also describe a relevant set of attributes collected for characterizing each respondent and how they applied such attributes for better interpreting the survey results. This survey was replicated by Li et al. [2008] having organizations from a fourth country as sampling frame. In both studies, the authors discuss the challenges and the limitations on establishing representative populations for SE surveys. However, no proposal or even guidelines for overcome such challenges was identified.

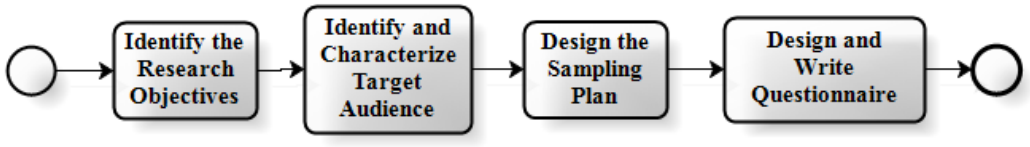


Figure 1. Activities for survey planning [Kasunic, 2005].

Regarding the quality of surveys reports, Stavru [2014] introduced a set of criteria of *thoroughness* to evaluate the quality of industrial surveys on agile method usage, which includes the need of survey papers specifying the *target audience (target population)*, *sampling frame* and *sample size*, among others. In Savru approach, one or more of the following *trustworthiness* attributes should be applied for evaluating each criterion: *neutrality, consistency, truth value* and *applicability*. The author observed that eight from the nine studies analyzed present insufficient *thoroughness* and subsequently low *trustworthiness*. Then, the author presents as set of recommendations to improve this scenario, including that *"special provisions should be taken to increase the objectivity of surveys on agile method usage in order to ensure that their findings are not biased by the individuals or organizations conducting them"*.

3. Research Proposal

The research proposal consists on *establishing a framework composed by a set of concepts and activities for supporting researchers on designing representative samples for surveys in Software Engineering.* The research plan has been adapted from the evidence-based approach for introducing new SE technologies used by the Experimental Software Engineering group (ESE Group) at COPPE/UFRJ [Dias-Neto et al., 2010]. Figure 1 presents the proposed research methodology, which includes the following main activities:

1. *Conceptual characterization of the technology,* presenting relevant concepts for providing systematized support for sampling in SE surveys;

2. *Conduction of Preliminary studies,* applying the result of the previous activity on replicating surveys from diverse SE researches conducted by ESE group;

3. *Framework development: concepts,* applying the lessons learned in the preliminary studies for reviewing the proposed concepts and organize then into a conceptual framework;

4. *Framework development: activities,* aggregating activities and evidence-based recommendations to the conceptual framework;

5. *Feasibility study*- planned to have, at least, SE researchers from a post-graduation class applying and evaluating the proposed framework.

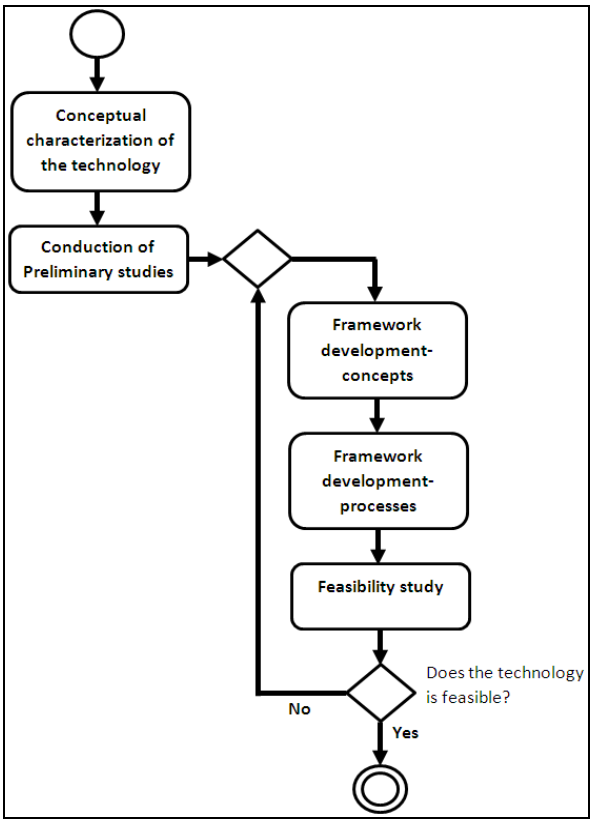


Figure 1. The research methodology proposed.

As the main contribution of this research, it is expected the evolution of quality on planning and replicating SE surveys. In addition, it is expected to contribute with the following SE research topics:

- Characterization of context in SE research [Petersen e Wholin et al., 2009];
- Identification and establishment of alternative sources of population in SE surveys [de Mello et al., 2014].
- Persuasive factors and Participation in SE surveys [Smith et al., 2013];
- Improvement of sampling in large-scale experiments [de Mello et al., 2015].

4. Research Progress and Publications

A conceptual characterization of the technology was designed based on surveys' knowledge available in the technical literature, the lessons learned by the ESE Group on

conducting questionnaire-based surveys and the specific challenges observed in SE research on characterizing populations and aggregating results from aggregated experiments [de Mello et al., 2013]. These concepts were applied to perform a first preliminary study [de Mello et al., 2013-2], in which was designed a *recruitment plan* for replicating a survey on requirements effort estimation. Through a comparison between convenience sampling and the planned sample, it was observed evidence that applying a recruitment plan over a professional social network (LinkedIn) contributed for delivering more heterogeneous samples without less in confidence level (experience) of the subjects. A second preliminary study on simulation-based studies in Software Engineering was also replicated [França and Travassos, 2014]. However, due to the specificity of the study, an insufficient effective sample size was retrieved for testing any hypothesis regarding the samples' quality.

Then, a new recruitment plan was designed to support a third preliminary study, regarding a replication of a survey on Agility in Software Processes [Abrantes and Travassos, 2012]. The use of the recruitment plan allowed the recruitment of a representative sample composed by 7,745 distinct individuals, members from a set of 19 groups of interest (grouped into eight strata) systematically selected from the professional social network LinkedIn (www.linkedin.com) [de Mello et al., 2014-2]. After analyzing the answers from the respondents regarding their main SE skills the original eight strata were reorganized into five groups [de Mello et al., 2014-3]. As evidenced in the first preliminary study, it was also observed in this third study that the recruitment process delivered a more heterogeneous sample without losing in confidence, when compared with the previous two survey executions [Abrantes and Travassos, 2013]. Such heterogeneity was essential to support the identification of relevant opinion divergences between groups regarding the research context, to reinforce some results observed in the previous executions and to put another ones in doubt [de Mello et al., 2014-4].

Thus, based mainly on the experience obtained conducting the aforementioned studies, the first version of the conceptual framework was designed [de Mello et al., 2014], including new concepts for dealing with challenges observed on SE surveys. Then, part of the conceptual framework was applied to evaluate nine possible alternative sources available in the Web for establishing representative populations in SE surveys, including *professional social networks*, *crowdsourcing tools* and *freelancing tools* [de Mello et al., 2014]. The use of the conceptual framework was also exemplified in the context of a real survey [de Mello et al., 2015-2].

After the establishment of the first version of the conceptual framework, a structured review over the proceedings of the two most relevant Empirical Software Engineering Conferences (ESEM and EASE) was undertaken aiming at investigating the state of practice on characterizing sampling frames in SE surveys [de Mello and Travassos, 2015]. As a result, 56 surveys were identified in which was observed that only seven reported efforts on designing representative samples. From these, four surveys were designed having SE researchers as their target audience, accessed through the list of authors of papers retrieved from a SLR (systematic literature review) previously conducted for each research context [Dias Neto and Travassos, 2008; Santos and da Silva, 2013; Carver et al., 2013; Gúzman et al., 2014]. The survey conducted by Rodríguez et al. [2012], demonstrates the benefits on accessing a national database composed by Finnish software professionals and organizations (FIPA). As a result,

4,450 SE practitioners from Finland could be recruited and 408 answers were obtained. Other two surveys [de Mello and Travassos, 2013-2; de Mello et al., 2014-3] consists on already presented survey replications conducted in the context of this research.

After the conduction of the mentioned structured review, it was observed the need of evolving the framework concepts and then activities and recommendations to support the framework use were designed. The results of the review allowed us to include a set of recommendations for applying the framework activities. Following subsections presents, respectively, the new version of the framework concepts and exemplify one of the framework activities.

4.1 Framework Development- Concepts

Following subsection briefly presents the new version of the framework concepts. Figure 2 presents such concepts associated to deliverables from survey planning and execution.

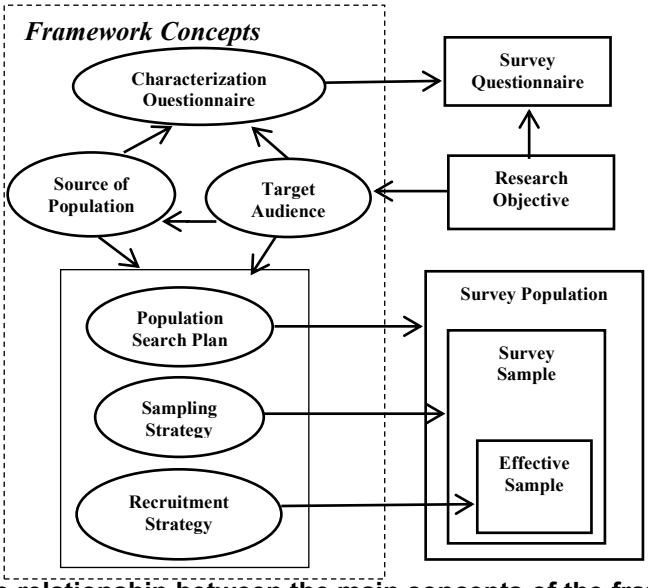


Figure 2. The relationship between the main concepts of the framework.

4.3.1 Target Audience

A survey *target audience* characterizes who can best provide the information needed in order to achieve the *research objective* [Kasunic, 2005]. In our framework, this concept is extended to a formal characterization of the survey *unit of observation* and *unit of analysis*. In questionnaire based surveys, data is always collected from *units of observation* represented by the individual (respondent). However, the survey's target audience may demand a higher level of analysis (*unit of analysis*). In a recent structured review conducted to characterizing sampling frames in SE surveys, it was observed that 15 from the 56 surveys since 2005 established specific groups of individuals as their unit of analysis, including *organizational units, organizations and project teams* [de Mello and Travassos, 2015].

4.3.2 Source of Population

A survey *population* consists on the set of accessible units of analysis from the target audience [Thompson, 2010]. Thus, a *source of population* consists on a database

(automated or not) in which a valid population for a specific target audience can be systematically retrieved. As a consequence, if a source of population can be considered valid for supporting a specific research context, it can be concluded that adequate sampling frames can be established from it for the same research context.

In the context of a source of population, its *search unit* characterizes the entity from which one or more units of analysis can be retrieved from a specific source of population. In an ideal scenario, it is expected that both unit of analysis and search unit are the same thing. However, SE literature presents some examples in which these units are different. For instance, Conradi et al. [2005] aimed at investigating the opinion of software project teams (unit of analysis), but accessed them keeping in touch with organizations (search unit) from three distinct countries. Dias Neto and Travassos [2008] opted to survey the authors (unit of analysis) of each paper (search unit) retrieved from the results of specifics SLRs conducted for each research context.

Figure 3 exemplified the concepts of source of population (SoP) and search unit (SU) with the concepts of target audience (TA), population (POP) and unit of analysis (UA). One can see that not necessarily all instance of search unit from a source of population can be used to compose a specific population. To be considered valid, a source of population should satisfy, at least, the following essential requirements (ER):

- *ER1. A source of population should not intentionally represent a segregated subset from the target audience, i.e., for a target population audience "X", it is not adequate to search for units from a source intentionally designed to compose a specific subset of "X".*
- *ER2. A source of population should not present any bias on including on its database preferentially only subsets from the target audience. Unequal criteria for including search units mean unequal sampling opportunities.*
- *ER3. All source of population' search units, their units of analysis (and their units of observation) must be identified by a logical or numerical id.*
- *ER4. All source of population' search units must be accessible. If there are hidden search units, it is not possible to contextualize the population.*

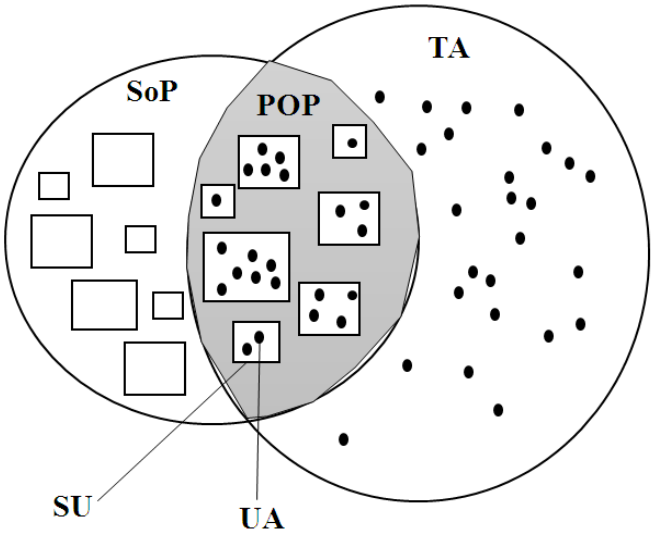


Figure 3. Population obtained from a source of population and its units.

There are still also nine desirable requirements (DR), three concerned with the samples' accuracy (ADR), two concerned with clearness (CDR) and four regarding sample's completeness (CoDR). The first version of such requirements can be observed in [de Mello et al., 2014].

4.3.3 Characterization Questionnaire

Attributes needed for characterizing each individual are frequently unavailable in the sources of population. Thus, such attributes data are commonly retrieved based on subjects' answers to one or more survey questions. For instance, Dias Neto and Travassos [2008] collected from each subject the set of attributes needed to support their research since the source of population/search unit used (digital libraries/ papers) do not retrieve another information regarding the subjects than their names and e-mails. The characterization questionnaire is typically included at the beginning or at the end of the survey questionnaire and should avoid ask any information already available (and updated) in the source.

4.3.4 Search Plan

A *search plan* describes how search units will be systematically retrieved from a source of population and evaluated in order to deliver the study population, being composed by the following elements:

- *Search string*- a set of *search expressions* connected through logical operators that can be applied to a source of population in order to retrieve adequate search units.
- *Search algorithm*- describes each step, automated or not, that must be followed in order to filter the search units in a source of population, including how to apply the planned search string.
- *Exclusion criteria*- describes a set of restrictions that must be applied in order to exclude undesirable search units retrieved from the search plan execution. Exclusion criteria can be especially helpful when the source of population is significantly generic and the use of search string are limited, such as in the case of the professional social networks [de Mello et al., 2013; de Mello et al., 2014-2] and yellow pages [Conradi et al., 2005].

4.3.5 Sampling Strategy

A *sampling strategy* establishes criteria for composing the survey *sampling frame* and describes the survey *sampling design*. While the *sampling frame* is the source from which a sample can be retrieved [Thompson, 2012], the sampling design describes the criteria for extracting samples from the sampling frame, i.e. which individuals (from which unit of analysis) will be invited to answer the survey.

4.3.6 Recruitment Strategy

The recruitment strategy characterizes how the individuals from the survey sample will be recruited. It includes the *invitation message* and the following factors that can influence subjects' participation [Smith, 2013]: *execution esteemed time, invitation method, period available, reminding method* and *Reward method* rewards may be, but is not limited to include *payments*, *raffles*, *gifts* and *donations for NGOs*.

4.4 Framework Development- Activities

Six activities and 17 tasks were developed to support SE researchers on applying the conceptual framework presented in Section 3. Each task is supported by one or more recommendations, totalizing 27. Such recommendations was developed based on evidence observed in the specialized literature on conducting SE surveys, specially identified in the structured review [de Mello and Travassos, 2015]. As mentioned in the Section 3, the conceptual framework was developed to support only population and sampling issues in SE surveys. Thus, Figure 4 presents through a BPMN model how the framework activities are inserted in the survey planning steps presented in the Section 2. Shadowed activities are out from the framework scope. One can see that the framework introduces new steps between the *characterization of the target audience* and the *design the sampling design* presented by Kasunic [2005]. The proposed framework also devotes specific activities for designing the characterization questionnaire and designing the recruitment strategy.

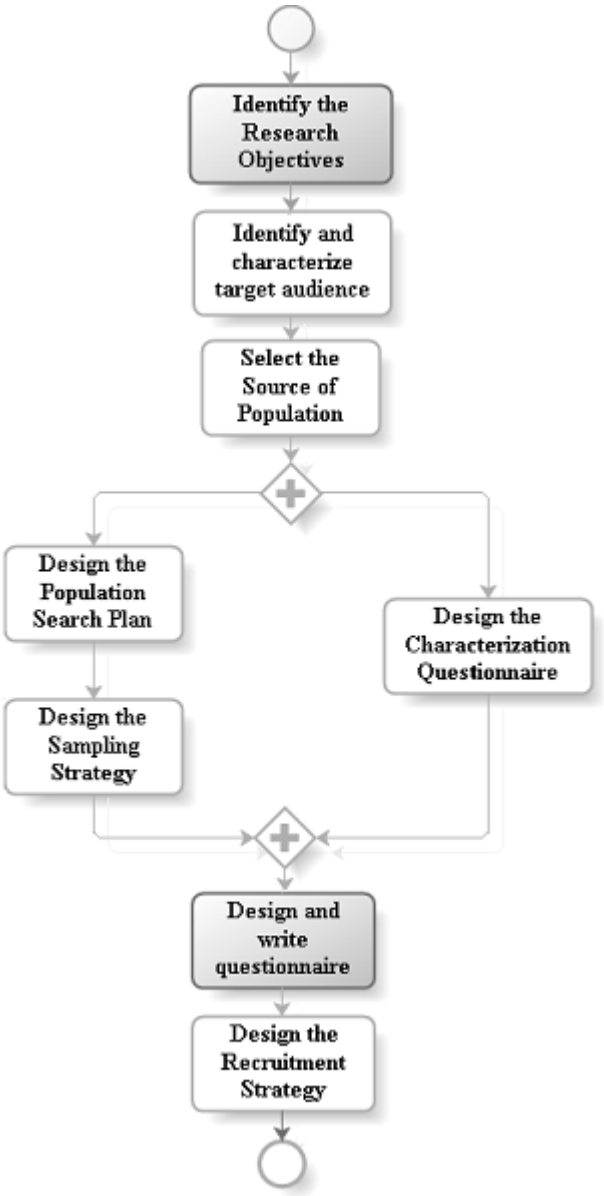


Figure 4. The framework activities inserted in the survey planning process.

Since the *research objectives* were identified, a *target audience* must be established, and an accessible population should be found, which will be supported by applying a *population search plan* over the selected *source of population*. Then, a compatible *sampling strategy* should be applied in order to deliver the sample of the survey trial. Finally, the *recruitment strategy* should be designed focusing on stimulating the participation in the survey. Due to the limitation of space, the description of all 17 tasks and the 27 recommendations are not displayed in this paper. Figure 5 exemplifies the third task of the activity "Identify and Characterize the Target Audience" (TA03) and its respective recommendations (R02, R03 and R04).

TA03. Establish the unit of analysis attributes. First, establish the set of control attributes and their respective values that will restrict the unit of analysis. Then, enumerate the other attributes that should be collected from each unit and define how to measure each one.

R02. Individuals are commonly characterized in SE through the following attributes: experience in the research context, experience in SE, current professional role, country and higher academic degree.

R03. Organizations are commonly characterized in SE through the following attributes: size (scale typically based in the number of employees), industry segment (software factory, avionics, finance, health, telecommunications, etc.), country and organization type (government, private company, university, etc.).

R04. Project teams can be characterized through attributes such as project size; team size, client/product professional segment (avionics, finance, health, telecommunications, etc.) and their physical distribution.

Figure 5. The framework activities inserted in the survey planning process.

5. Conclusion

This paper presented an on-going Doctoral research to develop a conceptual framework to support sampling in SE surveys. Most of the planned research steps were performed. As immediate next step, the second version of the conceptual framework will be submitted to a feasibility study having Doctoral and Master students from an Experimental Software Engineering class as subjects. Within the results of such study, the framework will be improved to be presented in the context of the Doctoral Thesis. Future steps include the conduction of additional studies for evaluating the conceptual framework and expanding the use of the framework concepts to support large scale experiments in SE. It is also expected that new evidence observed in future SE surveys could be used to improve and add new recommendations to the framework.

Acknowledgement

We would like to thank to Dr. Per Runeson (Lund University, Sweden) to the external supervision during the sandwich doctorate program and to Pedro Correa (COPPE/UFRJ) to the support in the research activities. We also thanks to Dr. Martin Höst (Lund University), Dr. Carolyn Seaman (UMBC, EUA), Dr. Alessandro Garcia (PUC-RJ, Brazil) and Dr. Márcio Barros (UNIRIO, Brazil) by the relevant contributions for the research. Finally, we thanks to CAPES to the support on the external scholarship program at Lund University.

Referências Bibliográficas

A. C. Dias Neto, G. H. Travassos. Surveying Model Based Testing Approaches Characterization Attributes. In: Proc. 2nd ACM/IEEE ESEM, pp. 324–326, 2008.

A. C. Dias-Neto, R. O. Spínola and G. H. Travassos. "Developing software technologies through experimentation: experiences from the battlefield," Proceedings of Conferencia Ibero-Americana en Software Engineering, pp. 107-121, 2010.

A. Nugroho and M. R. V. Chaudon. A survey into the rigor of UML use and its perceived impact on quality and productivity. Proceedings of the Second ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), pp. 90-99. ACM, 2008.

C. C. Franca, F. Q. B. da Silva. An Empirical Study on Software Engineers Motivational Factors. Proc. Third ACM/IEEE ESEM, pp. 405-409. IEEE, 2009.

E. Smith et al. Improving developer participation rates in surveys. 6th Intl. Workshop on Cooperative and Human Aspects of Software Engineering (CHASE), IEEE, 2013.

B. B. N. França and G. H. Travassos. Simulation Based Studies in Software Engineering: A Matter of Validity. CLEI Elet. Journal vol. 18.1, Article No. 4, 2014.

J. C. Carver, E. Hassler, E. Hernandes, N. A. Kraft. Identifying Barriers to the Systematic Literature Review Process. In: Proc. 7th ACM/IEEE ESEM, pp. 203–212, 2013.

J. F. Abrantes and G. H. Travassos. Towards Pertinent Characteristics of Agility and Agile Practices for Software Processes. CLEI Electronic Journal 16.1, No. 5, 2013.

J. L. Martínez-Fernández et al. Using Surveys to Evaluate a Business Rules Based Development Approach. Business Information Systems, pp. 132-143. Springer Berlin Heidelberg, 2010.

K. Petersen and C. Wohlin. Context in industrial software engineering research. Proceedings of the Third ACM/IEEE ESEM. IEEE, 2009.

L. Guzmán, C. Lampasona, C. Seaman, D. Rombach. Survey on Research Synthesis in Software Engineering. In: Proc. 18th EASE, pp. 2:1–2:10, 2014.

M. Ciolkowski, O. Laitenberger, S. Vegas, S. Biffl. Practical experiences in the design and conduct of surveys in empirical software engineering. In: Conradi R, Wang AI (eds). Empirical Methods and Studies in Software Engineering- Experiences from ESERNET, pp. 104-128. Springer Berlin Heidelberg, 2003.

M. Kasunic. Designing an Effective Survey. TR CMU/SEI-2005-HB-004, Carnegie Mellon University, 2005.

N. Bettenburg et al. What Makes a Good Bug Report? Proceedings of the 16th ACM SIGSOFT Intl. Symp. on Foundations of Soft. Eng., pp. 308-318. ACM, 2008.

P. Rodríguez et al. "Survey on agile and lean usage in finnish software industry," Proceedings of Proceedings of 6th ACM/IEEE ESEM. ACM, 2012.

R. Conradi et al. Reflections on conducting an international survey of Software Engineering. Proceedings of ESEM, pp. 10, 2005.

R. E. S. Santos, F. Q. B. Da Silva. Motivation to Perform Systematic Reviews and their Im-pact on Software Engineering Practice. In: Proc. 7th ACM/IEEE ESEM, pp. 292–295, 2013.

R. M. de Mello and G. H. Travassos. An ecological perspective towards the evolution of quantitative studies in software engineering. Proceedings of the 17th International Conf. on Evaluation and Assessment in Software Engineering (EASE). ACM, 2013.

R. M. de Mello and G. H. Travassos. Would Sociable Software Engineers Observe Better? In Proceedings of 7th ESEM, IEEE, 2013-2.

R. M. de Mello, P. C. da Silva, P. Runeson, G. H. Travassos Towards a framework to support large scale sampling in software engineering surveys. Proceedings of the 8th ACM/IEEE ESEM, pp. 48-52. ACM, 2014.

R. M. de Mello, P. C. da Silva, G. H. Travassos. Investigating Probabilistic Sampling Approaches for Large-Scale Surveys in Software Engineering. Proceedings of 11th Workshop on Experimental Software Engineering (ESELAW), Pucón, Chile, 2014-2

R. M. Mello, P. C. da Silva, G. H. Travassos Sampling improvement in software engineering surveys. Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, pp. 13-17. ACM, 2014-3.

R. M. de Mello, P. C. Silva, G. H. Travassos. Agilidade em Processos de Software: Evidências Sobre Características de Agilidade e Práticas Ágeis. In: XIII Brazilian Symposium on Sofware Quality, Blumenau, Brazil, 2014-4. (in Portuguese)

R. M. de Mello and G. H. Travassos, GH. Characterizing Sampling Frames in Software Engineering Surveys. In: Proceedings of 18th ESELAW, Lima, Peru, 2015.

R. M. Mello, K. T. Stolee, G. H. Travassos. Investigating Samples Representativeness for Online Experiments in Java Code Search. ESEM 2015. (submitted)

R. M. de Mello, P. C. da Silva, G. H. Travassos. Investigating Probabilistic Sampling Approaches for Large-Scale Surveys in Software Engineering. Journal of Software Engineering Research and Development (JSERD), 2015-2. (accepted)

S. K. Thompson SK. Sampling. John Wiley & Sons, 3 ed. 2012.

S. L. Pfleeger and B. A. Kitchenham. Principles of survey research: part 1: turning lemons into lemonade. ACM SIGSOFT Soft. Eng. Notes 26.6, pp. 16-18, 2001.

S. L. Pfleeger and B. A. Kitchenham. Personal Opinion Surveys. Guide to Advanced Empirical Software Engineering, pp 63-92. Springer London, 2008.

S. Stavru. A critical examination of recent industrial surveys on agile method usage. Journal of Systems and Software 94, pp: 87-97. 2014.

W. Kruskal and F. Mosteller. Representative Sampling III: The Current Statistical Literature. International Statistical Review / Revue Internationale de Statistique, Vol. 47, No. 3, pp. 245-265, 1979.

Processo de Conformidade Arquitetural em Integração Contínua

Arthur F. Pinto, Ricardo Terra

Departamento de Ciência da Computação,
Universidade Federal de Lavras (UFLA), Brasil

arthurfp@sistemas.ufla.br, terra@dcc.ufla.br

Abstract. *As software evolves, developers usually introduce deviations from the planned architecture, due to unawareness, conflicting requirements, technical difficulties, deadlines, etc. Although architectural compliance processes identify architectural violations, (i) these tools are underused and (ii) detected violations are rarely corrected. To address these shortcomings, this paper proposes a solution of architectural compliance into continuous integration. Thus, the architectural compliance process is triggered by every code integration, and when no violations are detected, the code is integrated into the repository. In addition, this paper presents the ArchCI tool—that implements the proposed solution using DCL as underlying conformance technique and Jenkins as the CI server—and a controlled evaluation that demonstrates the applicability of the solution.*

Resumo. *No decorrer de um projeto de software, desenvolvedores normalmente introduzem desvios em relação à arquitetura planejada, seja por desconhecimento, requisitos conflitantes, dificuldades técnicas, prazos curtos, etc. Embora processos de conformidade arquitetural identifiquem violações arquiteturais, (i) essas ferramentas são subutilizadas e (ii) violações detectadas são raramente corrigidas. Diante disso, este artigo propõe uma solução de conformidade arquitetural em integração contínua. Isso implica que o processo de conformidade arquitetural é ativado a cada integração de código e, quando violações não forem detectadas, o código poderá ser integrado ao repositório. Além disso, este artigo apresenta a ferramenta ArchCI – que implementa a solução proposta usando DCL como técnica de conformidade e Jenkins como servidor de CI – e uma avaliação controlada que demonstra a aplicabilidade da solução.*

1. Introdução

No decorrer de um projeto de software, desenvolvedores normalmente introduzem desvios em relação à arquitetura planejada, seja por desconhecimento, requisitos conflitantes, dificuldades técnicas, prazos curtos, etc. [14, 13, 19]. Isso se agrava em projetos com vários desenvolvedores uma vez que o acúmulo dos possíveis desvios arquiteturais que podem ocorrer durante sua implementação, são potencializados pelo aumento do número de desenvolvedores em um projeto, levando ao fenômeno conhecido como erosão arquitetural [11, 6]. Mais importante, esses desvios arquiteturais impactam negativamente o projeto, podendo anular características essenciais de um sistema, como manutenibilidade, reusabilidade, escalabilidade, portabilidade, etc. [13, 20].

Embora processos de conformidade arquitetural identifiquem violações arquiteturais, (i) essas ferramentas são subutilizadas e (ii) violações detectadas são raramente corrigidas. Diante disso, este artigo propõe uma solução de conformidade arquitetural em integração contínua. Isso implica que o processo de conformidade arquitetural é ativado a cada integração de código e, quando violações não forem detectadas, o código poderá ser integrado ao repositório, o que soluciona os problemas *(i)* e *(ii)*. Além disso, este artigo apresenta a ferramenta ArchCI – que implementa a solução proposta usando DCL (*Dependency Constraint Language*) como técnica de conformidade [19] e Jenkins como servidor de CI [16] – e uma avaliação controlada que demonstra a aplicabilidade da solução.

O restante deste artigo está organizado como a seguir. A Seção 2 introduz conceitos fundamentais ao estudo. A Seção 3 descreve a solução proposta que evita os problemas decorrentes de um processo de erosão arquitetural. A Seção 4 detalha a implementação da ferramenta ArchCI. A Seção 5 avalia a aplicabilidade da solução proposta. Por fim, a Seção 6 apresenta as considerações finais e trabalhos futuros.

2. Background

2.1. Controle de Versão

Um sistema de controle de versão (*Version Control System*, VCS) é um software com a finalidade de gerenciar diferentes versões no desenvolvimento de artefatos de um projeto [17, 18]. Como principal contribuição, oferece rastreabilidade das alterações, como o responsável pelas mudanças, hora e data, diferenças das versões, etc.

Os sistemas podem ser centralizados ou distribuídos [9]. VCSs centralizados apresentam repositórios de códigos, onde o acesso e a escrita de dados estão restritos a um grupo de desenvolvedores [2]. VCSs distribuídos, por outro lado, trabalham com a arquitetura *peer-to-peer*, de forma que cada cópia de um projeto contém todo o histórico e os metadados do projeto, garantindo aos desenvolvedores a capacidade de compartilhar as mudanças da forma que mais se adeque às suas necessidades [12]. Dentre as principais ferramentas de controle de versão – CVS, SVN, Git e Mercurial – escolheu-se, para o desenvolvimento deste projeto, o Git[1] por oferecer a possibilidade de se desenvolver de maneira centralizada e distribuída, além de ser um dos mais utilizados atualmente [12].

Neste artigo, é importante a contextualização com os seguintes conceitos [17, 18]: (i) *tag*, nome simbólico atribuído à uma versão específica; (ii) *branch*, um conjunto de versões de arquivos fontes que é identificado por uma *tag*; (iii) *commit*, comando que integra as alterações de um desenvolvedor a um *branch* do repositório local; e (iv) *push*, comando que integra uma série de *commits* de um desenvolvedor a um *branch* do repositório remoto.

2.2. Integração Contínua

Integração Contínua (*Continuous Integration*, CI) trata-se da prática de desenvolvimento de software, onde membros de uma equipe incorporam certas mudanças ao software, aplicando processos de compilação e testes que asseguram a integridade do projeto [8]. Essa prática facilita na detecção de erros e problemas nas fases anteriores à conclusão do software, visando um menor custo de reparo [3]. A solução proposta neste artigo objetiva

[1] http://git-scm.com/

complementar esse processo de integração, provendo meios de verificar a arquitetura do sistema de software.

Servidores de CI podem ser configurados para verificarem sempre que mudanças são realizadas em um repositório [7]. Assim, recupera as versões mais recentes das classes, compila o código, e, em seguida, executa os testes para integração, exibindo os resultados aos desenvolvedores [4]. Dentre os servidores de CI mais relevantes – Jenkins, TeamCity e CruiseControl – o Jenkins[2] conseguiu um alcance maior na comunidade *open-source*, tendo assim, certa vantagem para a identificação e correção de *bugs*, bem como certas melhorias, se tornando o servidor mais recomendado para este projeto [16].

2.3. Conformidade Arquitetural

Conforme um projeto de software é desenvolvido, sua arquitetura está sempre evoluindo à medida que seu sistema também evolui. Portanto, são necessários meios de rastrear essas evoluções e outros aspectos implícitos do sistema de software. Esse processo é chamado de *architectural monitoring* [11]. Torna-se, assim, imprescindível para um sistema de software garantir a conformidade entre a arquitetura planejada e sua implementação atual. Contudo, é comum o acúmulo de violações arquiteturais ao longo do tempo, levando ao fenômeno conhecido como erosão arquitetural [14].

Define-se como erosão arquitetural o fenômeno que ocorre quando a arquitetura implementada de um sistema de software diverge de sua arquitetura planejada [6]. Existem diversas técnicas para evitar a erosão arquitetural, bem como para se realizar o processo de *architectural monitoring*. Dentre as principais técnicas, pode-se citar: Modelos de Reflexão [10], Matrizes de Dependências Estruturais [15], Source Code Query Languages [21], ArchJava [1], Testes de Desenho [5], e Linguagens de Restrição Arquiteturais [19]. Dessas técnicas, a que será utilizada neste projeto será a linguagem DCL, devido ao seu fácil uso e ao fato da mesma apresentar uma alta expressividade na forma de se tratar o problema de erosão arquitetural.

DCL é uma linguagem declarativa de domínio específico, que apoia a definição de restrições estruturais entre módulos em sistemas orientados a objetos, tendo como objetivo principal, restringir a organização modular de um sistema de software, em vez de seu comportamento [19]. Através da definição de restrições estruturais por meio do DCL, torna-se possível capturar dois tipos de violações arquiteturais: *divergências* (quando uma dependência observada no código fonte não está de acordo com o modelo arquitetural do sistema) e *ausências* (dependência inexistente no código fonte, mas que é obrigatória de acordo com o modelo arquitetural). Essencialmente, esse modelo abrange qualquer forma de relação entre classes que podem ser verificadas estaticamente. Através da combinação de uma linguagem simples e autoexplicativa com uma ferramenta de suporte publicamente disponível, acredita-se que DCL possa auxiliar na prevenção da erosão arquitetural.

3. Solução Proposta

Embora processos de conformidade arquitetural identifiquem violações arquiteturais, (i) essas ferramentas são subutilizadas e (ii) violações detectadas são raramente corrigidas. Diante disso, esta seção descreve uma solução de conformidade arquitetural em integração contínua, conforme ilustrada na Figura 1.

[2]http://jenkins-ci.org/

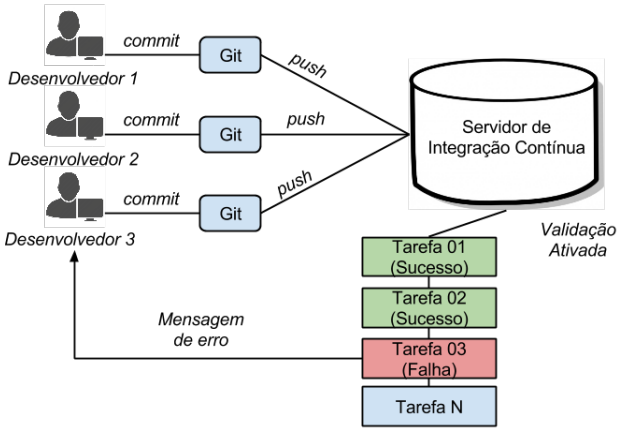


Figura 1. Funcionamento do ArchCI

O processo de conformidade arquitetural é ativado a cada integração de código e, quando violações não forem detectadas, o código poderá ser integrado ao repositório. Isso visa a integridade da arquitetura do software, uma vez que mantém o código-fonte sempre convergente com a arquitetura planejada. Assim, a solução proposta garante que:

- O processo de verificação de conformidade arquitetural seja realizado em toda integração de código sem a necessidade de instalações em máquinas de desenvolvedores, apenas no servidor de CI. Isso visa solucionar o problema *(i)* de subutilização de ferramentas de conformidade arquitetural.
- Como o processo de conformidade arquitetural é ativado a cada integração de código, é possível permitir a integração de código ao repositório apenas quando violações arquiteturais não forem detectadas. Isso visa solucionar o problema *(ii)* de violações detectadas serem raramente corrigidas.

É importante observar que a integração da solução proposta em processos reais de desenvolvimento de software contribuirá diretamente com a qualidade arquitetural do sistema de software, uma vez que a arquitetura implementada (como implementada no código fonte) estará sempre em conformidade com a arquitetura planejada.

3.1. Linguagem DCL

A solução proposta requer uma técnica de conformidade arquitetural subjacente. Para demonstrar a aplicabilidade da solução, utiliza-se DCL para definir as restrições arquiteturais de um projeto. Nessa técnica, define-se *módulos* que são conjunto de classes e, em seguida, *restrições arquiteturais* entre os módulos definidos, conforme Figura 2.

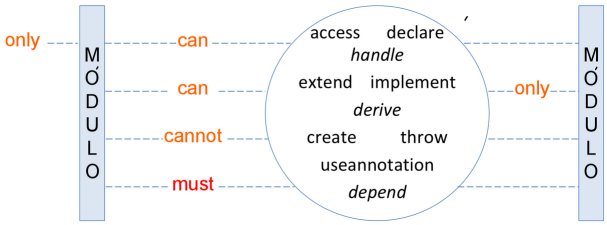


Figura 2. Sintaxe DCL

O exemplo a seguir demonstra a definição e o funcionamento de tais restrições estruturais entre módulos:

```
1: only Factory can-create Product
2: Util can-depend-only $java, Util
3: View cannot-access Model
4: Product must-implement Serializable
```

A restrição da linha 1 especifica que somente classes do módulo `Factory` podem criar objetos de classes no módulo `Product`. A restrição da linha 2 especifica que que as classes do módulo `Util` podem estabelecer dependências somente com o próprio módulo `Util` e a biblioteca padrão da linguagem Java. Já a restrição da linha 3 especifica que as classes do módulo `View` não podem acessar as classes do módulo `Model`. Por último, a restrição da linha 4 especifica que todas as classes no módulo `Product` devem implementar a interface `Serializable`.

Como pode ser observado, é de suma importância a definição de restrições arquiteturais. DCL provê quatro tipos de restrições: *cannot*, *can only*, *only can* e *must*. Assuma os módulos M_A e M_B de um sistema. Assuma também que A e B representem duas classes aleatórias do sistema e que $\overline{M_A}$ representa o complemento de M_A, assim como $\overline{M_B}$ representa o complemento de M_B. Por fim, assuma que `dep` corresponde às possíveis dependências que podem ser especificadas por meio do DCL, como *create*, *access*, *declare*, *handle*, etc. Dessa forma, é possível estipular a seguinte semântica vinculada ao tipo de restrição *cannot*:

$$\exists \mathtt{A}\, \exists \mathtt{B}\, [\, \mathtt{A} \in \mathtt{M_A} \ \wedge \ \mathtt{B} \in \mathtt{M_B} \ \wedge \ \mathtt{dep}(\mathtt{A}, \mathtt{B})\,]$$

Assim, para os tipos de restrição *can only* e *only can*, as semânticas podem ser estipuladas em função da restrição *cannot*:

$$\mathtt{only\ M_A\ can\text{-}dep\ M_B} \implies \overline{\mathtt{M_A}}\ \mathtt{cannot\text{-}dep\ M_B}$$

$$\mathtt{M_A\ can\text{-}only\text{-}dep\ M_B} \implies \mathtt{M_A\ cannot\text{-}dep}\ \overline{\mathtt{M_B}}$$

Por fim, a semântica vinculada ao tipo de restrição *must*:

$$\exists \mathtt{A}\, \neg \exists \mathtt{B}\, [\, \mathtt{A} \in \mathtt{M_A} \ \wedge \ \mathtt{B} \in \mathtt{M_B} \ \wedge \ \mathtt{dep}(\mathtt{A}, \mathtt{B})\,]$$

3.2. Jenkins

A solução proposta requer um servidor de CI subjacente. Para demostrar a aplicabilidade da solução, utiliza-se o servidor Jenkins para programação das tarefas que garantam a conformidade arquitetural das integrações realizadas pelos desenvolvedores. Cada tarefa inclusa no Jenkins refere-se a um projeto de software específico, ou mesmo suas ramificações (seus diferentes *branches*). Sendo assim, cada integração (*push*) realizada ao repositório, dispara um gatilho no servidor, que dará início à tarefa específica do referente projeto, onde esta, por sua vez, validará somente as classes alteradas na determinada integração.

Caso o processo de conformidade arquitetural não detecte violações, as alterações serão integradas ao repositório com sucesso. No entanto, caso violações sejam detectadas, a tentativa de integração será negada, informando ao desenvolvedor que acionou a tarefa, as violações encontradas. Desse modo, a ferramenta garante a propriedade de atomicidade, assegurando que somente as integrações que estejam em total acordo com as restrições de dependência estabelecidas sejam aceitas pelo servidor, rejeitando, assim, alterações que estejam em desacordo ou parcial acordo, mesmo que as mesmas sejam uma série de integrações realizadas ao servidor local (*commits*) antes da requisição de integração ao servidor remoto (*push*).

4. Ferramenta ArchCI

A ferramenta ArchCI implementa a solução proposta, tendo sua concepção voltada para o uso da linguagem DCL como técnica subjacente de conformidade arquitetural e o Jenkins como servidor de CI. Primeiramente, foi necessário criar uma implementação *standalone* de DCL que dependesse apenas de uma biblioteca de manipulação de AST (Eclipse JDT, *Java Development Tools*). Dessa forma, tornou-se possível sua integração ao Jenkins, o que implica que a ferramenta não requeira qualquer instalação em máquina cliente. Além disso, a verificação de conformidade arquitetural das integrações no servidor remoto é realizada de forma individual, verificando cada arquivo separadamente e, mais importante, apenas aqueles que sofreram alguma modificação.

Conforme ilustrado na Figura 3, a implementação de ArchCI segue uma arquitetura com cinco módulos principais:

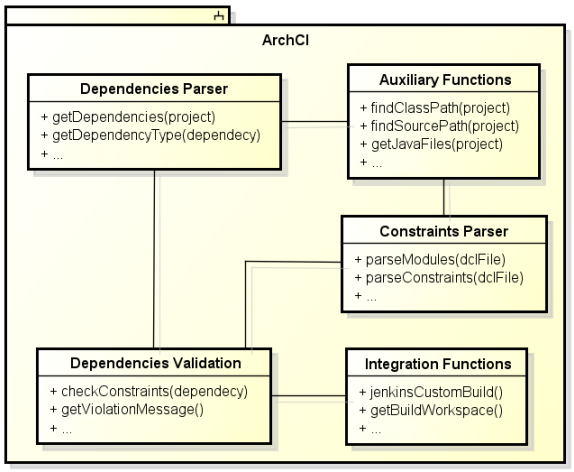


Figura 3. Arquitetura do ArchCI

- *Dependencies Parser*: Módulo responsável pela obtenção das dependências de um projeto, assim como a manipulação das mesmas. Apresenta funções que analisam cada elemento das classes a serem validadas, analisando o tipo de dependência ao qual o determinado elemento se refere.
- *Constraints Parser*: Módulo encarregado da análise e decomposição do arquivo contendo os módulos do projeto e as restrições de dependência estabelecidas para a arquitetura do sistema.

- *Dependencies Validation*: Módulo envolvendo funções para garantir a conformidade arquitetural do projeto por meio da verificação e validação de desvios arquiteturais com base nas restrições de dependência previamente estabelecidas.
- *Auxiliary Functions*: Módulo responsável por fornecer funções que auxiliem as tarefas do ArchCI de modo geral, tais como localizar o caminho das bibliotecas e dos arquivos necessários para a resolução das dependências, identificar o tipo de projeto, etc.
- *Integration Functions*: Módulo contendo as funções relacionadas às práticas de CI, assim como funções necessárias para integrar o código ao servidor Jenkins. Esse, por sua vez, engloba funções para a customização do *build*, obtenção do *workspace* com o código a ser integrado, identificação das classes a serem validadas, etc.

Por fim, após a conformidade arquitetural realizada durante a integração contínua, o ArchCI fornece como retorno uma mensagem de erro juntamente com as violações encontradas nas classes alteradas da integração, caso as mesmas existam. A interface da mensagem e sua representação é demonstrada na Figura 4(c), tendo como base o exemplo de restrição de dependência da Figura 4(a) e a violação da Figura 4(b).

```
module Main: project.main.*

Main cannot-depend java.lang.Math
```

(a) Exemplo de Restrição de Dependência

```
package project.main;

public class Main {
    public static void main(String[] args) {
        System.out.println(Math.pow(2, 5));
    }
}
```

(b) Exemplo de Violação

```
$ git add .
$ git commit -m "Integration Changes"
[dev a57ceb7] Integration Changes
 1 file changed, 4 insertions(+), 4 deletions(-)
$ git push

FAILURE
VIOLATION 1: 'project.main.Main' contains the method 'main' that statically invo
kes the method 'pow' of an object of 'java.lang.Math'
```

(c) Relatório após a tentativa de integração de código

Figura 4. Interface ArchCI

5. Avaliação

Para demonstrar a aplicabilidade da solução proposta, foi conduzida uma avaliação controlada envolvendo a aplicação `myAppointments` [13], um sistema de gerenciamento de informação pessoal simples implementado para ilustrar técnicas de conformidade arquitetural. O sistema possui funcionalidades primárias aos usuários, como criar, recuperar, atualizar e excluir contatos pessoais. Apesar de seu tamanho e sua complexidade serem simplificados, seu conjunto de restrições de dependência são provavelmente utilizados em muitos casos de conformidade arquitetural de projetos reais.

A arquitetura do `myAppointments` segue um padrão bastante conhecido chamado *Model-View-Controller* (MVC), fornecendo uma nítida divisão entre seus componentes. O componente *Model* encapsula o estado da aplicação, enquanto o componente *View* está associado à objetos da interface. O componente *Controller*, por sua vez, faz a mediação de todas as interações entre o *Model* e o *View*. Internamente ao componente *Model*, estão contidos *Domain Objects*, que representam entidades de domínio, e *Data Access Objects* (DAOs), que encapsulam o *framework* de persistência da aplicação. Tal arquitetura é ilustrada pelo diagrama da Figura 5:

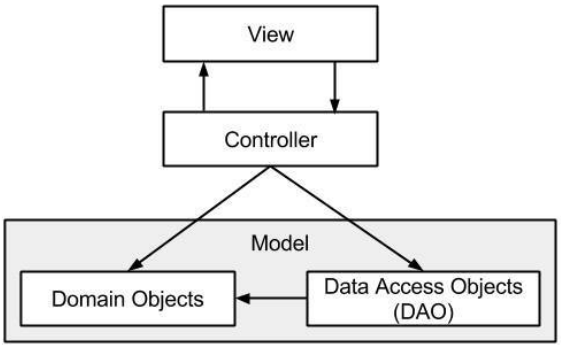


Figura 5. Arquitetura MVC do MyAppointments

Embora simplificado, o sistema do `myAppointments` trabalha com as principais restrições de dependência envolvendo o modelo MVC. Sua implementação usa as seguintes restrições arquiteturais (RA):

(RA1) Somente a camada *View* pode depender dos componentes providos pelo *AWT/Swing*.

(RA2) Somente os DAOs da camada *Model* podem depender dos serviços de banco de dados. Uma exceção é concedida para a classe `model.DB`, responsável por controlar as conexões do banco de dados.

(RA3) A camada *View* pode depender apenas dos serviços providos por ela mesma, pela camada *Controller* e pelo pacote *Util* (por exemplo, para dissociar a apresentação dos dados do acesso aos dados, componentes do *View* não podem acessar componentes do *Model* diretamente).

(RA4) *Domain Objects* não devem depender dos módulos DAO, *Controller* e *View*.

(RA5) Classes DAO podem depender somente de *Domain Objects*, das classes *Model* autorizadas a utilizar os serviços de banco de dados (como o `model.DB`), quanto do pacote *Util*.

(RA6) O pacote *Util* não pode depender de nenhuma classe específica do código fonte do sistema.

Para utilização da solução proposta, a definição das restrições arquiteturais em DCL é demonstrado na Figura 6.

```
%Módulos
module Controller:    myappointments.controller.*
module View:          myappointments.view.*
module Model:         myappointments.model.**
module Domain:        myappointments.model.domain.*
module Util:          myappointments.util.*
module DB:            myappointments.model.DB
module DAO:           "myappointments.model.[a-zA-Z0-9/.]*DAO"
module JavaAwtSwing:  java.awt.**, javax.swing.**
module JavaSql:       java.sql.**

%Restrições
only View can-depend JavaAwtSwing
only DAO, DB can-depend JavaSql
View cannot-depend Model
Domain can-depend-only $java
DAO can-depend-only Domain, Util, javaSql
Util cannot-depend $system
```

Figura 6. Restrições Arquiteturais DCL do MyAppointments

Para avaliar o funcionamento da solução proposta, foram intencionalmente criadas seis violações arquiteturais, uma para cada restrição de dependência do `myAppointments`, conforme ilustrado na Figura 7.

```
package myappointments.model.domain;

import java.util.Date; [...]

public class Appointment  {

    public Appointment() throws Exception{

        javax.swing.JOptionPane dialog;

        Date date = new java.sql.Date(2015, 04, 19);

        myappointments.controller.AgendaController ac;

    }
    [...]
```

(a) Violações RA1, RA2 e RA4

```
package myappointments.view;

import myappointments.model.AgendaDAO; [...]

public class AppointmentView extends
        AbstractAppointmentView {
    public AppointmentView
            (AppointmentController controller)
                throws Exception {
        this.controller = controller ;
        this.appForm = new AppointmentForm() ;

        Object aDAO = AgendaDAO.getInstance();

        initComponents() ;
        [...]
```

(b) Violação RA3

```
package myappointments.model;

import myappointments.view.AppointmentView; [...]

public class AgendaDAO extends AbstractAgendaDAO  {

    private static AgendaDAO agendaDAO
        = new AgendaDAO();

    private static AppointmentView av;
    [...]
```

(c) Violação RA5

```
package myappointments.util;

import myappointments.view.AppointmentView; [...]

public class DateUtils {

    public static final String HOUR_FMT = "HH:mm";
    public static final String SHORT_DATE_FMT =
        "MM/dd/yyyy";
    public static final String LONG_DATE_FMT =
        "MM/dd/yyyy HH:mm";

    private static AppointmentView av;
    [..]
```

(d) Violação RA6

Figura 7. Violações introduzidas no MyAppointments

(RA1) Um variável do tipo `javax.swing.JOptionPane` foi declarada dentro da classe `Appointment` que pertence a camada *Domain*. Isso representa uma violação na restrição (RA1) que indica que *somente a camada View pode depender dos componentes providos pelo* `AWT/Swing` (vide Figura 7(a)).

(RA2) Foi instanciado um objeto do tipo `java.sql.Date` na classe `Appointment` da camada *Domain*, violando a restrição (RA2) de que *somente os DAOs da camada Model podem depender dos serviços de banco de dados* (vide Figura 7(a)).

(RA3) O método `getInstance()` do objeto `AgendaDAO`, pertencente à camada DAO foi invocado pela classe `AppointmentView` da camada *View*, violando a restrição (RA3) de que *a camada View pode depender apenas dos serviços providos por ela mesma, pela camada Controller e pelo pacote Util* (vide Figura 7(b)).

(RA4) A classe `Appointment` da camada *Domain* instancia a variável `ac` do tipo `AgendaController` (pertencente à camada *Controller*), violando a restrição (RA4) de que *Domain Objects não devem depender dos módulos DAO, Controller e View* (vide Figura 7(a)).

(RA5) A classe `AgendaDAO` presente na camada DAO contém o campo `av` do tipo `AppointmentView` (pertencente à camada *View*), violando a restrição (RA5) de que *classes DAO podem depender somente de Domain Objects, das classes Model autorizadas a utilizar os serviços de banco de dados, quanto do pacote Util* (vide Figura 7(c)).

(RA6) Assim como na violação anterior, a classe `DateUtils` presente na camada *Util* contém o campo `av` do tipo `AppointmentView` (pertencente à camada *View*), violando a restrição (R6) de que *o pacote Util não pode depender de nenhuma classe específica do código fonte do sistema* (vide Figura 7(d)).

Dessa forma, foi realizada a avaliação do sistema para verificar se a ferramenta proposta é capaz de realizar o processo de conformidade arquitetural corretamente, detectando todas as violações criadas. O resultado da aplicação da ferramenta é demonstrado na Figura 8. Conforme observado, o ArchCI foi capaz de encontrar todas as violações criadas para esta avaliação com sucesso, cancelando o processo de integração (*push*) e informando as referentes violações ao desenvolvedor, cumprindo assim, seu propósito.

```
$ git add .
$ git commit -m "MyAppointments Changes"
[dev 8976d37] MyAppointments Changes
 23 files changed, 2189 insertions(+), 39 deletions(-)
$ git push

FAILURE
VIOLATION 1: 'myappointments.model.domain.Appointment' contains the local variable 'dialog' i
n method 'Appointment' whose type is 'javax.swing.JOptionPane'
VIOLATION 2: 'myappointments.model.domain.Appointment' contains the method 'Appointment' that
 creates an object of 'java.sql.Date'
VIOLATION 3: 'myappointments.model.domain.Appointment' contains the local variable 'ac' in me
thod 'Appointment' whose type is 'myappointments.controller.AgendaController'
VIOLATION 4: 'myappointments.view.AppointmentView' contains the method 'AppointmentView' that
 statically invokes the method 'getInstance' of an object of 'myappointments.model.AgendaDAO'
VIOLATION 5: 'myappointments.model.AgendaDAO' contains the field 'av' whose type is 'myappoin
tments.view.AppointmentView'
VIOLATION 6: 'myappointments.util.DateUtils' contains the field 'av' whose type is 'myappoint
ments.view.AppointmentView'
```

Figura 8. Violações detectadas pelo ArchCI no MyAppointments

Limitações: A avaliação foi realizada em um ambiente controlado – um sistema de pequeno porte, um único desenvolvedor, poucas integrações de código e um pequeno conjunto de violações. Entretanto, o objetivo da avaliação de se verificar a aplicabilidade da solução proposta foi atingido ao se demonstrar que é sim possível integrar um processo de conformidade arquitetural em Integração Contínua.

6. Conclusão

É de suma importância para a engenharia de software garantir a conformidade arquitetural de um sistema, principalmente no desenvolvimento de software em conjunto, onde problemas como a erosão arquitetural tornam-se mais comuns, causando a anulação de características como manutenibilidade, reusabilidade, escalabilidade, portabilidade, etc.

Este artigo apresenta uma solução para a verificação da conformidade arquitetural de um projeto de software – com base em restrições arquiteturais entre módulos – incorporadas em um servidor de Integração Contínua. Como principal contribuição, a solução proposta evita os problemas decorrentes de um processo de erosão arquitetural através de um processo de conformidade arquitetural mais rígido, e.g., integrações de código só ocorrem quando não foram detectadas violações arquiteturais.

Como trabalho futuro, pretende-se: (i) aplicar a solução proposta em cenários reais de desenvolvimento a fim de avaliar sua expressividade, aplicabilidade e desempenho; (ii) avaliar a usabilidade da ferramenta, e.g., melhor forma de se realizar a verificação, melhor forma de se apresentar as violações, além das características mais importantes para aceitação dos desenvolvedores, considerando distintas abordagens para o reporte de violações, e.g., exibição de *warnings* ou envio de e-mails, ao invés de bloquear a integração de código; (iii) analisar como fatores humanos influenciam as violações para propor novas funcionalidades; e (iv) realizar melhorias na implementação da ferramenta.

Agradecimentos

Este trabalho foi apoiado pela FAPEMIG, CAPES e CNPq.

Referências

[1] Jonathan Aldrich, Craig Chambers, and David Notkin. ArchJava: Connecting software architecture to implementation. In *24th International Conference on Software Engineering (ICSE)*, pages 187–197, 2002.

[2] Brian De Alwis and Jonathan Sillito. Why are software projects moving from centralized to decentralized version control systems? In *2nd Cooperative and Human Aspects on Software Engineering (CHASE)*, pages 36–39, 2009.

[3] Alan Berg. *Jenkins Continuous Integration Cookbook*. Packt Publishing, Birmingham, 2012.

[4] Jon Bowyer and Janet Hughes. Assessing undergraduate experience of continuous integration and test-driven development. In *28th International Conference on Software Engineering (ICSE)*, pages 691–694, 2006.

[5] João Brunet, Dalton Serey, and Jorge Figueiredo. Structural conformance checking with design tests: An evaluation of usability and scalability. In *27th International Conference on Software Maintenance (ICSM)*, pages 143–152, 2011.

[6] Lakshitha de Silva and Dharini Balasubramaniam. Controlling software architecture erosion: A survey. *Journal of Systems and Software*, 85(1):132–151, 2012.

[7] Paul M. Duvall, Steve Matyas, and Andrew Glover. *Continuous Integration: Improving Software Quality and Reducing Risk*. Pearson Education, Boston, 2007.

[8] Martin Fowler and Matthew Foemmel. Continuous integration. Technical report, Thought-Works, 2006.

[9] Konrad Hinsen, Konstantin Läufer, and George K. Thiruvathukal. Essential tools: Version control systems. *Computing in Science & Engineering*, 11(6):84–91, 2009.

[10] 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)*, pages 18–28, 1995.

[11] Oscar Nierstrasz and Mircea Lungu. Agile software assessment. In *20th International Conference on Program Comprehension (ICPC)*, pages 3–10, 2012.

[12] Bryan O'Sullivan. Making sense of revision-control systems. *Queue*, 7(7):30–40, 2009.

[13] Leonardo Passos, Ricardo Terra, Renato Diniz, Marco Tulio Valente, and Nabor Mendonça. Static architecture conformance checking: An illustrative overview. *IEEE Software*, 27(5):132–151, 2010.

[14] D. E. Perry and A. L. Wolf. Foundations for the study of software architecture. *Software Engineering Notes*, 17(4):40–52, 1992.

[15] Neeraj Sangal, Ev Jordan, Vineet Sinha, and Daniel Jackson. Using dependency models to manage complex software architecture. In *20th Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, pages 167–176, 2005.

[16] John Ferguson Smart. *Jenkins: The Definitive Guide*. O'Reilly Media, Inc, Sebastopol, 2011.

[17] Diomidis Spinellis. Version control, part 1. *IEEE Software*, 22(5):107–107, 2005.

[18] Diomidis Spinellis. Version control, part 2. *IEEE Software*, 22(6):c3–c3, 2005.

[19] Ricardo Terra and Marco Tulio Valente. A dependency constraint language to manage object-oriented software architectures. *Software: Practice and Experience*, 39(12):1073–1094, 2009.

[20] Ricardo Terra and Marco Tulio Valente. Definição de padrões arquiteturais e seu impacto em atividades de manutenção de software. In *VII Workshop de Manutencao de Software Moderna (WMSWM)*, pages 1–8, 2010.

[21] Mathieu Verbaere, Michael W. Godfrey, and Tudor Gîrba. Query technologies and applications for program comprehension. In *16th IEEE International Conference on Program Comprehension*, pages 285–288, 2008.

AtlasSPL - A Web-Based Tool for Feature Modeling

Marcelo Schmitt Laser[1], **Elder Macedo Rodrigues**[1], **Cristiano Moreira Martins**[1], **Flávio Oliveira**[1]

[1]School of Computer Science (FACIN)
Pontifical Catholic University of Rio Grande do Sul (PUCRS)
Postal Code 90.619-900 – Porto Alegre – RS – Brazil

{marcelo.laser, cristiano.martins}@acad.pucrs.br

{elder.rodrigues, flavio.oliveira}@pucrs.br

Abstract. *This paper presents a graphical web-based feature modeling tool called AtlasSPL that aims to facilitate the training of new professionals in basic Software Product Line design concepts. AtlasSPL provides an intuitive graphical interface for the construction and visualization of feature models, is an accessible solution, gives access to a repository of previously-validated feature models, and supports five different feature model notations. We believe that these qualities define it as a highly accessible solution for both educational and small-scale industrial settings.*

1. Introduction

Software Product Line Engineering (SPLE) is a technique that has grown considerably in recent years. Due to allowing for a great degree of systematic reuse, as well as decreasing long-term costs and time-to-market, it has proven itself as a valuable technique in the design, development and management of software where a core of commonalities can be identified between different products [Clements and Northrop 2001] [Pohl et al. 2005]. A key concept of a Software Product Line (SPL) is the definition of variability, this being the characteristics (or features) that vary from one product to another within a single family of software products.

The clear and expressive representation of variability during the design of an SPL is invaluable to its continued success [Pohl et al. 2005]. While there are different techniques to aid in this task, Feature Modeling [Kang et al. 1990] [Czarnecki and Eisenecker 2000] is one of the most widely used [Berger et al. 2013], allowing for a graphical representation of variability that is recognisable by all stakeholders. Although there are several tools that aid in feature modeling, none of them meet our usability and availability requirements.

Various notations have been proposed to represent Feature Models, most of them based on FODA [Kang et al. 1990]. Some of the most notable ones in the literature are the Czarnecki-Eisenecker base [Czarnecki and Eisenecker 2000] and extended [Czarnecki et al. 2005] notations, the FeatuRSEB notation [Griss et al. 1998] and the Gurp-Bosch-Svahnberg notation [van Gurp et al. 2001]. Despite having important differences in presentation, they share a common set of semantics that can be exploited in the construction of a feature modeling tool.

Over the course of our work, we have often run into difficulties when having to build and present feature models, for reasons ranging from lack of knowledge from our

collaborators to lack of support in the tools available to us. To resolve this issue, we have defined a set of requirements, based on our expertise in this field, for a tool to support feature modeling with the purpose of basic education in software variability and feature modeling, as well as facilitating the collaboration between professionals who may not be acquainted with abstract structures such as trees.

In this paper we present AtlasSPL, a web-based feature modeling tool built to meet these requirements. AtlasSPL was created with the primary purpose of training our collaborators in the use of feature models, and should therefore be a suitable tool for educational environments. We believe that it is also sufficiently complete to allow for the basic feature modeling activities of managing SPL variability, and is therefore suitable for small-scale industrial use.

The rest of this paper is organized as follows: Section 2 presents some background information regarding feature models and their applicability in Software Engineering, as well as some tools that aid in the feature modeling process; Section 3 presents our in-house feature modeling tool (AtlasSPL), along with a summary of the requirements on which this tool's design is based and an example of its application through the modeling of a pedagogical SPL; Section 4 presents some lessons learned from the development of these tools as well as some proposals of future work; Section 5 presents a summary of our conclusions based on the material presented in this paper.

2. Background

In this section we briefly present the basic concepts of feature modeling. This is followed by the descriptions of four tools that support the feature modeling process, along with the merits and flaws that we have perceived in them. Finally, we provide a contextualization of our own experiences in this field of research, along with the motivations for the development of a new feature modeling tool.

According to [Czarnecki and Eisenecker 2000], "a feature is an important property of a concept instance", representing any commonality or difference between the different products of an SPL. A model that groups the representation and relationships between the features of an SPL is called a Feature Model, which can be comprised of one or more Feature Diagrams. Feature Models are commonly used for the representation of variability in SPLs, both for their simplicity and expressiveness, and for the ease with which they can be explained to stakeholders, specially those from outside the field of Software Engineering.

A feature model is typically comprised of a root feature, representing the domain of the SPL, and several child features representing variation points of this domain. These variation points may in turn have child features of their own, either to further subdivide them into smaller variation points or to represent variants available to the SPL.

2.1. Related Feature Modeling Tools

Although a number of tools exist that aid in the feature modeling process, the better portion of them appears to be experimental or incomplete. Prior to our decision to develop a tool of our own, as well as to guide us in its design, we have identified three major tools that stand out as being in a more mature stage. We have also used a tool previously

designed by our own research group during previous projects as a basis for the conception of our work.

FeatureIDE is a framework based on Eclipse and developed to support Feature-Oriented Software Development (FOSD) [Thüm et al. 2014]. Although FeatureIDE is undeniably a powerful tool, used primarily for the purposes of teaching and research, its target user is a professional who already has a certain knowledge of feature modeling and SPL concepts. Conversely, our target users are students and professionals who are entirely unacquainted with these concepts, and our target activities include the design and visualization of SPL features.

FeaturePlugin is a feature modeling plug-in for the Eclipse IDE, and proposes to integrate feature modeling with a complete and established development environment [Antkiewicz and Czarnecki 2004]. It provides support to feature modeling through the use of tree structures and logical statements, and integrates feature modeling and feature-based configuration. Though it is a complete solution to feature modeling, it is restricted to a single environment (Eclipse) and does not provide a graphical editor, while we seek greater intuitiveness and compatibility.

The Software Product Line Online Tools (S.P.L.O.T.) are a collection of web-based tools that include a feature model editor with advanced constraint definition and validation techniques [Mendonça et al. 2009]. S.P.L.O.T. contains a vast repository of models which are available to the public, making it a substantial information base for feature modeling practices. The tool does not, however, offer any means of graphical construction of feature models, being based instead on a tree structure. The use of this kind of structure could be inadequate for presentation purposes where the audience is unacquainted with certain basics of Computer Theory, such as tree structures.

PlugSPL, the in-house tool on which AtlasSPL is largely based has been described extensively elsewhere [de M. Rodrigues et al. 2014]. Regardless, it is important to assert those characteristics of it that are specific to the SPL Design activity, both in regards to what is being adopted from it as to what components had to be developed or adapted for reuse. It is also important to highlight that PlugSPL is a stand-alone tool that supports several phases of the SPL development cycle, and was therefore built around very different requirements from AtlasSPL.

Our previous approach to the development of the SPL Design module, and indeed for the entire tool, had been to develop an extensible plugin-based tool written in C# and run directly from the user's machine. Our experience with this previous project has shown us that for the purposes of accessibility and ease of collaboration, it would be advantageous to turn instead to a web-based User Interface (UI). Though the original architecture had been made in a modular manner, such a drastic change to the mode of input/output creates certain issues that have to be addressed.

Another important difference between PlugSPL and AtlasSPL is the decision to have native support of a number of feature model notations, allowing each user to decide at the time of a model's creation which one to base it on. We believe that this allows for greater accessibility, as well as enabling the user to learn about the more common notations found in the literature.

Finally, we have chosen to reuse some of the assets created for PlugSPL, *viz.* the

persistence structures and the validation code executed in their construction. Adapting these structures for use in a broader scope (more than one notation) and to operate with web-based technologies was yet another challenge in the design and development of our new tool.

2.2. Context

Over the course of our group's research into Software Testing[1], we have developed a family of Software Testing Tools by use of SPL practices [de M. Rodrigues et al. 2010] [Silveira et al. 2011] [Costa et al. 2012]. Given the volatile nature of our research environment, we have had to repeat certain processes several times, particularly in what relates to product configuration, product generation [de M. Rodrigues et al. 2014].

In order to eliminate these repeated efforts, we have proposed and developed an in-house tool (PlugSPL) that supports several phases of the SPL development cycle [de M. Rodrigues et al. 2014]. Among the requirements of that tool we identified the support for graphical-based notation for designing feature models. Though the resulting tool was satisfactory in resolving the requirements we had at the time, it was made primarily for use by a core group of the research team that already had experience with feature modeling. Over the course of the following years, we have found that PlugSPL could be difficult to use by new collaborators, particularly in what refers to the SPL Design activity.

Given that feature modeling is a crucial activity during SPL design, we sought out means by which to train our collaborators in their use. Furthermore, it is widely accepted that feature models are of easier presentation to stakeholders who are not versed in variability analysis than other similar representations [de M. Rodrigues et al. 2014]. For this reason, we also sought tools that would allow collaboration during the design of feature models.

Despite their merits, the results that we turned up in regards to actual tools were scarce. Some major qualities that were lacking in these tools were simplicity, ease of use and accessibility, which rendered them inadequate to our needs in feature modeling practices. We also found the existing tools lacking in terms of presentation due to their complexity. In order to facilitate the understanding of variability management, as well as to provide a shared and collaborative environment for feature modeling, we decided to create an in-house web-based tool for graphical feature modeling.

3. AtlasSPL - Feature Model Editor Web-Tool

AtlasSPL is a web-based feature modeling tool that was designed and developed with the qualities of ease-of-use and versatility at the forefront of our team's goals. It is the culmination of six years of study and experience with SPL design, working with a volatile team whose members were constantly being shifted in and out of projects, with new members coming into the team often without any knowledge of SPL or feature modeling concepts. With AtlasSPL, we hope to greatly facilitate the training of new professionals in feature modeling and design, as well as basic notions of Software Product Lines.

This section presents: the design decisions taken for the development of AtlasSPL, as well as the requirements that drove them; a complete description of the functionalities

[1] www.cepes.pucrs.br

of the tool as well as the ways to access it; a brief description of the feature model notations supported by the tool, and; an explanation of the validation process applied by AtlasSPLL.

3.1. Requirements and Design Decisions

In order to present the current state of our studies in feature modeling practices, as well as the needs we perceived for the training of our collaborators, we have defined some requirements that must be addressed by a feature modeling tool. These requirements refer primarily to the usability and functionality of the tool, and do not largely concern themselves with architectural or performance constraints.

In our context, the first and most important requirement is ease of access and portability. If the tool is dependent on a particular machine configuration, operating system, language, development environment or framework, or if it requires prior installation and preparation, it could be limited in its applicability. This requirement has been the driving reason to our decision of making a web-based tool, resolving all of these dependencies so long as the client machine has a connection to the Internet.

Another requirement is the availability of multiple feature modeling notations, which is particularly important for processes of training and collaborative work. We judge that training in more than one feature model notation is essential to the comprehension of the basic feature model semantics, enabling the trained professional to design more complete and expressive models. Furthermore, the collaborative nature of SPLs demands that professionals be able to apply different views of the same concept in order for a better exchange of ideas to be reached.

The tool must also possess an intuitive user interface based on the graphical representation of feature modeling objects. Given that feature models are ultimately a graphical artifact, we find it important for the user to be given a “canvas” with which to represent them, rather than depending on textual input and tree structures for their design.

It is valuable, particularly in a tool directed at training and education, that there be feature model validation functionalities. Given that our aim is to provide the very first steps of training in feature modeling practices, we have opted to provide support to structural validation. To achieve this we have largely reused one of our research team’s previous assets.

Finally, we believe it advantageous to have an easily accessible repository of reference models. We have opted to give the user access to previously-validated models, as well as the option to add new models to the public repository. The repository also has the option of allowing users to create private access groups, permitting the collaboration between closed teams without the support of file-sharing solutions.

3.2. Tool Functionalities

AtlasSPL provides the user with the most basic operations required for feature modeling, these being: adding, moving and removing features and relationships; setting feature properties; renaming features; creating, saving and loading feature models, and; validating feature models. These functions were built into the tool in the way the authors judged to be the most intuitive for the user. Figure 1 shows the user interface for AtlasSPL and

presents an example feature model in construction, drawn from the Arcade Game Maker Pedagogical Product Line (AGM) and using the FODA notation.

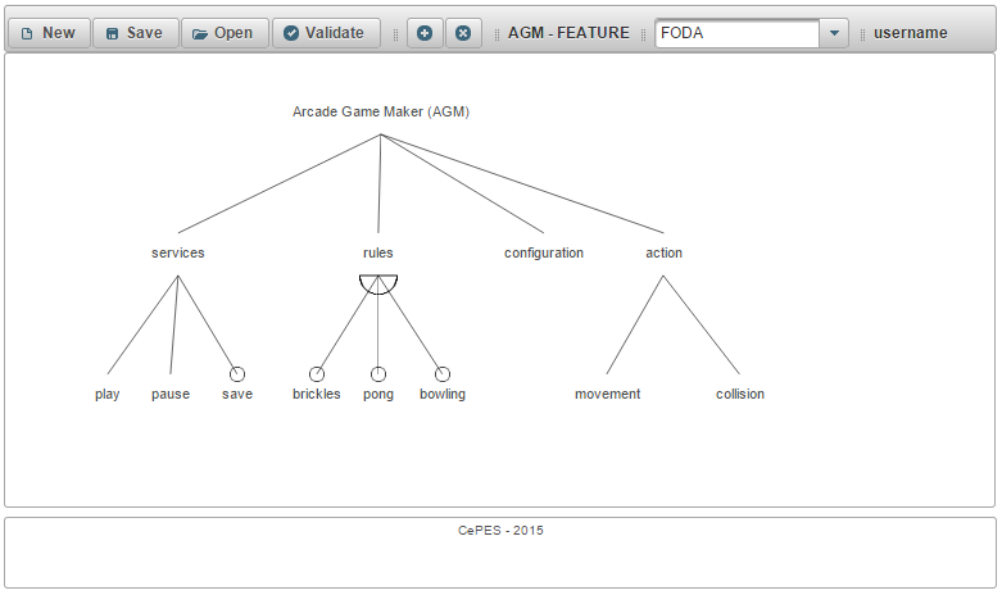


Figure 1. AtlasSPL Webtool - AGM Feature Model using FODA

Upon clicking the "New" button, a dialogue menu is opened (see Figure 2) where the user may name the new feature model and select the feature model notation to be used (refer to Subsection 3.3 for the available notations).

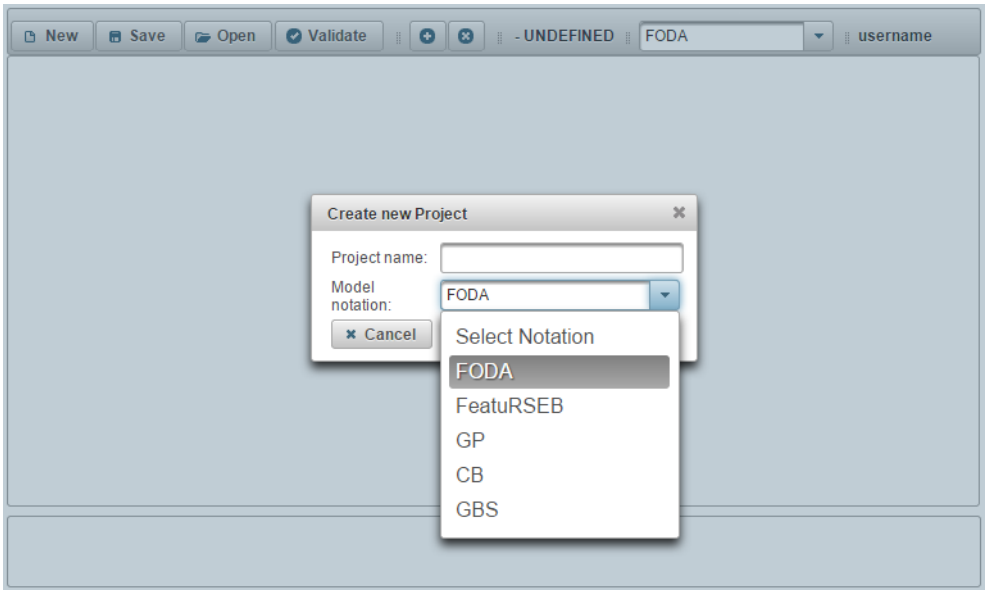


Figure 2. New feature model dialogue menu

There are three ways for the user to add a new feature to the model, allowing for greater efficiency. The user may click the "+" button and then click anywhere inside the canvas. The user may also right-click anywhere inside the canvas and select "New Feature" from the context menu. Finally, the user may double-click anywhere inside the

canvas. Any of these actions will result in a feature object being created at the current cursor position and a dialogue menu being opened to name it.

To remove a feature, the user must right-click on that feature and select “Remove Feature” from the context menu. In future versions of AtlasSPL, it will also be possible to select a feature by clicking it and then deleting the selected feature. To move a feature, the user may click it and drag it over the canvas. To rename it, the user must right-click the feature and select “Feature Name” from the context menu.

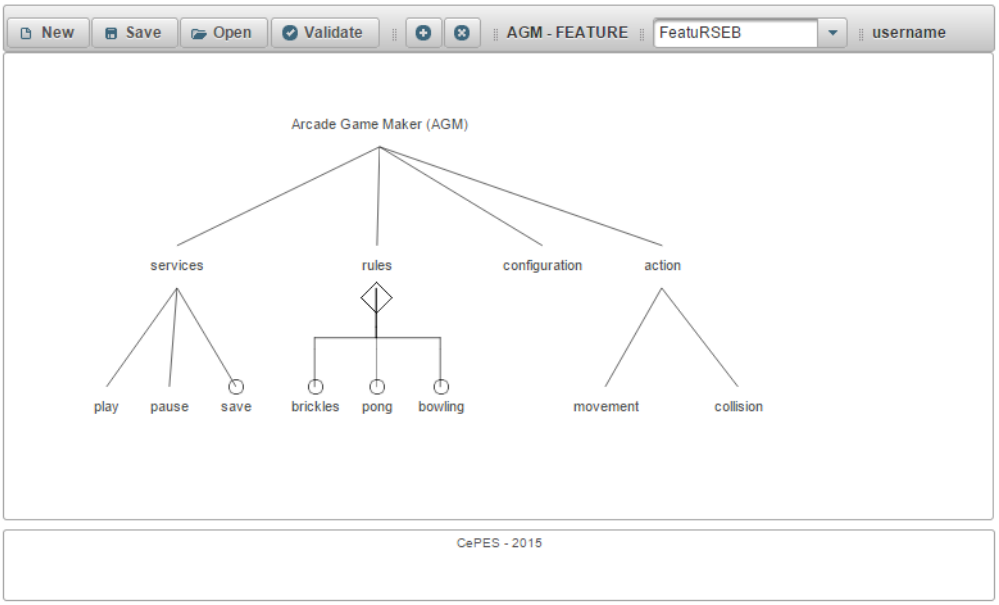


Figure 3. AtlasSPL Webtool - AGM Feature Model using FeatuRSEB

To add a relationship, the user must click anywhere upon the lower edge of the desired parent feature (the area will be highlighted when moused over) and drag the cursor towards the child feature. By clicking on an existing relationship and dragging the cursor over to another feature, the relationship’s child feature may be changed. By clicking on an existing relationship and dragging the cursor to an empty area of the canvas, the relationship is removed.

The appearance of a relationship is dependent on the properties of its child feature and the selected feature model notation. For example, Figure 1 shows the alternative relationship between the subfeature *rules* and its variants *brickles*, *pong* and *bowling* as a semi-circle, in accordance to the FODA notation. Meanwhile, Figure 3 represents this as an XOR relationship by using the unfilled diamond shape and straight lines, in accordance with FeatuRSEB.

To set a feature’s properties one must right-click it and select the desired relationship type from the context menu. The available relationship types vary according to the feature model notation in use.

Finally, to conduct the structural validation of a feature model, the user must click the “Validate” button, which will attempt to reconstruct the diagram currently on the canvas. After the validation process is completed, a dialogue box will open to present the results, either stating “Valid Diagram” or “Invalid Diagram”. Figure 4 shows an example

of the tool declaring a feature model as invalid.

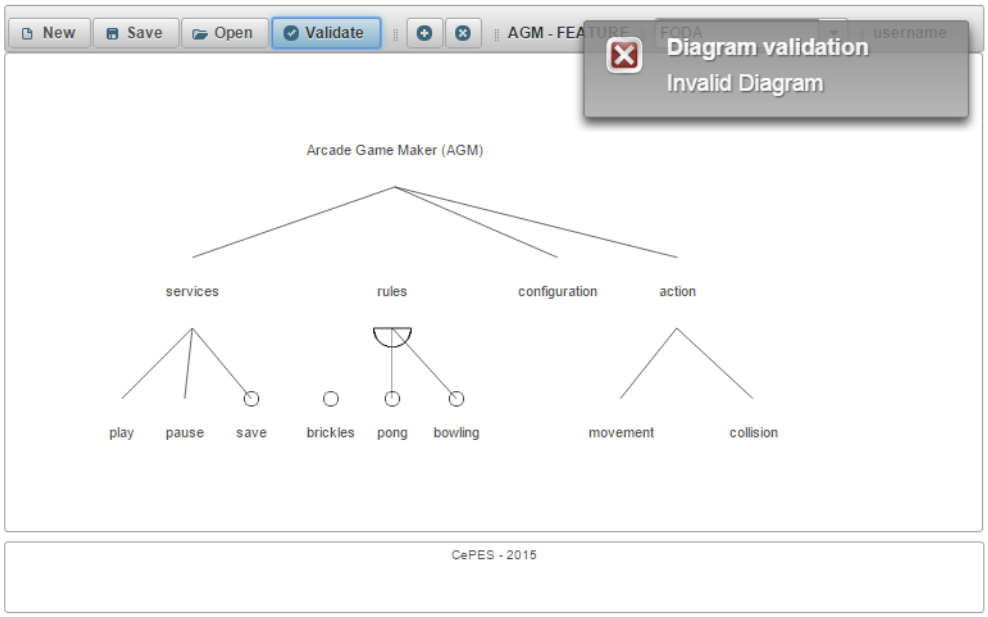


Figure 4. Validation dialogue box

For a more thorough validation, presenting data such as the number of valid configurations that may be derived from the feature model and the number of dead features within that model, AtlasSPL allows the user to export a feature model in the S.P.L.O.T. native XML format.

3.3. Adopted Feature Model Notations

We have selected five feature model notations for which to provide native support. This selection was based on our own experience, as well as brief literature searches by the authors into surveys of and comparisons between feature model notations. The selected notations are:

- FODA [Kang et al. 1990];
- Czarnecki-Eisenecker Base Notation [Czarnecki and Eisenecker 2000];
- Czarnecki-Eisenecker Extended Notation [Czarnecki et al. 2005];
- FeatuRSEB [Griss et al. 1998];
- Gurp-Bosch-Svahnberg Notation [van Gurp et al. 2001].

The Feature-Oriented Domain Analysis notation (FODA), which is the original feature modeling notation, can be used as a basis for the study of feature model semantics. It presents all of the main components of a feature model, these being features themselves, relationships, the notions of mandatory (must be present in all product configurations) and optional (may or may not be present in a product configuration) features, and the notion of alternative (mutually exclusive) features.

The Czarnecki-Eisenecker Base notation (also known as the Generative Programming notation, hereby referred to as GP) has been broadly used as the primary reference notation for feature models. Building on FODA, GP permits all of the original components of its predecessor using different graphical representations, and proposes new

components to allow for greater expressiveness. The alternative features are now called an XOR feature group, to differentiate them from the new OR feature group (one or more may be present in a product configuration).

The Czarnecki-Eisenecker Extended notation (also known as the Cardinality-based notation, hereby referred to as CB) is an extension of the GP notation, presenting a new and more precise form by which to graphically represent relationships between features. By allowing the user to define a specific cardinality for each relationship, it is possible to precisely determine the occurrence of features in product configurations.

FeatuRSEB proposes to integrate FODA with Reuse-Driven Software Engineering Business (RSEB), an UML-based process, and presents feature models primarily from the point of view of the developer, or "reuser". FeatuRSEB deals with binding time directly by specifying whether an alternative relationship is resolved statically (XOR) or dynamically (OR).

The Gurp-Bosch-Svahnberg notation (GBS) proposes a framework to organize the approaches that existed at the time into. This notation is also primarily characterized by the definition of binding time within relationships, as well as presenting the idea of external features, features that are provided by a certain configuration's target platform and are therefore important to that configuration, but which are not part of the system itself.

3.4. Feature Model Validation

AtlasSPL provides structural validation of feature models by use of a number of procedures that check the consistency of a model by reconstructing it in the form of a restrictive data structure. This data structure was designed to only allow the instantiation of feature models if they are built in accordance to the guidelines of a given notation. Any structural inconsistencies in a feature model will result in a failure to instantiate this data structure, which AtlasSPL responds to by declaring the feature model as invalid.

While the algorithms by which this structural validation is executed are beyond the scope of this work, we present now the particular inconsistencies that are validated by AtlasSPL.

- There must be only one root feature within a feature diagram. Should there be more than one root feature in a single diagram, the feature model is declared invalid.
- All features must be connected by relationships. Should there be any features that have not been connected to the rest of the diagram, the feature model is declared invalid.
- All features must be reachable from the root feature through a single set of relationships. If AtlasSPL identifies more than one path from the root feature to any other given feature, the feature model is declared invalid.
- All features must be given a name (label) and all names must be different from one another. Should any two features share the same name, or should any one feature have the empty string for a name, the feature model is declared invalid.

4. Future Work

This section presents the lessons learned during the development of this research, as well as prospects for future research based on them.

- Given the successful application of feature modeling concepts within a web environment that was achieved by this project, the authors believe it would be advantageous to expand upon this initiative and build modules to support other parts of the SPL development cycle. While feature modeling has proven to be one of the most difficult topics in training new collaborators, it is also important to acquaint them with other SPL concepts.
- The extension of AtlasSPL to deal with the design and validation of logical constraints may prove useful for more advanced training. Furthermore, it would enable the tool to fully support the SPL Design activity within a production setting.
- The creation of a common data structure for multiple feature model notations may enable AtlasSPL to serve as a centralizing tool for feature modeling. It is likely that we may approach this matter through the analysis of semantic commonalities between notations, as well as the creation of parsers for existing feature modeling tools.
- We are aware that the tool must be improved to meet the requirements of different environments, especially in regards to providing better support for Feature Model analysis and validation. Moreover, the tool must provide detailed feedback to the user regarding any modeling mistakes so as to be usable by inexperienced users without the presence of an experienced tutor.
- Studies to empirically evaluate the success of AtlasSPL in achieving its proposed goals would solidify it as an educational tool. The authors are particularly interested in studying the intuitiveness of its UI, as well as the degree by which it aids in the teaching of basic feature modeling concepts.

5. Conclusion

AtlasSPL is a feature modeling web-tool proposed for the purpose of easing the training of professionals entirely unacquainted with feature model and SPL concepts. It aims to achieve this goal by providing a simple and accessible interface, based on graphical components to aid in the construction of feature models as well as their visualization. Furthermore, AtlasSPL was designed to be sufficiently complete so as to serve as an initial tool for the inception and presentation of new SPLs.

The tool offers native support to five different feature modeling notations, these being the Feature-Oriented Domain Analysis notation, the Czarnecki-Eisenecker base and extended notations, the FeatuRSEB notation and the Gurp-Bosch-Svahnberg notation. By allowing the user to choose between these notations, AtlasSPL enables different degrees of precision and simplicity, as well as different semantics to facilitate collaboration between users familiar with different notations.

Acknowledgements

Study partially developed by the Research Group of the PDTI 001/2012, financed by Dell Computers of Brazil Ltd. with resources from Law 8.248/91. We thank Dell for the support in the development of this work.

References

Antkiewicz, M. and Czarnecki, K. (2004). FeaturePlugin: Feature Modeling Plug-in for Eclipse. In *Proceedings of the 2004 OOPSLA Workshop on Eclipse Technology eXchange*, pages 67–72, New York, NY, USA. ACM.

Berger, T., Rublack, R., Nair, D., Atlee, J. M., Becker, M., Czarnecki, K., and Wasowski, A. (2013). A survey of variability modeling in industrial practice. In *Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems*, New York, NY, USA. ACM.

Clements, P. C. and Northrop, L. (2001). *Software Product Lines: Practices and Patterns*. SEI Series in Software Engineering. Addison-Wesley.

Costa, L. T., Czekster, R., Oliveira, F. M., Rodrigues, E. M., Silveira, M. B., and Zorzo, A. F. (2012). Generating Performance Test Scripts and Scenarios Based on Abstract Intermediate Models. In *24th International Conference on Software Engineering and Knowledge Engineering*, pages 112–117.

Czarnecki, K. and Eisenecker, U. (2000). *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, Boston, MA.

Czarnecki, K., Helsen, S., and Eisenecker, U. (2005). Staged configuration through specialization and multi-level configuration of feature models. In *Software Process Improvement and Practice*.

de M. Rodrigues, E., Passos, L., Teixeira, F., Zorzo, A. F., and Saad, R. (2014). On the Requirements and Design Decisions of an In-House Component-based SPL Automated Environment. In *26th International Conference on Software Engineering and Knowledge Engineering*, pages 483–488.

de M. Rodrigues, E., Viccari, L. D., Zorzo, A. F., and Gimenes, I. M. (2010). PLeTs-Test Automation using Software Product Lines and Model Based Testing. In *22nd International Conference on Software Engineering and Knowledge Engineering*, pages 483–488.

Griss, M., Favaro, J., and d'Alessandro, M. (1998). Integrating feature modeling with the RSEB. In *Proceedings. Fifth International Conference on Software Reuse*, pages 76–85.

Kang, K., Cohen, S., Hess, J., Novak, W., and Peterson, A. (1990). Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

Mendonça, M., Branco, M., and Cowan, D. D. (2009). S.P.L.O.T.: software product lines online tools. In Arora, S. and Leavens, G. T., editors, *OOPSLA Companion*, pages 761–762. ACM.

Pohl, K., Böckle, G., and van der Linden, F. J. (2005). *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, 1 edition.

Silveira, M. B., Rodrigues, E. M., Zorzo, A. F., Vieira, H., and Oliveira, F. (2011). Model-Based Automatic Generation of Performance Test Scripts. In *23rd International Conference on Software Engineering and Knowledge Engineering*, pages 1–6, Miami, FL, USA. Knowledge Systems Institute Graduate School.

Thüm, T., Kästner, C., Benduhn, F., Meinicke, J., Saake, G., and Leich, T. (2014). FeatureIDE: An Extensible Framework for Feature-oriented Software Development. *Science of Computer Programming*, 79:70–85.

van Gurp, J., Bosch, J., and Svahnberg, M. (2001). On the notion of variability in software product lines. In *Proceedings Working IEEE/IFIP Conference on Software Architecture*, pages 45–54.

Engenharia de Software Orientada a Agentes: Um Estudo Comparativo entre Metodologias que Suportam o Processo de Desenvolvimento de Sistemas Multiagente

Rafhael R. Cunha [1], Diana F. Adamatti[1], Cléo Z. Billa[1]

[1]Centro de Ciências Computacionais – Universidade Federal do Rio Grande (FURG)
Av. Itália km 8 – Bairro Carreiros – Rio Grande - RS - Brasil

{rcrafhaelrc,dianaada,cleo.billa}@gmail.com

Resumo. *Este artigo pertence ao domínio da Agent Oriented Software Engineering (AOSE) e Sistemas MultiAgente (SMA). O trabalho apresenta brevemente as metodologias Prometheus, Tropos e MaSE, elencando suas etapas e características de desenvolvimento. O objetivo deste trabalho é analisar as metodologias exploradas, através da comparação dos artefatos que delas resultam. Ao final deste trabalho, conclui-se que cada uma das metodologias analisadas possui enfoques distintos, atribuindo ao projetista a tarefa de escolher a que melhor se adeque as suas necessidades.*

1. Introdução

No decorrer dos anos, a área de engenharia de software (ES) tem procurado definir processos de desenvolvimento de software e linguagens de modelagem que visem estabelecer etapas bem definidas para a construção de um software. Este fato tem como propósito tornar a produção de um software mais robusta, rápida, organizada, confiável e de fácil manutenção e reutilização [Guedes 2012].

Na área de inteligência artificial, o paradigma orientado a agentes tem sido pesquisado e utilizado para minimizar a complexidade e aumentar a eficiência de softwares distribuídos [Jayatilleke et al. 2007] [Rodriguez et al. 2011]. Esta prática tem se mostrado eficiente para a construção de softwares com essas características, viabilizando um aumento no desenvolvimento de sistemas multiagente (SMA) [Guedes 2012].

Entretanto, devido à complexidade e distribuição desse tipo de sistema, novos desafios para a área de ES tradicional foram encontrados. Esses desafios ocorreram em virtude de linguagens conceituadas para se fazer modelagem na área, como *Unified Modelling Language* (UML), não suprirem as características presentes neste grupo de sistemas conforme [Cunha et al. 2015]. Essa ausência levou ao surgimento de uma subdivisão da área que mescla conceitos de engenharia de software e inteligência artificial, chamada de *Agent Oriented Software Engineering* (AOSE) ou Engenharia de Software Orientada a Agentes. Seus objetivos principais são propor métodos e linguagens/notações para projetar e modelar softwares orientados a agentes [Guedes 2012].

Dentro desse contexto, diversas metodologias foram propostas buscando suprir a demanda de softwares orientados a agentes [Bergenti et al. 2004]. Essas metodologias foram criadas pelos mais variados motivos. Algumas, basearam-se em melhorias nos diagramas presentes na UML, outras, criaram seus próprios meta-modelos e notações para seu uso.

Este trabalho tem como motivação apresentar uma revisão de algumas metodologias para construção de sistemas multiagente, mostrando as etapas presentes nessas metodologias e os artefatos utilizados para apoiar essas etapas. Por fim, como objetivo principal na realização deste trabalho, pretende-se fazer uma análise comparativa, especificando a similaridade/disparidade dos artefatos gerados pelas metodologias estudadas.

2. Metodologias AOSE

Segundo [Brandão 2014], as metodologias existentes até o presente momento para modelar SMA sobre a perspectiva da engenharia de software são as apresentadas na Figura 1. Nesta figura, os retângulos azuis indicam metodologias e processos OO; magentos com traços cheios indicam metodologias AO que estendem OO; magentas pontilhadas indicam metodologias AO que estendem outras AO; magentas tracejadas usam princípios OO, mas não estendem metodologias existente; violetas indicam que estende OO mas usam também técnicas de IA/ eng. de conhecimento; e Tropos que tem como base a analise de requisitos usando a*.

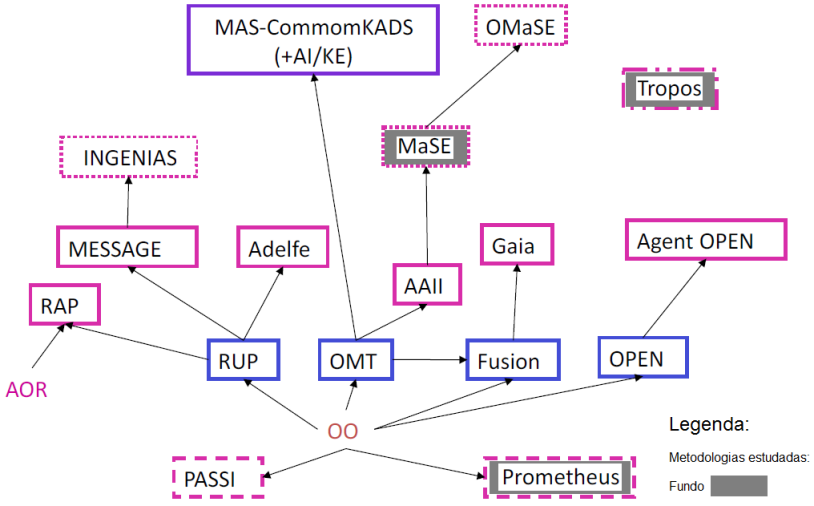


Figura 1. Metodologias AOSE [Brandão 2014]

A escolha das metodologias a serem estudadas neste trabalho, baseou-se na tentativa de explorar ao menos uma metodologia derivada de cada uma das metodologias clássicas, como RUP, OMT, Fusion e Open. Entretanto, Open foi excluída devido a inexistência de documentações que pudessem levar ao seu entendimento. Além disso, procurou-se selecionar uma metodologia que não apresentava nenhuma derivação, foi o caso de Prometheus; também buscou-se uma metodologia que não herdasse características do paradigma OO, como Tropos. Ingenias foi excluída deste trabalho em razão de sua extensividade, porém, uma extensão deste trabalho é encontrado em [Cunha et al. 2014].

2.1. Prometheus

Segundo [Padgham and Winikoff 2005] e [Khallouf and Winikoff 2009] Prometheus é uma metodologia que consiste em três fases: a fase de especificação do sistema; a fase de projeto arquitetural; e a fase de projeto detalhado. A primeira fase é composta por duas etapas: Determinar o ambiente do sistema (percepções e ações) e determinar os objetivos e funcionalidades do sistema (objetivos e cenários de casos de uso). Na fase de

projeto arquitetural são definidos quais agentes devem existir no sistema a ser modelado [Padgham and Winikoff 2002]. O objetivo final dessa etapa da metodologia é especificar completamente a interação entre os agentes. A última fase, projeto detalhado, foca no desenvolvimento da estrutura interna de cada um dos agentes e como os mesmos irão realizar suas tarefas dentro do sistema modelado.

2.2. Tropos

Segundo [Bresciani et al. 2004], a metodologia Tropos tem como propósito apoiar todas as atividades de análise e projeto do desenvolvimento de software orientado a agentes. Tropos é composto por cinco fases distintas no seu processo de desenvolvimento, sendo: Requisitos Iniciais; Requisitos Finais; Projeto Arquitetural; Projeto Detalhado e Implementação. Na primeira fase de análise dos requisitos, o usuário da metodologia identifica os agentes de domínio e modela-os como atores sociais, que dependem uns dos outros para atingirem os objetivos, planos e compartilharem dos mesmos recursos. Na segunda fase, o modelo conceitual é estendido de modo a incluir um novo ator, o qual representa o sistema, além de uma série de dependências com outros atores do ambiente. Nas próximas fases, projeto arquitetural e as fases de projeto detalhado, se concentram na especificação do sistema, em maneiras de garantir as exigências resultantes das fases anteriores. A etapa de implementação segue em passos, baseando-se na especificação do projeto detalhado com base no mapeamento estabelecido entre as construções da plataforma de execução e as noções do projeto detalhado [Bresciani et al. 2004].

2.3. MaSE

A metodologia *Multiagent Systems Engineering* (MaSE) é composta de duas fases principais. A primeira fase inclui três etapas: captura de objetivos, aplicação de casos de uso e refinamento de papéis. A segunda fase é composta por quatro etapas: criação de classes de agentes, construção das conversações, montagem das classes dos agentes e o projeto do sistema. A finalidade da primeira etapa da fase de análise é capturar os objetivos do sistema, extraindo-os da especificação inicial do sistema. Na etapa de aplicação de casos de uso, o analista transforma os objetivos e sub-objetivos em casos de uso. A terceira etapa tem como propósito garantir que foram identificados todos os papéis necessários no sistema e desenvolver as tarefas que definem o comportamento dos papéis e os padrões de comunicação. Na segunda fase, a primeira etapa identifica as classes de agentes a partir dos papéis refinados. A próxima fase é a construção de conversações. O objetivo desta etapa é definir os detalhes dessas conversas, baseados nos detalhes internos de tarefas simultâneas. Os elementos internos dos agentes são projetados durante a etapa de montagem das classes dos agentes, que inclui duas sub-etapas: a definição da arquitetura de agentes e a definição dos componentes da arquitetura. Projeto do sistema é o último passo da metodologia MaSE. Nessa etapa é utilizado o diagrama de implantação para demonstrar os números, tipos e locais das instâncias dos agentes no sistema.

3. Similaridades encontradas nos artefatos das metodologias exploradas

A comparação proposta neste trabalho consiste na verificação dos artefatos similares presentes nas metodologias exploradas. Para isso, nas subseções seguintes, foram explanados os detalhes que explicam essas similaridades. Nas tabelas ilustradas nessas subseções, o símbolo -, significa a não existência de artefatos que sejam semelhantes. O símbolo *, corresponde ao encontro de artefato semelhante, porém com restrições.

3.1. Similaridades dos artefatos da metodologia Prometheus

Tabela 1. Similaridade dos artefatos da metodologia Prometheus com as demais metodologias

Prometheus	Tropos	MaSE
Diag. de Cenários	-	-
Diag. Visão Geral Objetivos	Diag. de Objetivos	Diag. de Hierarquia de Objetivos
Diag. Papéis do Sistema	-	Diag. de Modelo de Papéis
Diag. Familiaridade de Agentes	Diag. de Ator	Diag. de Classe de Agentes
Formulário Descritor de Agentes	-	-
Diag. de Acoplamento de Dados	-	-
Diag. de Ligação de Papel com Agente	-	Diag. de Modelo de Papéis
Diag. de Visão geral do Sistema	-	-
Diag. de Interação	Diag. de Interação	Diag. de Classe de Comunicação
Diag. de Protocolos de Interação	Diag. de Interação (AUML)	Diag. de Classe de Comunicação
Diag. de Visão Geral de Agentes	-	-
Diag. Capacidades contendo Planos	-	-
Formulário Descritor de Planos	-	-

A visualização das similaridades presentes na metodologia Prometheus com as demais, é apresentada na tabela 1. Abaixo são listados os principais artefatos da metodologia Prometheus, explicando suas funções e comparando com os artefatos de outras metodologias.

- Diagrama de Cenários: Este diagrama tem a função de disponibilizar uma visão generalizada acerca do sistema [Padgham and Winikoff 2002]. Contudo, as demais metodologias não pensaram em um diagrama para demonstrar a seu usuário esse tipo de visão. Todas propõem diagramas que entram em detalhes mais específicos do SMA. Por esse motivo, não existe nenhum outro diagrama das metodologias estudadas que poderia se equiparar a esse.
- Digrama de Visão Geral de Objetivos: Segundo [Padgham and Winikoff 2002], os objetivos de um sistema são representados por meio de um diagrama de visão geral de objetivos.
  - Diagrama de Objetivos (Tropos): Conforme [Padgham and Thangarajah 2004], os objetivos representam os interesses estratégicos dos autores. Para modelar esses objetivos, na metodologia Tropos usa-se o diagrama de objetivos. Embora os conceitos possam parecer semelhantes, ocorre uma diferenciação entre a semântica da palavra objetivo nas duas metodologias. Para Prometheus, objetivos do sistema vão além dos interesses dos agentes, o que não ocorre em Tropos.

  - Diagrama de Hierarquia de Objetivos (MaSE): Em MaSE, existe uma fase específica na sua metodologia para a captura de objetivos. O conceito de objetivos dessa metodologia é idêntico ao apresentado na metodologia Prometheus, por esse motivo os dois diagramas apresentam similaridade em sua notação gráfica.
- Diagrama de Papéis do Sistema: Este diagrama serve para ligar as percepções, objetivos e ações aos papéis identificados.
  - Tropos: A metodologia Tropos não possui nenhum diagrama que expresse os conceitos envolvidos no diagrama de papéis da metodologia Prometheus, pois trabalha com alguns conceitos diferentes. Por exemplo, em Tropos os papéis do sistema estão implícitos na modelagem de atores, o mesmo não ocorre em Prometheus. Por esse razão, não existe um diagrama para expressar fielmente tal representação.
  - Diagrama de Modelo de Papéis (MaSE): Na metodologia MaSE, o diagrama de modelo de papéis, tem finalidade semelhante, com exceção da falta de expressividade em relação as percepções. Entretanto, MaSE acrescenta em seu diagrama os protocolos de interação entre os papéis, o que não é visto no diagrama de papéis da metodologia Prometheus. Para este fim, Prometheus possui o diagrama de protocolos de interação.
- Diagrama de Familiaridade de Agentes: O objetivo do diagrama de familiaridade de agentes presente na metodologia Prometheus é ligar um agente com os agentes que ele possui algum tipo de interação [Padgham and Winikoff 2002].
  - Diagrama de Ator (Tropos): Na metodologia Tropos, o diagrama de atores faz esse papel com excelência, e ainda acrescenta outros conceitos como: objetivos, subobjetivos, entre outros.
  - Diagrama de Classes de Agentes (MaSE): A metodologia MaSE dispõe do diagrama de classes de agentes para representar essa interação entre diferentes tipos de agentes no sistema. Ela modela as interações em um nível hierárquico mais alto, ou seja, entre os papéis. A metodologia Prometheus modela sobre os agentes, não importando-se com os papéis. Na verdade a metodologia Prometheus utiliza desse diagrama justamente para identificar quais agentes do sistema podem ser agrupados e assim identificar os papéis.
- Formulário Descritor de Agentes: Nenhuma outra metodologia propõe algum método para descrever informações de alto nível acerca dos agentes utilizando linguagem natural.
- Diagrama de Acoplamento de Dados: Nenhuma outra metodologia propõe algum método para representar a necessidade de dados que devem ser guardados no sistema.
- Diagrama de Ligação de Papel com Agente: Este diagrama tem a finalidade de fazer a ligação dos agentes com os papéis que eles desempenharão [Padgham and Winikoff 2002].
  - Tropos: Esta metodologia não lida com o conceito de papéis, pois os papéis estão implícitos na modelagem dos atores.
  - Diagrama de Modelo de Papéis (MaSE): MaSE possui um diagrama chamado de modelo de papéis que tem como propósito garantir que foram

identificados todos os papéis necessários no sistema e desenvolver as tarefas que definem o comportamento dos papéis e os padrões de comunicação [Henderson-Sellers and Giorgini 2005]. Observa-se que esse diagrama além de suprir a informação presente no diagrama de ligação de papel com agente da metodologia Prometheus, acrescenta também informações dos protocolos de comunicação desses papéis e as funcionalidades que devem ser executadas por esses papéis.

- Diagrama de Visão Geral do Sistema: Conforme [Padgham and Winikoff 2002], o diagrama de visão geral do sistema é o artefato crucial na fase de processo arquitetural do Prometheus. Este diagrama enlaça agentes, percepções, dados, ações, mensagens. Pode-se dizer que ele é uma mesclagem de todos os diagramas desenvolvidos anteriormente. Em virtude disso, nenhuma outra metodologia é capaz de agrupar essa quantidade de informações em um único artefato. Contudo, não significa que as metodologias não tem esse poder de expressividade, e sim que elas necessitam de mais diagramas para pode chegar a esse nível de detalhamento.
- Diagrama de Interação: Conforme [Padgham and Winikoff 2002], o diagrama de interação serve para especificar completamente a interação com os agentes.
  - Diagrama de Interação (Tropos): Esta metodologia dispõe um diagrama de interação semelhante ao encontrado na metodologia Prometheus. Ambos tem o mesmo propósito. Portanto, é possível utilizar o presente na metodologia Tropos sem perda de sentido.
  - Diagrama de Classe de Comunicação (MaSE): A metodologia MaSE contém um diagrama chamado de diagrama de classes de comunicação que também especifica as interações entre os agentes. Entretanto, esse diagrama é um pouco mais refinado, pois nele é possível definir um protocolo de coordenação entre dois agentes [Henderson-Sellers and Giorgini 2005].
- Diagrama de Protocolo de Interação: Segundo [Padgham and Winikoff 2002], o objetivo do diagrama de protocolo de interação é definir precisamente quais sequências de interações são válidas dentro do sistema.
  - Diagrama de Interação (Tropos): A metodologia Tropos utiliza do diagrama de interação da *Agent Unified Modeling Language* (AUML) [1]. Através desse diagrama é possível satisfazer essas condições e expressar de forma exata as mesmas informações presentes no diagrama de protocolo de interação da metodologia Prometheus.
  - Diagrama de Classe de Comunicação (MaSE): MaSE possui um diagrama chamado de diagrama de classe de comunicação que tem como finalidade definir os detalhes internos das conversas entre os agentes. Portanto, através desse diagrama, é possível transcrever o diagrama de Protocolo de Interação, sem perda de informações.
- Diagrama de Visão Geral de Agente: O diagrama de Visão Geral de Agente presente na metodologia Prometheus, demonstra capacidades, ações, mensagens, dados e percepções acerca dos agentes. Por ser um diagrama que contém um número considerável de informações, não existe nenhum outro artefato único nas demais metodologias que consigam expressar tantas informações juntas. Por esse razão,

[1] AUML é uma adaptação da UML que buscou suprir as características do âmbito de agentes .

esse diagrama só é possível ser transpassado através da junção de dois ou três diagramas das outras metodologias.

- Diagrama de Capacidade contendo Planos: Este diagrama tem o poder de expressar as percepções, dados, ações, mensagens e planos que compõem os agentes. As outras metodologias não tem o poder de representar em um único diagrama todas as informações que o diagrama de capacidade contendo planos possui.
- Formulário Descritor de Planos: Conforme acontece com o outro formulário de descritor de agentes, nenhuma outra metodologia tem em seu processo um artefato que possibilite que seu usuário descreva em linguagem natural um formulário com a descrição dos planos do SMA.

3.2. Similaridades dos artefatos da metodologia Tropos

A visualização das similaridades presentes na metodologia Tropos com as demais é apresentada na tabela 2. Abaixo são listados os principais artefatos da metodologia Tropos, explicando suas funções e comparando com os artefatos de outras metodologias.

Tabela 2. Similaridade dos artefatos da metodologia Tropos com as demais metodologias

Tropos	Prometheus	MaSE
Diag. de Ator	Diag. Geral de Objetivos*	Diag. de Hierarquia de Objetivos*
Diag. de Objetivos	-	-
Diag. de Objetivos para Modelar Plano	-	-
Diag. de Capacidade	Diag. de Visão Geral de Agente	-
Diag. de Interação	Diag. de Interação	Diag. de Classe de Comunicação

- Diagrama de Ator: Este diagrama é utilizado para modelar os atores, seus objetivos concretos e nebulosos.
  - Diagrama Geral de Objetivos (Prometheus): A metodologia Prometheus pensa em objetivos de maneira diferente a apresentada na metodologia Tropos. Para Tropos, os objetivos e subobjetivos são ligados diretamente aos agentes. Em Prometheus, esses objetivos fazem parte do sistema, onde são definidos papéis, que posteriormente são assumidos por agentes, para que futuramente esses objetivos sejam satisfeitos pelos agentes. Um diagrama parecido na metodologia Prometheus é o diagrama geral de objetivos. Entretanto, neste diagrama é somente possível modelar os objetivos e subobjetivos. Não existe nenhuma relação com os agentes e não é possível expressar as dependências entre os objetos da mesma forma que é contemplado no diagrama de Ator da metodologia Tropos.
  - Diagrama de Hierarquia de Objetivos (MaSE): Na metodologia MaSE ocorre o mesmo problema exemplificado em Prometheus. Contudo, MaSE também apresenta um diagrama para modelar objetivos, intitulado diagrama de hierarquia de objetivos. Porém, também não é possível transcrever o diagrama de ator de Tropos para esse diagrama sem perda de conteúdo.

- Diagrama de Objetivos: Este diagrama tem por finalidade modelar objetivos, planos, e a ligação de composição/decomposição além da contribuição entre planos e objetivos.
  - Prometheus: A metodologia Prometheus trabalha com o conceito de objetivos associado ao SMA, ao invés dos agentes. Além disso, o diagrama de objetivos da metodologia Tropos modela alguns outros conceitos. Portanto, não existe nenhum diagrama da metodologia Prometheus que consiga suprir essa demanda de informações.
  - MaSE: Esta metodologia é semelhante a do Prometheus em referência ao conceito de Objetivos. Portanto, também não existe um diagrama para expressar tais informações na metodologia MaSE.
- Diagrama de Objetivos para Modelar Planos: Funciona de forma semelhante ao diagrama de objetivos, pois a notação gráfica desse diagrama é semelhante, mudando apenas o conteúdo do que se deseja expressar.
- Diagrama de Capacidade: Este diagrama é utilizado para modelar as capacidades dos agentes, percepções, dados, ações e mensagens [Bresciani et al. 2004].
  - Diagrama de Visão Geral de Agentes (Prometheus): Esta metodologia disponibiliza de um diagrama chamando de Diagrama de Visão Geral de Agentes. Através desse diagrama, é possível modelar o diagrama de capacidades utilizado na metodologia Tropos sem perda de expressividade.
  - MaSE: Esta metodologia não suporta o conceito de capacidade estipulada na metodologia Tropos. Por essa razão, não possui diagramas para modelar essa informação.
- Diagrama de Interação: Este diagrama serve para modelar as interações que ocorrem entre os agentes que compõem o sistema.
  - Diagrama de Interação (Prometheus): A metodologia Prometheus possui um diagrama de interação semelhante ao encontrado na metodologia Tropos. Pode-se transcrever as informações de um para o outro sem perda de informação.
  - Diagrama de Classe de Comunicação (MaSE): Esta metodologia dispõe do diagrama de classe de comunicação que consegue desempenhar o papel do digrama de interação sem devaneios. Contudo, no diagrama da metodologia MaSE ainda é possível especificar um protocolo de comunicação entre agentes.

3.3. Similaridades dos artefatos da metodologia MaSE

A visualização das similaridades presentes na metodologia MaSE com as demais é apresentada na tabela 3. Abaixo são listados os principais artefatos da metodologia MaSE, explicando suas funções e comparando com os artefatos de outras metodologias.

- Diagrama de Hierarquia de Objetivos: Segundo [Henderson-Sellers and Giorgini 2005], a captura de objetivos é a primeira finalidade da etapa da análise de requisitos. Depois dessa captura, os objetivos são organizados conforme sua importância.
  - Diagrama de Visão Geral de Objetivos (Prometheus): A metodologia Prometheus possui o diagrama de visão geral de objetivos, que possui o mesmo grau de representatividade. Portanto, é possível transcrever um no outro sem perda de informações.

Tabela 3. Similaridade dos artefatos da metodologia MaSE com as demais metodologias

MaSE	Prometheus	Tropos
Diag. de Hierarquia de Objetivos	Diag. de Visão Geral de Objetivos	Diag. de Objetivos
Diag. de Sequência	Diag. de Familiaridade de Agentes	Diag. de Interação (AUML)
Diag. de Modelo de Papéis	Diag. de Ligação de Papel com Agente*	-
Diag. de Tarefas Concorrentes	Diag. de Protocolo de Interação	-
Diag. de Classes de Agentes	Diag. de Cenários*	-
Diag. de Classe de Comunicação	Diag. de Protocolo de Interação	-
Diag. de Arquitetura de Agentes	-	-
Diag. de Implantação	-	-

  - Diagrama de Objetivos (Tropos): Existe uma diferença no significado da palavra objetivo perante as duas metodologias. Em Tropos, os objetivos são modelados de acordo com os atores. Por esse motivo, é possível transcrever o diagrama de Hierarquia de Objetivos de Mase para Tropos. Porém, o processo inverso não é viável.

- Diagrama de Sequência: O objetivo do diagrama de sequência na metodologia MaSE é representar sequências de eventos entre os papéis.
  - Diagrama de Familiaridade de Agentes (Prometheus): Na metodologia Prometheus, o diagrama de Familiaridade de Agentes desempenha papel semelhante ao diagrama de sequência. Neste diagrama, é possível modelar a troca de eventos entre os agentes, ou seja, em um nível hierárquico mais baixo se comparado ao diagrama de sequências de MaSE, que modela a troca de eventos entre papéis. Se forem especificados todos os agentes que compõem os papéis do sistema, é possível transcrever o diagrama de sequências de MaSE no diagrama de familiaridade de agentes de Prometheus sem perda de expressividade.
  - Diagrama de Interação (Tropos): A metodologia Tropos utiliza do diagrama de interação da AUML para especificar a troca de eventos entre agentes/papéis de um SMA. Por esse motivo, é possível utilizá-lo para representar os mesmos conceitos envolvidos no diagrama de sequência da metodologia MaSE.
- Diagrama de Modelo de Papéis: Este diagrama tem como propósito representar papéis, tarefas e protocolos de interações [Henderson-Sellers and Giorgini 2005].
  - Diagrama de Ligação de Papel de Agente (Prometheus): O diagrama de ligação de papel de agente da metodologia Prometheus possui propósito semelhante ao diagrama de modelo de papéis. Entretanto, neste diagrama não é possível representar as tarefas e nem os protocolos de interação. A transcrição das informações do diagrama de modelo de papéis de MaSE

para o diagrama de ligação de papéis de Prometheus acarretaria na perda de informações.
  - Tropos: Não trabalha com o conceito de papéis. Para a metodologia, os papéis estão implícitos nos agentes. Por esse motivo, não existem diagramas para essa representação na metodologia Tropos.

- Diagrama de Tarefas Concorrentes: O propósito deste diagrama é definir protocolos de interação complexos e de alto nível que necessitem de coordenação entre agentes múltiplos [Henderson-Sellers and Giorgini 2005].
  - Diagrama de Protocolo de Interação (Prometheus): A metodologia Prometheus detêm do diagrama de protocolo de interação que tem características semelhantes ao diagrama de tarefas concorrentes. Entretanto, neste diagrama o foco é validar essas interações. Contudo, em linhas gerais, ambos tem o mesmo objetivo, que é demonstrar e validar as interações entre os agentes.
  - Tropos: A metodologia Tropos, embora trabalhe com o conceito de interações entre os agentes, não utiliza uma abstração maior, como protocolos. Por esse motivo, não existe um diagrama que seja possível modelar tal finalidade nesta metodologia.

- Diagrama de Classes de Agentes: O objetivo deste diagrama é retratar a organização geral dos agentes no sistema [Henderson-Sellers and Giorgini 2005]. Além disso, as ligações entre essas classes representam as conversações que devem existir entre elas.
  - Diagrama de Cenários (Prometheus): Prometheus, possui um diagrama que tem finalidade semelhante ao diagrama de classe de agentes, chamado de diagrama de cenários. Entretanto, o diagrama de cenários tem como propósito oferecer uma visão geral do sistema sob o ponto de vista de cenários, e não agentes, como é o caso de diagrama de classes de Agentes. Além disso, no diagrama de cenários da metodologia Prometheus não é possível estipular conversas entre os cenários. Portanto, embora que com o mesmo propósito, ambos os diagramas apresentam visões diferentes do sistema.
  - Tropos: A metodologia Tropos não dispõe de nenhum diagrama que consiga demonstrar as conversações entre as classes de agentes e uma visão generalizada das classes que representam os agentes.

- Diagrama de Classe de Comunicação: O objetivo do diagrama de classe de comunicação é expressar o protocolo de conversa entre os agentes [Henderson-Sellers and Giorgini 2005].
  - Diagrama de Protocolo de Interação: Em Prometheus, existe o diagrama de protocolo de interação que tem finalidade semelhante ao diagrama ed classe de comunicação. Ele pode expressar as mesmas informações sem perda de sentido.
  - Tropos: Similar as demais explicações sobre protocolos em Tropos, sabe-se que esta metodologia não suporta esse conceito. Por esse motivo, não contém diagramas para tal finalidade.

- Diagrama de Arquitetura de Agentes: Este diagrama tem como finalidade demonstrar a interação entre os componentes da arquitetura de agentes [Henderson-Sellers and Giorgini 2005]. Além disso, através desse diagrama é possível demonstrar a visibilidade entre os componentes e a conexões externas destes para recursos, como agentes, sensores, processadores de efeitos e banco de dados.
  - Prometheus: A metodologia Prometheus não trabalha com o conceito da arquitetura de agentes. Para Prometheus, os agentes são baseados na arquitetura *Believe-Desire-Intention* (BDI) [Rao and Georgeff 1991]. Outro ponto importante, consiste na falta de modelagem do Prometheus em relação aos componentes externos que o SMA pode utilizar. Por esse motivo, não existem diagramas na metodologia Prometheus para representar as informações contidas no diagrama de Arquitetura de Agentes.
  - Tropos: A metodologia Tropos também não trabalha com o conceito de arquitetura de agentes e nem com a possibilidade de representar os componentes que podem interagir com o ambiente. Portanto, não existe diagrama para tal finalidade nesta metodologia.
- Diagrama de Implantação: O diagrama de implantação da metodologia MaSE demonstra os números, tipos e locais das instâncias dos agentes no sistema [Henderson-Sellers and Giorgini 2005]. Nenhuma metodologia das exploradas disponibiliza uma forma de representação semelhante a encontrada no diagrama de implantação da metodologia MaSE. Algumas, chegam no ponto de especificar as interações dos agentes com o ambiente. Entretanto, não apresentam o maior nível de todos, que é demonstrar onde esses agentes estão instalados no sistema.

4. Conclusões e trabalhos futuros

Este trabalho teve como objetivo apresentar uma revisão de algumas metodologias para modelagem de sistemas multiagente. Também foi realizado um estudo comparativo entre as metodologias estudadas (Prometheus, Tropos e MaSE) de forma a apresentar similaridades e diferenças entre as metodologias.

Com base esse estudo, percebe-se que cada metodologia uma possui enfoque em características distintas. Enquanto Prometheus e Tropos focam sua modelagem no âmbito de agentes, MaSE possui conceitos que propiciam também a modelagem de recursos e arquiteturas alternativas de agentes. Percebe-se também que nenhuma metodologia explorada é capaz de suprir os conceitos envolvidos no paradigma multiagente, apresentando disparidade de objetivos, fato que enfraquece a normalização da área.

Desta forma, conclui-se que a área de AOSE deve progredir para atingir uma padronização. Sugere-se que seja feito um mapeamento dos requisitos necessários para desenvolver qualquer tipo de SMA. Posteriormente, deve-se desenvolver notações gráficas que corresponderão a conceitos relacionados a todas as características mapeadas anteriormente. Através dessas ações, consegue-se definir uma linguagem de modelagem, semelhante a UML, porém, que atenda a requisitos provenientes dos SMA.

5. Agradecimentos

Os autores agradecem à Universidade Federal do Rio Grande - FURG e a Fundação de Amparo à Pesquisa do Estado do Rio Grande do Sul - FAPERGS pelo suporte financeiro

na realização do presente trabalho.

Referências

Bergenti, F., Gleizes, M.-P., and Zambonelli, F. (2004). *Methodologies and software engineering for agent systems: the agent-oriented software engineering handbook*, volume 11. Springer.

Brandão, A. A. F. (2014). Apresentação de oficina no wesaac 2014 - engenharia de software orientada a agente. Enviado por e-mail pela autora (anarosabrandao@gmail.com), em 17 julho 2014.

Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., and Mylopoulos, J. (2004). Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236.

Cunha, R. R., Adamatti, F. D., and Billa, Z. C. (2014). Engenharia de software orientada a agentes: Um estudo sobre metodologias que suportam o processo de desenvolvimento de sistemas multiagente. Universidade Federal do Rio Grande - Trabalho Individual apresentado ao programa de Pós Graduação em Engenharia de Computação (PPGCOMP).

Cunha, R. R., Adamatti, F. D., and Billa, Z. C. (2015). Engenharia de software orientada a agentes: Um estudo comparativo entre uml e metodologias que suportam o processo de desenvolvimento de sistemas multiagente. Workshop-Escola de Sistemas de Agentes, seus Ambientes e apliCações, to appear.

Guedes, G. T. A. (2012). *Um metamodelo UML para a modelagem de requisitos em projetos de sistemas multiagentes*. PhD thesis, Universidade Federal do Rio Grande do Sul.

Henderson-Sellers, B. and Giorgini, P. (2005). *Agent-oriented methodologies*. IGI Global.

Jayatilleke, G. B., Padgham, L., and Winikoff, M. (2007). Evaluating a model driven development toolkit for domain experts to modify agent based systems. In *Agent-Oriented Software Engineering VII*, pages 190–207. Springer.

Khallouf, J. and Winikoff, M. (2009). The goal-oriented design of agent systems: a refinement of prometheus and its evaluation. *International Journal of Agent-Oriented Software Engineering*, 3(1):88–112.

Padgham, L. and Thangarajah, J. (2004). Tutorial prometheus. http://www.cs.rmit.edu.au/agents/pdt/docs/Tutorial.pdf.

Padgham, L. and Winikoff, M. (2002). Prometheus: A methodology for developing intelligent agents. *John Wiley & Sons*.

Padgham, L. and Winikoff, M. (2005). *Developing intelligent agent systems: A practical guide*, volume 13. John Wiley & Sons.

Rao, A. S. and Georgeff, M. P. (1991). Modeling rational agents within a bdi-architecture. *KR*, 91:473–484.

Rodriguez, L., Insfran, E., and Cernuzzi, L. (2011). *Requirements Modeling for Multi-Agent Systems*. INTECH Open Access Publisher.

Avaliação Experimental da Relação de Coesão e Acoplamento com o Esforço de Compreensão de Software

Elienai B. Batista[1], Claudio Sant'Anna[1]

[1]Departamento de Ciência da Computação – Universidade Federal da Bahia (UFBA) – Salvador – BA - Brasil

elienaibittencourt@gmail.com, santanna@dcc.ufba.br

Abstract. *Coupling and cohesion are well-known software design quality attributes. They are believed to directly influence the comprehensibility of a system. It is claimed that the greater the cohesion and the lower the coupling of a module the easier it is to understand it. However very few studies have been conducted to analyze the relation of values of cohesion and coupling, quantified by means of metrics, with the effort for comprehending source code. In this context, we propose two quasi-experiments that aim to empirically evaluate in which extent the degrees of cohesion and coupling of software modules are related to the effort for understanding their source code.*

Resumo. *Acoplamento e coesão são atributos de qualidade do design de software bem conhecidos. Acredita-se que eles podem influenciar diretamente a compreensibilidade de um sistema. Afirma-se que, quanto maior a coesão e menor o acoplamento de um módulo, mais fácil é sua compreensão. Entretanto, poucos estudos foram realizados para analisar qual é a relação entre os valores de coesão e acoplamento, quantificados por meio de métricas, com o esforço de compreensão do código fonte. Diante disso, propomos a realização de dois quase-experimentos para de avaliar empiricamente em que nível o grau de coesão e de acoplamento de módulos de software estão relacionados ao esforço para compreender seus códigos fontes.*

1. Introdução

Acoplamento e coesão são dois conceitos bem conhecidos e usados como atributos de qualidade do design de software [Pflegger & Atlee, 2010]. A coesão de um módulo é o grau pelo qual aquele módulo está focado em implementar apenas uma responsabilidade do sistema. Acoplamento é uma indicação da quantidade e força das dependências entre os módulos que compõem um sistema [Pflegger & Atlee, 2010]. Afirma-se que um software bem projetado é aquele cujos módulos são bem coesos e fracamente acoplados entre si.

Uma série de métricas tem sido propostas para quantificar coesão e acoplamento de sistemas orientados a objetos [Chidamber & Kemerer 1994], [Henderson- Sellers et al., 1996], [Briand et al., 1998], [Briand et al., 1999], [Silva et al., 2011]. A maioria dessas métricas quantifica coesão e acoplamento de cada uma das classes que compõem o código fonte do sistema. Existem métricas de dois tipos: estruturais e conceituais. As métricas estruturais consideram características sintáticas do código e avaliam as relações de dependência estrutural: (i) entre os métodos de uma classe, para quantificar

coesão, e (ii) entre as classes do sistema, para quantificar acoplamento [Briand et al., 1998], [Briand et al., 1999]. Métricas conceituais consideram aspectos semânticos do código fonte para calcular coesão e acoplamento. A maioria delas se baseia em técnicas de mineração de textos para determinar elementos do código fonte semanticamente relacionados entre si [Marcus & Poshyvanyk, 2005], [Silva et al., 2012].

Acredita-se que, quanto maior for coesão e menor for o acoplamento dos módulos do sistema, menor é o esforço para se compreender o sistema [Henderson-Sellers et al., 1996], [Briand et al., 1999]. Módulos com baixa coesão são, teoricamente, mais difíceis de compreender, pois possuem código relativo a diferentes responsabilidades, o que pode atrapalhar o entendimento de cada uma delas. Módulos com alto acoplamento também são, teoricamente, mais difíceis de compreender, pois para entendê-los pode ser necessário entender os módulos dos quais ele depende.

Compreensão de software consiste da realização de atividades para se obter conhecimento geral sobre um software, sobre suas funcionalidades, sua organização e seu propósito [Bois et al., 2006]. Estima-se que desenvolvedores dediquem em média mais da metade do esforço de manutenção de software com atividades de compreensão. Além disso, parte substancial do esforço de compreensão de software está relacionada à compreensão do código fonte. Apesar da importância do processo de compreensão e da grande quantidade de métricas de acoplamento e coesão existentes, poucos estudos foram realizados para avaliar empiricamente qual é a relação entre coesão e acoplamento, quantificados por meio de métricas, e o esforço para se compreender o código fonte de sistemas de software.

Os trabalhos relacionados a esse tema ou estudam como outras características do código fonte estão relacionadas à compreensão dos sistemas, ou estudam a relação de coesão e/ou acoplamento com outros atributos externos, como propensão a defeitos ou propensão a mudanças. Por exemplo, Bois et al. (2006) estudam se refatorações para eliminar um anomalia específica de design melhoram a compreensibilidade do código fonte. Silva et al. (2012), por sua vez, estudam a relação entre medições de coesão e acoplamento com a propensão que classes tem de sofrer mudanças. Não conhecemos trabalhos que avaliem como medições de coesão ou acoplamento, de forma isolada, estão relacionadas ao esforço para se compreender o código fonte de sistemas.

2. Proposta

Diante desse contexto, o trabalho aqui proposto tem o objetivo de avaliar empiricamente em que nível o grau de coesão e de acoplamento de módulos de sistemas de software estão relacionados ao esforço para compreender seu código fonte. Pretendemos também avaliar se os diferentes tipos de métricas – estrutural e conceitual – tem relação diferente com o esforço de compreensão. Para atingir esses objetivos, pretendemos realizar dois quase-experimentos: (i) um no qual participantes executem tarefas de compreensão do código fonte de classes com diferentes graus de coesão conceitual e estrutural, quantificados por meio de métricas de código fonte e (ii) outro no qual os participantes executem tarefas de compreensão do código fonte de classes com diferentes graus de acoplamento, quantificados por meio de métricas de código fonte.

O primeiro experimento visa responder as seguintes questões de pesquisa: (i) A coesão de uma classe tem influência sobre o esforço para compreender seu código fonte? (ii) Existe diferença entre a relação de coesão estrutural e coesão conceitual com

o esforço de compreensão do código fonte? O segundo estudo visa responder a seguinte questão de pesquisa: O acoplamento de uma classe tem influência sobre o esforço para compreender seu código fonte?

De forma resumida, os experimentos serão configurados da seguinte forma: No laboratório, participantes realizarão atividades que demandarão a compreensão do código fonte de um conjunto de classes. A realização dessas atividades permitirá que seja medido o esforço desprendido por cada participante para compreender cada classe. As classes devem apresentar diferentes valores de coesão e acoplamento para que possamos compará-los com o esforço de compreensão.

Medição do esforço de compreensão. Para quantificarmos o esforço necessário para compreender o código fonte de cada classe, usaremos duas métricas: (i) o tempo necessário para o participante responder um conjunto de perguntas sobre cada uma das classes, e (ii) o número de perguntas com respostas erradas. As perguntas são do tipo que exigem a simulação mental do código fonte, tais como "Dado que determinado método recebe um conjunto específico de argumentos, qual será seu retorno?" Essa estratégia está alinhada com o que se é recomendado na literatura para medir esforço de compreensão de código fonte [Bois et al., 2006].

Métricas: Pretendemos usar a métrica de coesão conceitual *Lack of Concern-Based Cohesion* (LCbC) [Silva et al., 2008], pois ela já vem sendo usada em outros estudos de comparação com métricas estruturais [Silva et al., 2012], [Silva et al., 2014]. Pretendemos usar a métrica de coesão estrutural *Lack of Cohesion on Methods* 5 (LCOM5) [Henderson-Sellers et al., 1996]. Essa métrica representa a versão mais atual da tradicional métrica de coesão LCOM, de Chidamber e Kemerer (1994). Além disso, há ferramentas disponíveis que automatizam sua aplicação. Para quantificar acoplamento, planejamos usar a métrica Coupling Between Object Classes (CBO) [Chidamber & Kemerer, 1994]. CBO é uma das métricas de acoplamento mais referenciadas na literatura e tem sua aplicação automatizada por algumas ferramentas.

Seleção de Classes: Esse é um ponto crítico e não trivial da preparação dos estudos. Ao selecionar as classes temos que procurar minimizar ao máximo a influência de fatores de confusão. Nosso plano é, portanto, selecionar classes: (i) de tamanhos semelhantes em termo de número de linha de código, (ii) de domínios simples e acessíveis a todos os participantes, (iii) com nomes de identificadores com qualidade semelhantes e (iv) sem comentários. Além disso, no estudo sobre coesão, as classes devem apresentar valores similares de acoplamento e vice-versa. Selecionaremos classes de diferentes sistemas de informação *open source* escritos em Java.

Participantes: Os participantes dos estudos serão alunos de graduação e de pós-graduação em Ciências da Computação da Universidade Federal da Bahia (UFBA). Os participantes assinarão um termo de consentimento e responderão a um questionário para podermos caracterizar sua experiência em programação orientada a objetos.

Ameaças à validade: Além dos fatores de confusão relacionados à similaridade do código fonte das classes, pretendemos realizar estudo piloto para controlar outras ameaças à validade, tais como, a possível fadiga dos participantes e a falta de clareza das perguntas sobre as classes. O questionário sobre o perfil dos participantes nos dará informações para controlar possíveis discrepâncias de nível de experiência.

3. Andamento do Trabalho

Até o momento, já realizamos um estudo preliminar para avaliar a relação entre coesão (estrutural e conceitual) e compreensão de código fonte. O estudo teve como participantes 14 alunos de graduação e quatro alunos de pós-graduação. Cada participante realizou atividades de compreensão em quatro classes. Essas atividades consistiram do participante responder quatro questões sobre cada uma das classes.

Esse estudo se caracterizou como preliminar por envolver poucos participantes e poucas classes. Além disso, inadvertidamente, deixamos de controlar um fator de confusão importante: o grau de acoplamento das classes. Como resultado, uma classe com grau de acoplamento bem mais alto que as outras demandou muito mais esforço de compreensão. Esse resultado nos motivou a estudar também a relação entre acoplamento e compreensão. Um dos pontos positivos do estudo preliminar foi o fato da estratégia usada para medir o esforço de compreensão ter funcionado bem.

A partir da experiência com o estudo preliminar, pretendemos realizar os dois experimentos propostos com um número maior de participantes e classes. Também seremos mais rigorosos no controle dos fatores de confusão. Atualmente, estamos trabalhando no planejamento do novo estudo sobre coesão: as classes estão selecionadas e questionários construídos. Sua execução se dará ainda no primeiro semestre desse ano.

Referências

Bois, B. D., Demeyer, S., Verelst, J., Ments, T. and Temmerman, M (2006). "Does God Class Decomposition Affect Comprehensibility?" *IASTED Conf. on Software Engineering*. Innsbruck, pp. 346-355.

Briand, L. C., Daly, J. W. and Wust, J. K. (1998). A Unified Framework for Cohesion Measurement in Object-Oriented Systems, *Empirical Software Engineering - An International Journal*, 3(1), pp. 65-117.

Briand, L. C., Daly, J. W. and Wust, J. K. (1999). "A Unified Framework for Coupling Measurement in Object-Oriented Systems". *IEEE Transactions on Software Engineering*. 25(1), pp. 91-12.

Chidamber, S.; Kemerer, C. (1994) "A Metric Suite for Object Oriented Design". IEEE Transactions on Software Engineering, 20(6), pp. 476-493.

Henderson-Sellers, B., Constantine, L. and Graham, M. (1996) "Coupling and Cohesion (Towards a valid metrics suite for object-oriented analysis and design)". *Object Oriented Systems*, vol 3, pp. 143-158.

Marcus, A. & Poshyvanyk, D. (2005) "The Conceptual Cohesion of Classes". Intl' Conference on Software Maintenance (ICSM '05). Washington, DC, pp. 133-142.

Pfleeger, S.; Atlee, J. (2010) "Software Engineering: theory and practice", 4th ed, Prentice Hall.

Silva, B., Sant'Anna, C., Chavez, C. and Garcia, A. (2012) "Concern-Based Cohesion: Unveiling a Hidden Dimension of Cohesion Measurement". *IEEE International Conference on Program Comprehension (ICPC 2012)*, Passau, pp. 103-112.

Silva, B., Sant'Anna, C., Chavez, C. "An Empirical Study on How Developers Reason about Module Cohesion." Int'l Conference on Modularity (Modularity 2014), Lugano, pp. 121-132.

Estabelecimento de uma Arquitetura de Referência para Ferramentas de Gerenciamento de Variabilidades

Ana Paula Allian[1], Edson OliveiraJr[1], Elisa Y. Nakagawa[2]

[1]Universidade Estadual de Maringá (UEM) – Maringá, PR – Brazil

[2]Universidade de São Paulo (USP) – São Carlos, SP – Brazil

ana.allian@gmail.com, edson@din.uem.br, elisa@icmc.usp.br

Abstract. *Variability management (VM) is one of the core activities for the success of software reuse. Several VM tools have been proposed toward supporting such an activity. In another context, reference architectures (RA), a special type of software architecture, can contribute to the reuse of knowledge about VM and standardization of tools for this pourpose. This paper presents an ongoing research on VMTools-RA, a RA for VM tools. We intend this RA can contribute to the development of new VM tools, besides providing support to integration and reuse of projects experiences.*

Resumo. *Gerenciamento de variabilidades (GV) é uma das atividades essenciais para o sucesso do reúso de software. Várias ferramentas de GV têm sido propostas para apoiar tal atividade. Em outro contexto, arquiteturas de referência (AR), um tipo especial de arquitetura de software, podem contribuir para o reúso de conhecimento sobre GV e padronização de ferramentas para tal proposito. Este artigo apresenta a pesquisa em andamento sobre a VMTools-RA, uma AR para ferramentas de GV. Espera-se que essa AR possa contribuir para o desenvolvimento de novas ferramentas de GV, além de fornecer suporte à integração e reuso de experiências em projetos.*

1. Introdução

Variabilidade é o termo utilizado para representar como produtos de software podem se diferenciar entre si. A variabilidade pode ser identificada mediante o conceito de *feature*, uma característica de um sistema que é relevante e visível para o usuário final. O conceito de variabilidade se consolidou em diferentes abordagens, sobretudo em linha de produto de software [Linden et al. 2007], tornando a atividade de Gerenciamento de Variabilidades (GV) essencial para o sucesso dessa estratégia de reúso de software sistemático.

Atualmente, existem diversas ferramentas de GV. Por causa da heterogeneidade de ferramentas, a indústria tem utilizado diferentes tipos de soluções para gerenciar a variabilidade, além de criar suas próprias ferramentas [Berger et al. 2013]. Além disso, essas ferramentas não seguem um padrão pré-definido para seu desenvolvimento. Portanto, entende-se como uma oportunidade de pesquisa responder a seguinte questão:***É possível propor uma padronização em nível arquitetural para ferramentas de GV?*** Para responder a tal questão, pode-se explorar os conceitos relacionados às arquiteturas de referência.

Uma Arquitetura de Referência (AR) é considerada um padrão pré-definido projetado para um determinado contexto de negócio. Tal conceito faz uso de ativos (*assets*),

muitas vezes, concebidos em projetos anteriores. Além disso, uma AR promove a reutilização de experiências e facilita o desenvolvimento, padronização, qualidade e evolução de sistemas de software [Angelov et al. 2009, Nakagawa et al. 2014].

2. Proposta de Solução

Esta pesquisa tem como objetivo propor uma AR para ferramentas de GV seguindo o processo ProSA-RA, que é iterativo e sistematiza as etapas para construir, representar e avaliar ARs [Nakagawa et al. 2014]. ProSA-RA é composto de quatro etapas:

Investigação das fontes de informação:obtidas de pessoas, software, livros, modelos de referência, ontologias, etc. Para o desenvolvimento dessa etapa, um estudo secundário na forma de uma Revisão Sistemática da Literatura(RSL) seguindo as diretrizes propostas em [Kitchenham et al. 2009] foi realizado[1]. Foram identificadas **43** ferramentas de GV. A maioria das ferramentas utilizam processos específicos para modelar as *features* sendo que **13%** apoiam o método *Feature Oriented Domain Analisys*(FODA). **41%** foram desenvolvidas para modelagem de variabilidades e **23%** dão suporte à configuração de variabilidades. No que diz respeito à representação gráfica, **32%** utilizam representações hierárquicas em formato de árvore. Com relação à arquitetura, **46%** foram desenvolvidas como *plugins* e **11%** são baseados em arquiteturas multicamadas. Além disso, **27,9%** tem um mecanismo de persistência em banco de dados e **55%** dão suporte à arquivos XML/XMI. A Figura 1 apresenta uma visão conceitual da relação entre ferramentas de GV e AR e resume a perspectiva da pesquisa sobre a criação de AR. Observe que um conjunto de ferramentas de GV pode ser utilizado para fornecer informações ao projeto de uma AR. Tal AR serve como base para desenvolver novas ferramentas de GV.

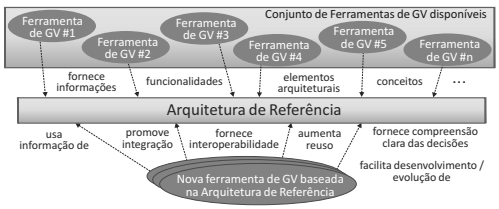


Figura 1. Relação entre Ferramentas de GV e AR

Somados às fontes de pesquisa anterior, pode-se destacar a RSL realizada por [Lisboa et al. 2010] que abrange as principais funcionalidades das ferramentas de GV. Tal estudo identificou **19** soluções com funcionalidades e objetivos como: ***i)Planejamento***, responsável por identificar informações para a definição do escopo do domínio; ***ii)Modelagem***, representa o escopo de domínio baseado em variabilidade, *features*, regras de composição, etc; e ***iii)Validação***, funcionalidades responsáveis pela validação do domínio.Outro estudo importante é a RSL de [Pereira et al. 2015] que identificou **41** ferramentas de GV. O estudo de [Pereira et al. 2015] utilizou a mesma classificação de funcionalidades de [Lisboa et al. 2010], sendo que **73%** necessitam de suporte às atividades de planejamento; **81%** dão suporte à derivação de produtos; [Berger et al. 2013] realizou um estudo importante no qual apresentou os tipos de abordagens e ferramentas de GV utilizados na indústria. Tal estudo identificou que **38%** são ferramentas de domínio específico, **29,4%** são ferramentas de código aberto e **26,5%** são ferramentas comerciais. Tais estudos contribuíram para o projeto da VMTools-RA com

[1]Um artigo sobre a RSL está em fase de submissão ao periódico *Information and Software Technology.*

a identificação de funcionalidades, características, tecnologias, conceitos sobre ferramentas de GV. As funcionalidades identificadas foram elencadas e agrupadas. Como exemplo pode-se citar: *Modelo de feature*, representa as variabilidades do produto; *Rastreabilidade*, relaciona *features* com requisitos; e *Documentação*.

Estabelecimento dos requisitos arquiteturais: Essa etapa visa elencar os requisitos da AR proposta a partir das informações identificadas na Etapa 1. Neste projeto, as funcionalidades encontradas serviram como base na identificação dos requisitos de sistema das ferramentas de GV. Como exemplo, para a funcionalidade *Rastreabilidade* temos o seguinte requisito de sistema: *Permitir rastreabilidade entre features e requisitos.* Após a identificação dos requisitos do sistema de GV observou-se que os mesmos poderiam ser agrupados em grupos mais abstratos identificando assim, os requisitos arquiteturais da AR proposta. Como exemplo temos os seguintes requisitos de sistema: *i) Permitir rastreabilidade entre features e requisitos*; *ii) Permitir modelo de feature*. Tais requisitos de sistema foram mapeados ao requisito arquitetural: *AR deve permitir modelagem de feature*.

Projeto arquitetural: os requisitos arquiteturais identificados serão utilizados como base para realizar o projeto arquitetural da AR. Descrições textuais, fluxogramas ou UML, por exemplo, podem ser utilizados para representar a AR em diferentes visões arquiteturais como estrutural, de implantação e de tempo de execução. A Figura 2 apresenta uma visão geral da VMTools-RA em uma representação mais abstrata com os principais elementos e suas relações. A VMTools-RA é baseada no estilo arquitetural em camadas: *i) camada de apresentação*, refere-se à interface do usuário; *ii) camada de aplicação*, contém o modelo lógico para gerenciar os diferentes elementos do GV; *iii) camada de persistência*, para o armazenamento das informações. Além disso, essa AR está dividida em três grupos: *modelagem*, representa os modelos de *features* e de variabilidade; *configuração*, permite criar e editar configurações dos modelos de *features* garantindo ao usuário gerar configurações válidas de um modelo de produto; e *validação*, verifica a consistência dos produtos de software garantindo que funcionem corretamente.

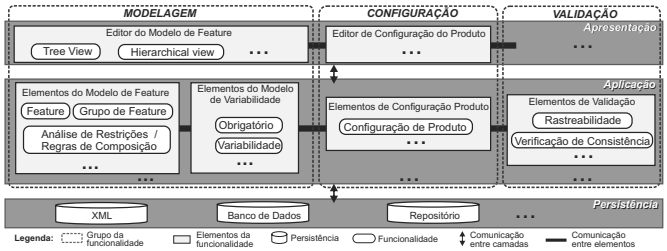


Figura 2. Visão Geral da VMTools-RA: Proposta inicial.

Avaliação arquitetural: Tal etapa está descrita a seguir na Seção 3.

3. Avaliação da Solução Proposta

A avaliação da AR tem por objetivo obter informações para averiguar: (i) se a AR está adequadamente representada; (ii) se são considerados os atributos de qualidade para a AR, tais como, integração e segurança; (iii) se a AR pode ser plenamente instanciada; e (iv) o que pode ser alterado a fim de evoluir a AR. Para a avaliação desta pesquisa, será realizado um estudo empírico e um estudo de caso. O estudo empírico envolverá

um estudo qualitativo e será apoiado por procedimentos de *Grounded Theory* como, por exemplo, *Coding* buscando codificar o conhecimento e respaldo dos especialistas sobre a AR proposta. Neste projeto, um questionário será elaborado e aplicado a especialistas em AR e GV. Os resultados obtidos com base nas respostas serão categorizados por meio da técnica de (*Coding*) para contribuir com a avaliação da viabilidade da AR. Já o estudo de caso envolverá a implementação de uma ferramenta de GV.

4. Atividades Realizadas

Até o momento foi realizada uma RSL com o objetivo de identificar ferramentas de GV existentes na acadêmia e na indústria, bem como analisar suas principais características, padrões arquiteturais e funcionalidades. Somados à RSL, outros estudos citados na Seção 2 estão sendo considerados para extrair as funcionalidades principais e definir os requisitos essenciais da AR de ferramentas de GV. Além disso, um conjunto de requisitos já foi identificado e uma proposta inicial de uma visão geral da VMTools-RA foi estabelecida.

5. Conclusão

Neste artigo foram apresentadas as atividades que estão sendo conduzidas para o estabelecimento da VMTools-RA, uma AR para ferramentas de GV. A partir da VMTools-RA, objetiva-se padronizar o desenvolvimento de novas ferramentas de GV por meio da identificação das principais funcionalidades, características, requisitos, conceitos e informações desse domínio. Espera-se com isso, contribuir com um maior reúso de experiências em projetos, aumentando a produtividade durante a fase de desenvolvimento dessas ferramentas, além de promover a integração e estabelecer as melhores práticas de desenvolvimento de software.

Referências

Angelov, S., Grefen, P., and Greefhorst, D. (2009). A classification of software reference architectures: Analyzing their success and effectiveness. In *WICSA/ECSA, Cambridge, UK*, pages 141–150.

Berger, T., Rublack, R., Nair, D., Atlee, J. M., Becker, M., Czarnecki, K., and Wsowski, A. (2013). A survey of variability modeling in industrial practice. In *VaMoS, Pisa, Italy*, pages 7:1–7:8.

Kitchenham, B., Pearl, B. O., Budgen, D., Turner, M., Bailey, J., and Linkman, S. (2009). Systematic literature reviews in software engineering. *Information and Software Technology*, 51(1):7–15.

Linden, F. J. v. d., Schmid, K., and Rommes, E. (2007). *Software product lines in action: The best industrial practice in product line engineering*. Springer-Verlag, New York.

Lisboa, L. B., Garcia, V. C., Lucrédio, D., Almeida, E. S., Meira, S. R. L., and Fortes, R. P. M. (2010). A systematic review of domain analysis tools. *Information and Software Technology*, 52(1):1–13.

Nakagawa, E. Y., Guessi, M., Maldonado, J. C., Feitosa, D., and Oquendo, F. (2014). Consolidating a process for the design, representation, and evaluation of reference architectures. In *WICSA, Sydney, NSW*, pages 143–152.

Pereira, J. A., Constantino, K., and Figueiredo, E. (2015). A systematic literature review of software product line management tools. In *ICSR, Miami, FL, USA*, pages 73–89.

SMartyComponents: Um Método para Especificação de Arquiteturas de Linhas de Produtos de Software Componentizadas

Marcio H. G. Bera, Edson OliveiraJr, Thelma E. Colanzi

[1]Programa de Pós-Graduação em Ciências da Computação (PCC) da
Universidade Estadual de Maringá (UEM).
Departamento de Informática, Av. Colombo, 5790 - Zona 07
CEP 87020-900 - Maringá - PR - Brasil

marciobera@hotmail.com, edson@din.uem.br, thelma@din.uem.br

Abstract. *Reuse is the key to productivity in software development. Approaches such as Software Product Line (SPL) and Component Based Development (CBD) support the reuse of artifacts and components.UML Components is a CBD method that stands out for assisting the user to identify as early as possible components. Stereotype-based Management of Variability (SMarty) is an approach used for identify and represent variabilities in UML models. Thus, this proposal uses the SMarty concepts applied in the UML Components process, allow this especify component-based SPLAs.*

Resumo. *Reúso é a chave para a produtividade no desenvolvimento de software. Abordagens como Linha de Produto de Software (LPS) e Desenvolvimento Baseado em Componentes (DBC) apoiam o reúso de artefatos e componentes. O UML Components é um método de DBC que se destaca por auxiliar o usuário a identificar os componentes o mais cedo possível. Stereotype-based Management of Variability (SMarty) é uma abordagem utilizada para identificar e representar variabilidades em modelos UML. Assim, esta proposta utiliza os conceitos de SMarty aplicados ao processo do UML Components, permitindo assim especificar ALPSs componentizadas.*

Palavras-chave: Arquitetura de Linha de Produto de Software, Variabilidade, Desenvolvimento Baseado em Componentes, *UML Components*, *SMarty*.

1. O problema de pesquisa e motivação

Com a crescente evolução da tecnologia nos últimos anos, a indústria de software busca meios para maximizar a produtividade no desenvolvimento de software [Jensen 2015]. Recursos são investidos na criação de produtos para um mesmo domínio sem que os artefatos destes produtos possam ser reutilizados.

Linha de Produto de Software (LPS) é uma abordagem que proporciona a reutilização dos artefatos, baseado em uma infraestrutura central denominada núcleo de artefatos, que contém artefatos com similaridades e variabilidades [Linden et al. 2007]. Uma LPS pode gerar vários produtos de uma mesma família, o que indica a existência de várias arquiteturas diferentes, ou seja, uma para cada produto específico. Uma abstração de todas estas arquiteturas é denominada Arquitetura de LPS (ALPS).

O Desenvolvimento Baseado em Componentes (DBC) é uma abordagem muito difundida na literatura, e que também visa o reúso. Existem várias abordagems de DBC na literatura, destaque para o *UML Components* [Cheesman and Daniels 2001], que permite a identificação de componentes nos primeiros *workflows*.

A abordagem ***S**tereotype-based **Ma**nagement of Vari**abil**i**ty*** (SMarty) [OliveiraJr et al. 2010, Marcolino et al. 2013] é formada por um perfil denominado *SMartyProfile*, e um processo denominado *SMartyProcess*. *SMartyProfile* possui um conjunto de estereótipos para representar variabilidades. *SMartyProcess* toma de entrada os artefatos de saída de um processo, e por meio de um conjunto de diretrizes, guia os *stakeholders* a identificar e representar variabilidades em tais artefatos.

2. Trabalhos relacionados

O estudo de Razavian e Khosravi [Razavian and Khosravi 2008] é uma abordagem anotativa para representar variabilidades em arquiteturas baseadas em componentes nos níveis de componentes, conectores e interfaces. Já o estudo de Haber *et al.* [Haber et al. 2011], é uma abordagem transformacional para representar variabilidades, que combina a representação modular das mudanças entre variantes do sistema com meios expressivos para capturar a influência das características dos produtos. O estudo de Nakagawa *et al.* [Nakagawa et al. 2013] apresenta um processo denominado *ProSA-RA2PLA*, que sistematiza a utilização de arquiteturas de referência para a construção de ALPSs. No entanto, nenhum destes trabalhos propõe uma maneira de especificar ALPSs componentizadas.

3. O trabalho proposto

A combinação do método *UML Components* com a abordagem *SMarty* já foi realizada em trabalhos realizados anteriormente, e a sua efetividade foi comprovada em [Contieri Junior et al. 2011]. Tal combinação foi denominada *SMartyComponents*, porém não houve nenhuma formalização da mesma. *SMarty* 5.2 fornece gerenciamento de variabilidades no relacionamento de componentes com portas e interfaces, relacionamento de portas com interfaces, e relacionamento de interfaces com operações. A Figura 1 apresenta um exemplo da LPS *Arcade Game Maker* (AGM) com representação de variabilidades *SMarty* em nível de porta e interfaces. A porta `pPlayGame` é um ponto de variação e possui três interfaces ≪alternative_OR≫. Os estereótipos *SMarty* em portas, são anotados no compartimento do componente.

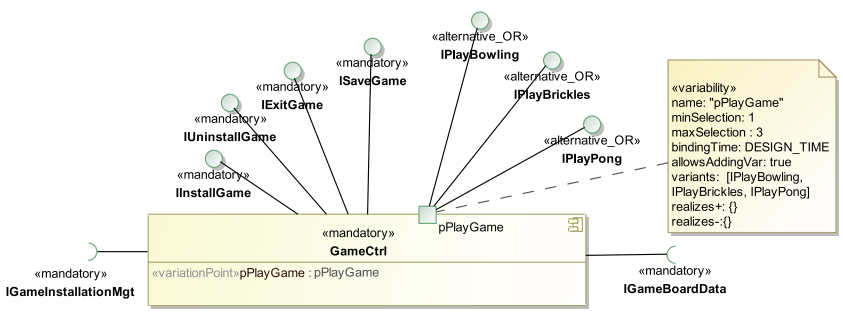


Figura 1. Exemplo de Representação de Variabilidade com *SMarty* 5.2

De um modo geral, *SMartyComponents* explora os *workflows* de Requisitos e Especificacão do *UML Components*. Assim, os artefatos resultantes de cada *workflow*, são

tomados de entrada pelo *SMartyProcess*, que identifica e, por meio dos estereótipos do *SMartyProfile*, representa as variabilidades. A Figura 2 apresenta a visão geral de *SMarty-Components*.

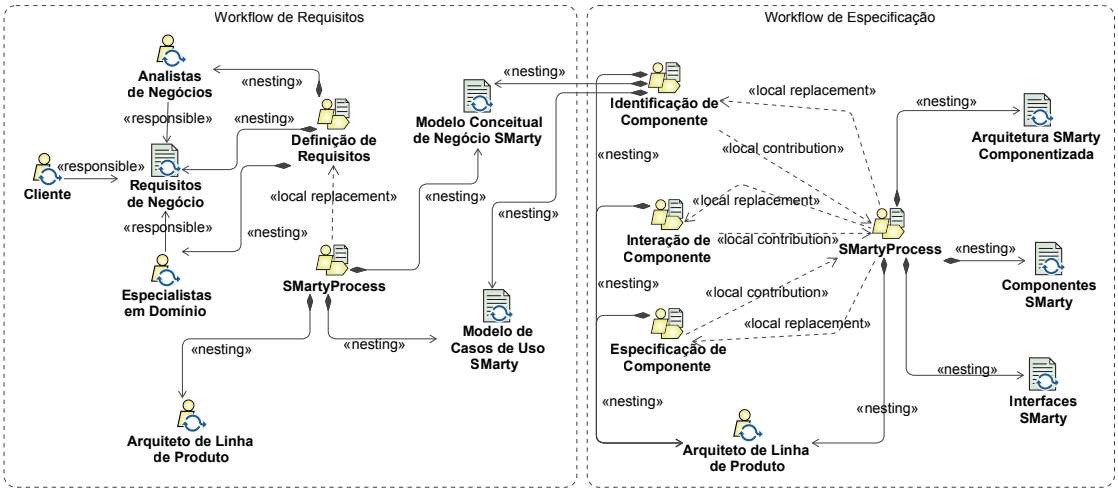


Figura 2. Visão Geral de *SMartyComponents*

Assim, ao término da última atividade do *workflow* de Especificação, serão gerados três artefatos: (i) *Arquitetura SMarty componentizada*; (ii) *Componentes SMarty*; e (iii) *Interfaces SMarty*. Um exemplo da atividade Iteração de Componente do *workflow* de Especificação pode ser visualizado na Figura 3. Neste exemplo, os artefatos `Interfaces do Sistema SMarty`, `Espec. de Componente & Arquitetura SMarty` e `Interfaces de Negócio SMarty` são entradas para as tarefas Descobrir Operações de Negócio, Refinar Interfaces & Operações e Refinar Espec. de Componentes & Arquitetura. A tarefa Refinar Interfaces & Operações gera o artefato `Especificação de Interfaces` e alimenta a tarefa Refinar Espec. de Componente & Arquitetura. Esta, de acordo com os artefatos recebidos, refina o artefato `Espec. de Componente & Arquitetura SMarty`. Os artefatos resultantes destas tarefas são entradas para o *SMartyProcess*. Por fim, os artefatos de saída do *SMartyProcess* contém as variabilidades identificadas e representadas.

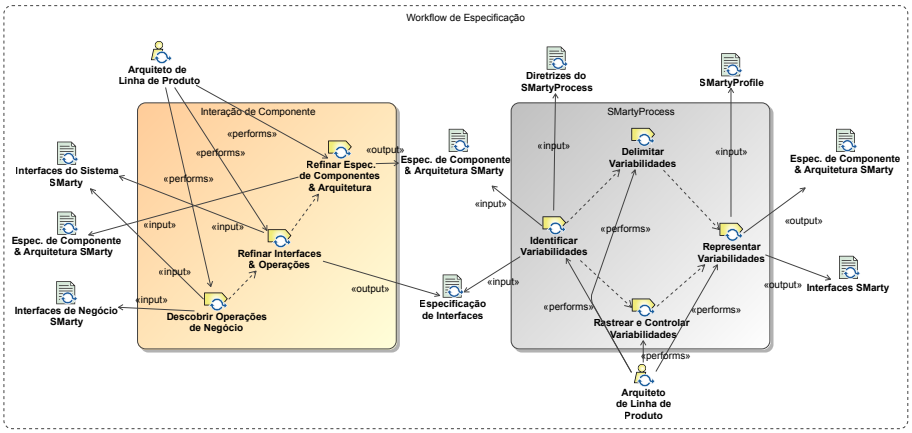


Figura 3. Atividade: Interação de Componente do *Workflow* de Especificação

4. Resultados

Em um Mapeamento Sistemático (MS) realizado, o estudo de Razavian e Khosravi [Razavian and Khosravi 2008] se destacou representar variabilidades em uma quantidade maior de elementos que demais estudos, por ser baseado em UML e por ser uma abordagem anotativa. Além disso, o estudo possui um estudo de caso com uma LPS denominada *Virtual University*. O estudo experimental comparou a efetividade de *SMarty* em relação a abordagem de Razavian e Khosravi. Os resultados obtidos neste estudo evidenciam a efetividade de *SMarty* e compõem um artigo publicado na *International Conference on Enterprise Information Systems* (ICEIS) [Bera et al. 2015]. A proposta de *SMartyComponents* foi definida e um estudo empírico qualitativo está sendo preparado para avaliá-la.

Referências

Bera, M. H. G., OliveiraJr, E., and Colanzi, T. E. (2015). Evidence-based SMarty Support for Variability Identification and Representation in Component Models. In *Proc. ICEIS*, pages 295–302.

Cheesman, J. and Daniels, J. (2001). *UML Components: A Simple Process for Specifying Component-based Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition.

Contieri Junior, A. C., Correia, G. G., Colanzi, T. E., Gimenes, I. M. S., OliveiraJr, E., Ferrari, S., Masiero, P. C., and Garcia, A. F. (2011). Extending UML Components to Develop Software Product-Line Architectures: Lessons Learned. In *Proc. ECSA*, pages 130–138.

Haber, A., Rendel, H., Rumpe, B., and Schaefer, I. (2011). Delta Modeling for Software Architectures. In *Proc. MBEES*, pages 1–10.

Jensen, R. W. (2015). *Improving Software Development Productivity: Effective Leadership and Quantitative Methods in Software Management*. Pearson Education, Inc., Westford, Massachusetts, USA.

Linden, F. J., Schmid, K., and Rommes, E. (2007). *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Marcolino, A., OliveiraJr, E., Gimenes, I., and Maldonado, J. (2013). Towards the Effectiveness of a Variability Management Approach at Use Case Level. In *Proc. SEKE*, pages 214–219.

Nakagawa, E. Y., Becker, M., and Maldonado, J. C. (2013). Towards a Process to Design Product Line Architectures Based on Reference Architectures. In *Proc. SPLC*, pages 157–161. ACM.

OliveiraJr, E., Gimenes, I., and Maldonado, J. (2010). Systematic Management of Variability in UML-based Software Product Lines. *Journal of Universal Computer Science*, 16(17):2374–2393.

Razavian, M. and Khosravi, R. (2008). Modeling Variability in the Component and Connector View of Architecture Using UML. In *Proc. AICCSA*, pages 801–809.

SMartyMetrics: uma Proposta de Framework de Métricas para Arquiteturas de Linha de Produto de Software

André Felipe Ribeiro Cordeiro, Edson OliveiraJr

[1]Departamento de Informática – Universidade Estadual de Maringá (UEM)
CEP 87020-900 – Maringá-PR – Brasil

cordeiroandrefelipe@gmail.com, edson@din.uem.br

Abstract. *This paper presents a research proposal of a metrics framework for evaluating Software Product Line Architectures (SPLA), taking into account the SMarty approach. Such an approach aims to managing variability management in UML-based Software Product Lines. The metrics study involves literature review, proposal and experimental validation of the metrics and the framework, based on SMarty UML models. Among main expected contributions from this work is the definition of a set of metrics to evaluate SPLAs and a framework, as well as improving the state of the art with regard to evaluation of SPLAs.*

Resumo. *Este artigo apresenta uma proposta de pesquisa no contexto de um framework de métricas para avaliação de Arquiteturas de Linha de Produto de Software (ALPS), considerando a abordagem SMarty. Tal abordagem permite o gerenciamento de variabilidades em Linhas de Produto de Software modeladas em UML. O estudo de métricas envolve a revisão da literatura, proposta e validação experimental das métricas e do framework, considerando modelos UML SMarty. Entre as principais contribuições esperadas para este trabalho estão a definição de um conjunto de métricas para avaliar ALPSs e de um framework, além do aprimoramento do estado da arte com relação à avaliação de ALPSs.*

1. Introdução

Linha de Produto de Software (LPS) é uma abordagem de desenvolvimento de software baseada no reuso sistemático e organizado em um domínio específico de atuação [Linden *et al.* 2007]. Os artefatos de uma LPS são classificados em artefatos comuns e variáveis [Capilla *et al.* 2013]. Os artefatos comuns estão presentes em todos os produtos derivados da LPS e geralmente envolvem características do domínio. Os artefatos variáveis são aqueles que diferenciam os produtos derivados da LPS.

O conjunto de artefatos definidos para uma LPS é armazenado em um repositório, conhecido como Núcleo de Artefatos [Linden *et al.* 2007]. Um dos artefatos

mais importantes do núcleo é a arquitetura da LPS (ALPS). Tal artefato descreve os detalhes arquiteturais, levando em consideração as características técnicas e de negócio da futura LPS.

Gerenciar esses artefatos torna-se relevante para o processo de adoção e manutenção de uma LPS. Entre as atividades de gerenciamento, está a Avaliação de LPS (AVLPS) [Oliveira Junior *et al.*, 2013]. A partir da AVLPS, os arquitetos e gerentes podem avaliar características como complexidade dos produtos derivados e *Return On Investment* (ROI) [Pohl *et al.* 2005]. Dada a complexidade do gerenciamento de LPSs, alguns métodos de AVLPS têm sido propostos na literatura, com critérios e processos sistemáticos.

Entre os métodos de AVLPS propostos, está o *Systematic Evaluation Method for UML-based Software Product Line Architectures* (SystEM-PLA) [Oliveira Junior *et al.* 2013], para avaliação de ALPSs modeladas em UML. Este método permite a avaliação de ALPSs nos estágios iniciais do processo de desenvolvimento, possibilitando que alterações realizadas na LPS não sejam onerosas ao projeto. O *SystEM-PLA* utiliza a abordagem *Stereotype-based Management of Variability* (*SMarty*) [Oliveira Junior *et al.* 2010] para realizar a(s) avaliação(ções). *SMarty* fornece um conjunto de estereótipos denominado *SMartyProfile* e um conjunto de diretrizes denominado *SMartyProcess.* O *SMartyProfile* permite a representação de artefatos comuns e variáveis em modelos UML da LPS e o *SMartyProcess* apresenta diretrizes para identificar e aplicar os estereótipos em elementos variáveis da LPS.

A abordagem SMarty é utilizada pelo método SystEM-PLA na geração de artefatos, mais especificamente, de modelos UML representativos de LPS, que permitem a realização das atividades de avaliação definidas pelo método.

Este projeto apresenta a proposta de um *framework* de métricas para as atividades de AVLPS. Tal *framework* deve incorporar as métricas já suportadas pelo SystEM-PLA [Oliveira Junior e Gimenes 2014; Marcolino *et al.* 2013], além de novas métricas, que devem ser definidas e avaliadas experimentalmente. Essas novas métricas são propostas com o objetivo de ampliar do escopo de AVLPS, considerando a abordagem SMarty.

2. Métricas para Arquitetura de LPS existentes no *SystEM-PLA*

As métricas apresentadas em Oliveira Junior e Gimenes (2014), Marcolino *et al.* (2013) e Oliveira Junior *et al.* (2008), para avaliar ALPSs modeladas em UML estão organizadas em duas categorias: métricas básicas e métricas para atributos de qualidade.

As métricas básicas [Oliveira Junior *et al.* 2008] verificam as características dos elementos presentes no(s) modelo(s) UML da LPS. Tais elementos podem ser comuns ou variáveis. Caso o elemento seja comum, é provável que o mesmo esteja em todos os

produtos derivados. Caso seja variável, a verificação de possíveis restrições entre os elementos do modelo é necessária.

As métricas para atributos de qualidade verificam a complexidade e extensibilidade de uma LPS a partir de seus produtos derivados. Tais atributos de qualidade foram adaptados ao contexto de LPS para auxiliarem na AVLPS. As métricas de complexidade [Marcolino *et al.* 2013] consideram a Complexidade Ciclomática (CC) e o número de Métodos Ponderados por Classe (*Weighted Methods per Class – WMC*), existentes em cada elemento da LPS. As métricas de extensibilidade [Oliveira Junior e Gimenes 2014] consideram a presença de características de herança e classes abstratas.

3. *SMartyMetrics*: um Framework de Métricas para ALPS

De forma sucinta, este projeto tem o objetivo de construir um framework de métricas para AVLPSs. Além das construção do *framework*, estão previstas as definições e validações experimentais de outras métricas para AVLPS.

O conjunto de métricas presentes no *Framework* deve contemplar as métricas existentes no *SystEM-PLA*, e o acréscimo de um novo conjunto de métricas, que devem ser investigadas neste projeto. A proposta de novas métricas acontece com o objetivo de ampliar o escopo das AVLPSs. A ampliação do escopo se faz necessária por conta das características não cobertas pelas métricas básicas e de atributos de qualidade existentes. Por exemplo, entre as características não cobertas, estão o esforço de manutenção e a estabilidade de classes**[**Alshayeb *et al.* 2011].

Algumas métricas, como a métrica para avaliar a estabilidade de classes não são exclusivas de LPS. Neste caso, tais métricas devem ser adaptadas considerando as informações relevantes para a AVLPS. Detalhes de possíveis adaptações ainda não estão definidos, pois tal decisão depende diretamente das métricas que forem selecionadas para a incorporação ao *framework*. Com relação as métricas que devem ser investigadas, ainda não se definiu o número de métricas, bem como quais características serão priorizadas. O que se sabe até o momento é que as futuras métricas deverão possibilitar análises significativas com a relação as LPSs modeladas em UML.

Para a construção do *Framework* de Métricas, objetivo geral deste projeto, estão sendo considerados os seguintes objetivos específicos: Revisão Sistemática de Literatura (RSL) sobre as métricas, tanto de LPS, quanto de Orientação a Objeto (OO), que possam ser aplicadas em Modelos de LPS UML *SMarty*; Validação Teórica e Experimental de tais métricas selecionadas na RSL; Representação das métricas adotando o *Structured Metrics Metamodel* (SMM) **(**SMM 2015**),** para possibilitar a troca de dados entre projetos/ferramentas; Proposição e Modelagem do *Framework* de Métricas e Validação Experimental do *Framework* de Métricas.

4. Conclusão

Este artigo apresentou um projeto de pesquisa contendo as atividades necessárias a construção de um *Framework* de métricas para AVLPSs, baseadas em modelos UML *SMarty*. O *SMartyMetrics* deve conter as métricas já suportadas pelo método *SystEM-PLA* e novas métricas que serão definidas e validadas. Ao final, espera-se que o *framework* construído contribua para a evolução do estado da arte na AVLPS, possibilitando a expansão da abordagem SMarty

Referências

Alshayeb, M.; Naji, M.; Elish, M. O.; Al-Ghamdi, J. Towards measuring object-oriented class stability. *IET Software*, 2011, v. 05, p. 415-424, 2011.

Capilla, R.; Bosch, J.; Kang, K. Systems and Software Variability Management: Concepts, Tools and Experiences. SpringerLink : Bucher. Springer, 2013.

Linden, F. J.; Schmid, K.; Rommes, E. *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007.

Marcolino, A. S.; Oliveira Junior E. A.; Gimenes I. M. S.; Conte, T. U. Towards Validating Complexity-Based Metrics for Software Product Line Architectures. In: *SBCARS*, 2013, Brasília. 2013. v. 1. p. 69-94.

Oliveira Junior**,** E. A.; Gimenes, I. M. S. Empirical Validation of Product-line Architecture Extensibility Metrics. In: *ICEIS*, 2014, Lisbon. v. 2. p. 111-118.

Oliveira Junior, E. A.; Gimenes, I. M. S.; Maldonado, J. C.; Masiero, P. C.; Barroca, L. Systematic Evaluation of Software Product Line Architectures. *Journal of Universal Computer Science*, 2013, v. 19, p. 25-52, 2013.

Oliveira Junior, E. A.; Gimenes, I. M. S.; Maldonado, J. C. Systematic Management of Variability in UML-based Software Product Lines. *Journal of Universal Computer Science* (JUCS), 2010, v. 16, p. 2374-2393, 2010.

Oliveira Junior**,** E. A.; Gimenes, I. M. S.; Maldonado, J. C. A Metric Suite to Support Software Product Line Architecture Evaluation. In: *CLEI*, 2008, Santa Fe. 2008. p. 489-498.

Pohl, K.; Bockle, G.; Linden, F. J. *Software Product Line Engineering: Foundations, Principles, and Techniques*. Secaucus, NJ, USA: Springer-Verlag, 2005.

SMM. Structured Metrics Meta-Model. OMG Group. 2015. Disponível em : http://www.omg.org/spec/SMM/. Acesso em 14/04/2015.

Estudo de Caracterização de Bugs de Projetos de Código Aberto

Guilherme A. de Oliveira[1], **Humberto T. Marques-Neto**[1]

[1]Instituto de Ciências Exatas e Informática
Pontifícia Universidade Católica de Minas Gerais (PUC Minas)
30.535-901 – Belo Horizonte – MG – Brasil

Resumo. *Em sistemas de código aberto populares, uma grande quantidade de requisições de manutenção são reportados em sistemas de gerenciamento de bugs. Por exemplo na fundação Mozilla, cerca de 250 bugs foram criados por dia em 2012. Com certeza, entender melhor o processo de manutenção pode melhorar a produtividade dos desenvolvedores que resolvem bugs. Este trabalho realiza uma caracterização de bugs de um sistema de código aberto, a partir do ciclo de vida de um bug. Os resultados dessa caracterização apontam a necessidade de se investir no processo de aceitação de novos bugs e também no controle de qualidade para verificação de bugs resolvidos.*

1. Introdução

A manutenção é uma das fases mais importantes e custosas de um software [Erlikh 2000, Mookerjee 2005]. Desenvolvedores de projetos de código aberto, normalmente, trabalham nas requisições de manutenção (i.e., *bugs*) assim que elas ficam disponíveis no sistema de gerenciamento de *bugs*, e.g., Bugzilla, Jira [Mockus et al. 2002, Liu et al. 2012].

Entretanto, sistemas de código aberto populares podem possuir uma grande quantidade de desenvolvedores e de *bugs*. Por exemplo, a fundação Mozilla[1] possuía mais de 280 mil *bugs* cadastrados entre 2009 e 2012. Nesse mesmo período de tempo, 5.045 desenvolvedores interagiram de alguma forma na resolução desses *bugs*. Somente em 2012, foram criados aproximadamente 7.500 *bugs* por mês com 1.390 desenvolvedores trabalhando ativamente na resolução destes *bugs*.

Essa enorme quantidade de *bugs* aliada ao trabalho não coordenado de muitos desenvolvedores pode prejudicar a produtividade das tarefas de manutenção em grandes sistemas de código aberto [Mockus et al. 2002, Tan and Mookerjee 2005]. Uma análise do processo de manutenção poderia indicar pontos críticos a serem aperfeiçoados para melhorar esse cenário. Este trabalho apresenta uma caracterização de *bugs* do Mozilla, realizada com o objetivo de ajudar no entendimento do processo de manutenção de sistemas de código aberto.

2. Estudo

Esta seção descreve o estudo realizado com 283.971 *bugs* da fundação Mozilla, que possui *bugs* de diversos produtos como Firefox, Thunderbird, entre outros. Foram estudados *bugs* criados entre 2009 e 2012, para caracterizar o processo de manutenção. Esse processo é referido como o ciclo de vida de um *bug* e mostra o fluxo de trabalho para a

[1]www.mozilla.org, acessado em 2015-04-01.

resolução de um *bug* [Anvik et al. 2006, Anvik and Murphy 2011]. A Figura 1 mostra o ciclo de vida juntamente com as porcentagens de *bugs* analisados no estudo. Os valores nas transições dessa figura mostram a porcentagem de *bugs* que saíram do estado de origem. O número dentro de cada estado indica a porcentagem de *bugs* que permaneceram naquele mesmo estado. Portanto, somando os valores dentro do estado juntamente com as transições de saída, obtém-se 100%.

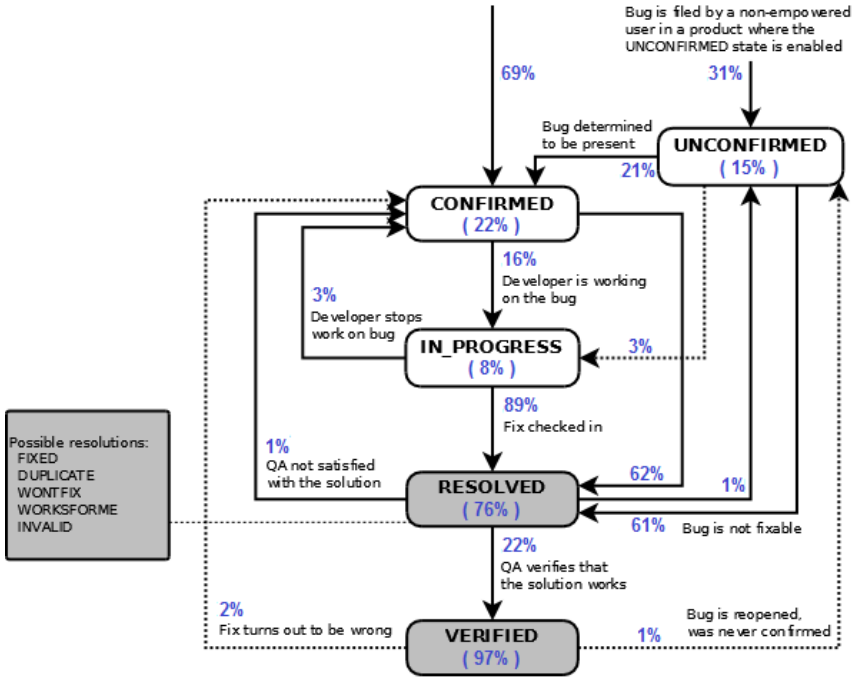


Figura 1. Ciclo de vida de um bug no Bugzilla

Durante o processo de manutenção, um *bug* pode caminhar por diversos estados do ciclo de vida. Quando um novo *bug* é reportado no Bugzilla, ele pode ser registrado como *Unconfirmed* ou *Confirmed* (respectivamente 31% e 69% dos *bugs* da base). Normalmente, os *bugs* reportados por usuários comuns são registrados como *Unconfirmed*, e super-usuários (i.e., usuários com permissões especiais) podem criar *bugs Confirmed*. Entretanto, em certos projetos Mozilla, usuários comuns também podem criar *bugs Confirmed*. Por isso a quantidade de *bugs Confirmed* (69%) neste estudo é maior que *Unconfirmed* (31%).

Bugs Unconfirmed podem mudar para *Confirmed* se receberem votos de outros usuários comuns confirmando a ocorrência do problema, ou se algum super-usuário decidir mudar seu estado. Super-usuários também podem mudar o estado de *Unconfirmed* para *Resolved*, caso o *bug* reportado não possa ser corrigido (i.e., inválido, duplicado, etc.) ou caso não era um problema de fato. Interessante notar que a maioria dos *bugs Unconfirmed* do estudo são resolvidos diretamente (61%) e apenas uma pequena parte (21%) segue para o início do processo de manutenção como *Confirmed*. Em poucas ocasiões (3%), um super-usuário pode atribuir um *bug Unconfirmed* diretamente para um desenvolvedor trabalhar nele, mudando assim seu estado para *In_progress*.

Quando um *Confirmed bug* é designado a um desenvolvedor, este *bug* assume o estado de *In_progress*. Também é possível que um *Confirmed bug* não possa ser corri-

gido, nesse caso o *bug* é marcado como *Resolved*. Pode-se notar que a porcentagem de *Confirmed bugs* que são marcados diretamente como resolvidos (62%) é bem próxima a dos *Unconfirmed* que seguem o mesmo caminho (61%). Outra informação inesperada é que apenas 16% dos *Confirmed bugs* são alocados para desenvolvedores trabalharem.

Um *bug* no estado *In_progress* indica que existe um desenvolvedor trabalhando ativamente nele. Se o desenvolvedor enviar a resolução do *bug*, o estado muda para *Resolved*. Os dados do estudo indicam que a maioria dos *bugs In_progress* (89%) são resolvidos. Se o desenvolvedor parar de trabalhar no *bug*, este volta para o estado de *Confirmed*. Segundo os *bugs* da base estudada, são ocorrências incomuns um *bug* retornar para *Confirmed* (3%).

O estado *Resolved* representa *bugs* que foram resolvidos ou fechados. Caso seja necessário, um *bug* resolvido pode ser verificado por usuários do controle de qualidade (QA – Quality Assurance) para certificar que a resolução está realmente correta. Nesse caso, depois de ser aprovado pela verificação, o *bug* assume o estado de *Verified*. Como mostrado na Figura 1, apenas 22% dos *bugs* estudados passaram por esse controle de qualidade. Também é possível que um *bug* dado como resolvido retorne ao estado de *Confirmed* (1%) ou *Uncofirmed* (1%) se o controle de qualidade não ficar satisfeito com a solução. Entretanto, a maior parte dos *bugs* resolvidos (76%) permanecem como *Resolved* e não são verificados pelo controle de qualidade.

Um *bug* verificado pode retornar aos estados iniciais do processo de manutenção como *Confirmed* ou *Unconfirmed*. Entretanto, esse cenário é bastante incomum ocorrendo em 3% dos *bugs* estudados. Então, 97% dos *bugs* verificados permanecem nesse estado.

A Tabela 1 apresenta a quantidade de *bugs* estudados que passaram, em algum momento do seu ciclo de vida, por algum estado do processo de manutenção. Essa tabela também mostra a porcentagem de *bugs* que passaram por aquele estado, de acordo com o total de bugs criados entre 2009 e 2012 (283.971 *bugs*). Além disso, a Tabela 1 apresenta o tempo de espera (em dias) de *bugs* nesse estado. Nesse tempo de espera, foram considerados apenas os *bugs* que saíram do estado, i.e., esta informação ignora os *bugs* que permaneceram no mesmo estado porque neste caso o tempo de espera seria infinito.

Pode-se perceber que o estado em que se passaram a maior quantidade de *bugs*, foi o *Resolved*, que foi também o estado que obteve a menor média de dias de espera. Os estados *Unconfirmed* e *Confirmed* são os que tem as maiores médias de tempo de espera. A quantidade de *bugs* que caminharam pelo estado *Unconfirmed* foram bem menores que *Confirmed*. A quantidade de *bugs* que passaram por *In_progress* é a menor da base estudada. Isso era esperado visto que grande parte dos *bugs* reportados são resolvidos diretamente. Para o estado *Verified* a quantidade de *bugs* que passaram por esse estado e a média de dias de trabalho são relativamente baixa, indicando que poucos testes podem ser feitos em *bugs* resolvidos.

3. Conclusão

Este trabalho apresenta uma avaliação de 283.971 *bugs* da fundação Mozilla. Analisando estes *bugs* no processo de manutenção, é possível encontrar informações interessantes. É importante evidenciar que os projetos Mozilla precisam de investimentos maiores no processo de aceitação de novos *bugs*. Isso se deve ao fato que mais de 60% (170 mil) dos

Tabela 1. Bugs Caminhando pelos Estados do Ciclo de Vida do Mozilla

Estado	Bugs		Tempo de Espera (dias)				
	Quantidade	%	Min	Max	Média	Desvio	Mediana
Unconfirmed	87.216	30,71	0	1360	93	184	1
Confirmed	214.996	75,71	0	1373	61	150	6
In_Progress	47.448	16,71	0	1312	33	95	5
Resolved	226.685	79,83	0	1259	2	45	6
Verified	49.573	17,46	0	1229	15	73	7
Total	283.971	100	–	–	–	–	–

bugs são resolvidos logo após sua criação. Estes *bugs* são fechados justamente porque não podem ser corrigidos (por exemplo, por serem inválidos ou duplicados).

Outro ponto importante é que apenas 22% dos *bugs* resolvidos são verificados pelo controle de qualidade do Mozilla. Uma análise mais aprofundada é necessária para descobrir o motivo dessa ocorrência. Isso pode indicar a necessidade de atrair mais pessoas para a área de controle de qualidade da fundação Mozilla.

Como trabalhos futuros, pretende-se realizar um estudo em intervalos menores de tempo da base de dados, para verificar se o comportamento analisado neste trabalho varia em conformidade com o intervalo de tempo. Outra possibilidade seria usar algoritmos de *clusterização* para criar grupos mais coesos de *bugs* e analisar o processo de manutenção para cada grupo. Finalmente, pode-se analisar a produtividade dos desenvolvedores em relação aos tipos de *bugs* resolvidos.

Referências

Anvik, J., Hiew, L., and Murphy, G. C. (2006). Who should fix this bug? In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, pages 361–370, New York, NY, USA. ACM.

Anvik, J. and Murphy, G. C. (2011). Reducing the effort of bug report triage: Recommenders for development-oriented decisions. *ACM Trans. Softw. Eng. Methodol.*, 20(3):10:1–10:35.

Erlikh, L. (2000). Leveraging legacy system dollars for e-business. *IT Professional*, 2(3):17–23.

Liu, K., Tan, H. B. K., and Chandramohan, M. (2012). Has this bug been reported? In *20th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE)*, pages 28:1–28:4.

Mockus, A., Fielding, R. T., and Herbsleb, J. D. (2002). Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology*, 11(3):309–346.

Mookerjee, R. (2005). Maintaining enterprise software applications. *Communications of the ACM*, 48(11):75–79.

Tan, Y. and Mookerjee, V. (2005). Comparing uniform and flexible policies for software maintenance and replacement. *IEEE Transactions on Software Engineering*, 31(3):238–255.

Análises Estruturais para Identificação de Falso-Positivos em Recomendações de Refatoração

Rafael S. Lima, Ricardo Terra

Departamento de Ciência da Computação,
Universidade Federal de Lavras (UFLA), Brasil

rafaelsplima@computacao.ufla.br,terra@dcc.ufla.br

Resumo. *Desenvolvedores – para atingir objetivos de curto prazo – realizam alterações de código que se opõem à organização estrutural existente. Nesse cenário, ferramentas de identificação de oportunidades de refatoração são normalmente aplicadas no intuito de reverter tais alterações. O problema, entretanto, consiste no fato de que tais ferramentas reportam uma parcela considerável de falsos positivos. Diante disso, este estudo tem o intuito de prover desenvolvedores com uma série de análises que visam explicar o porquê o método deve ser refatorado ou não. No estado atual da pesquisa, foram propostas três análises aplicáveis em refatorações Extract Method e Move Method cujas implementações estão sendo realizadas no Ambiente Moose.*

1. Introdução

Desenvolvedores – para atingir objetivos de curto prazo – realizam alterações de código que se opõem à organização estrutural existente [8]. Para reverter essas alterações, normalmente são aplicadas refatorações que consistem no processo de alteração de um sistema de software sem modificar seu comportamento no intuito de melhorar a estrutura e o entendimento do código [1, 4].

Diversas técnicas vêm sendo propostas na literatura para a identificação de oportunidades de refatoração, por exemplo, baseadas em *Feature Envy bad smells* [9], métricas [2], similaridade estrutural [6, 7], etc. O *problema*, entretanto, consiste no fato de que normalmente essas ferramentas reportam diversas recomendações e que uma parcela relevante consiste de falsos positivos, o que implica em uma baixa precisão.

Diante disso, este estudo – a partir de uma série de recomendações sugeridas por outras ferramentas – tem o intuito de prover desenvolvedores com uma série de análises estruturais para identificação de falsos positivos, visando explicar o porquê o método deve ser refatorado ou não. Por exemplo, assuma que o método m da classe C depende dos tipos $\{A, B\}$ enquanto que todos os outros métodos de C dependem de $\{X, Y\}$. Uma ferramenta baseada em similaridade estrutural recomendaria mover m para uma outra classe C' que seja mais similar.

No entanto, por mais coerente que se pareça, uma análise da dependência em nível de pacotes poderia indicar que A, B, X e Y pertencem ao mesmo pacote p, o que poderia levar a um questionamento por parte do desenvolvedor.

O restante deste artigo está organizado conforme a seguir. A Seção 2 introduz conceitos fundamentais ao estudo. A Seção 3 apresenta a proposta de pesquisa descrevendo um conjunto de análises estruturais no auxílio de refatorações. E, por fim, a Seção 4 conclui descrevendo as contribuições esperadas.

2. *Background*

Similaridade Estrutural: Coeficiente de similaridade mede o grau de correspondência entre duas entidades conforme um critério estabelecido [5]. Assume-se que uma entidade de código fonte (e.g., um método) é representada pelo conjunto de tipos com quais ela estabelece dependência. Desse modo, o cálculo de similaridade entre duas entidades pode ser realizado usando o coeficiente *Jaccard*, que considera a relação existente entre o número de tipos comuns e o número de tipos encontrados em cada entidade, como a seguir:

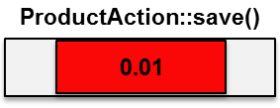
$$sim(E_1, E_2) = \frac{a}{a + b + c}$$

onde $\boldsymbol{a}$ = número de tipos comuns, $\boldsymbol{b}$ = números de tipos exclusivos à entidade E_1, $\boldsymbol{c}$ = números de tipos exclusivos à entidade E_2. Por exemplo, considere dois métodos $m_1 = \{B, C, D\}$ e $m_2 = \{C, D\}$. Como $\boldsymbol{a} = 2$, $\boldsymbol{b} = 1$ e $\boldsymbol{c} = 0$, $sim(m_1, m_2) = 0.666$.

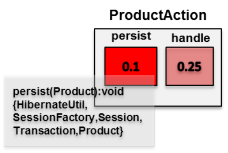
Extract Method: Métodos longos que acumulam diversas responsabilidades – o que implica em baixa coesão e alto acoplamento no nível de métodos – são *bad smells comumente* encontrados em sistemas de software. Por exemplo, assuma um método que realize duas responsabilidades. Nesse cenário, desenvolvedores normalmente aplicam a refatoração *Extract Method* que extrai um fragmento do método original para um novo método. Essa refatoração, além de contribuir diretamente para a modularidade, promove reúso e reduz duplicação de código [1, 4].

Move Method: Métodos implementados em classes incorretas – o que implica em baixa coesão e alto acoplamento no nível de classes – também são *bad smells* comumente encontrados em sistemas de software. Nesse cenário, desenvolvedores normalmente aplicam a refatoração *Move Method* que move o método para uma classe mais apropriada. Essa refatoração diminui o acoplamento e aumenta a coesão em nível de classe, além de promover a organização [1, 4].

Moose: É um ambiente independente de linguagem para engenharia reversa e reengenharia de sistemas de software.[1] Moose provê um conjunto de serviços que incluem um meta-modelo comum, visualização e avaliação de métricas, repositório de modelos e um apoio visual para consulta, navegação e agrupamento [3]. A Figura 1 ilustra um exemplo de duas visualizações do Moose baseadas em similaridade estrutural: (a) uma que reporta oportunidades de refatoração *Extract Method* que pôde identificar um bloco no método `save` que depende de tipos diferentes do restante do método; e (b) uma que reporta oportunidades de refatoração *Move Method* que pôde identificar que o método `persist` depende de tipos diferentes dos demais métodos da classe `ProductAction`.



(a) Rec. de *Extract Method*



(b) Rec. de *Move Method*

Figura 1. Exemplo de visualizações no Moose

[1] http://www.moosetechnology.org

3. Proposta de Pesquisa

O objetivo do presente estudo é prover análises *baseadas em similaridade estrutural* que validem ou invalidem recomendações providas por ferramentas de identificação de oportunidades de refatoração. Conforme ilustrado na Figura 2, logo que uma ferramenta apontar uma oportunidade de refatoração, disponibilizar-se-á uma série de análises que visam explicar o porquê o método deve ser de fato refatorado ou não.

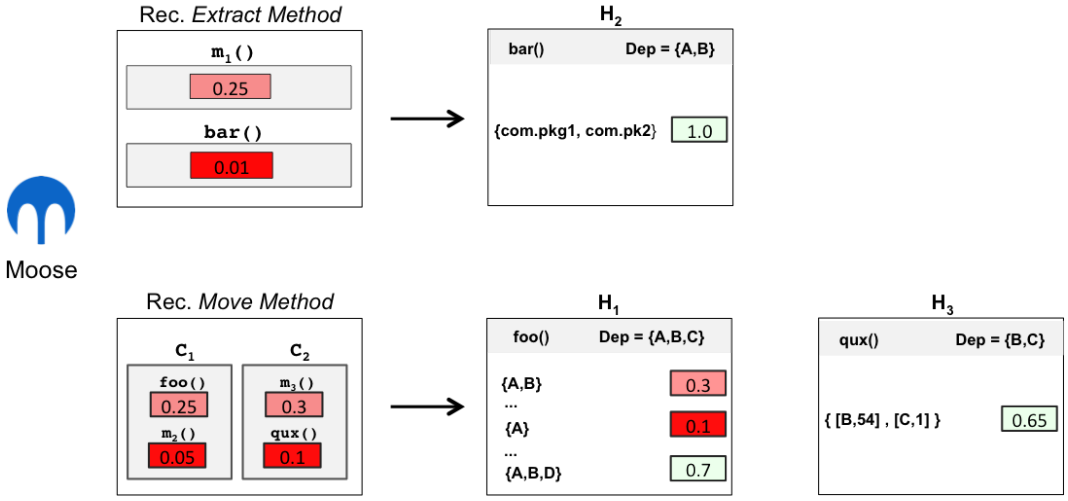


Figura 2. Proposta de Pesquisa

No estado atual desta pesquisa, foram propostas análises – as quais são ilustradas na Figura 2 – com o objetivo de verificar as seguintes hipóteses:

- H_1. *Apenas um ou mais tipos são responsáveis pela baixa similaridade.* Isso indicaria que uma sugestão de refatoração não seria recomendada se o método não dependesse de um certo tipo e/ou fosse adicionado um novo tipo. Por exemplo, o método `foo` depende de $\{A, B, C\}$ e tem similaridade de 0.25 com a classe atual. Se C fosse retirado e D adicionado, a similaridade subiria para 0.7.
- H_2. *Embora os tipos sejam diferentes, a maioria pertence ao mesmo pacote.* Isso indicaria que uma sugestão de refatoração não seria dada se a similaridade fosse calculada no nível de pacotes e não de tipos. Por exemplo, um bloco do método `bar` depende de $\{A, B\}$ e tem similaridade de 0.01 com o restante do método. Se o conjunto considerado fosse o do pacote dos tipos – $\{com.pkg1, com.pkg2\}$ – a similaridade subiria para 1.0.
- H_3. *Os tipos que a entidade depende são largamente (no fator quantitativo) utilizados*. Isso indicaria que uma sugestão de refatoração não seria dada se a similaridade fosse calculada considerando multiplicidade, i.e., o número de vezes que um tipo é acessado. Por exemplo, o método qux depende de $\{B, C\}$ e tem similaridade de 0.1 com a classe atual. Se o conjunto considerasse multiplicidade – $\{[B, 54], [C, 1]\}$ – a similaridade subiria para 0.65.

As análises supracitadas foram elaboradas a partir de justificativas de desenvolvedores para não acatarem certas recomendações. Novas análises serão propostas por investigações qualitativas em repositório de sistemas de código aberto. A avaliação, contudo,

será realizada por meio de questionários. Serão enviadas recomendações que casem com alguma análise proposta, indagando o desenvolvedor se o mesmo a aplicaria ou não com as devidas justificativas, as quais serão utilizadas para aperfeiçoar e corrigir as análises.

4. Considerações Finais

Desenvolvedores – para atingir objetivos de curto prazo – realizam alterações de código que se opõem à organização estrutural existente. Para reverter essas alterações, normalmente são aplicadas refatorações. Nesse cenário, existem diversas ferramentas que identificam oportunidades de refatoração baseadas em *Feature Envy bad smells*, métricas, similaridade estrutural, etc.

A maior contribuição desta pesquisa consiste em prover desenvolvedores com uma série de análises que visam explicar o porquê o método deve ser refatorado ou não. O estudo limita-se a refatorações *Extract Method* e *Move Method* e utiliza ferramentas centradas em similaridade estrutural adaptadas para o ambiente Moose [6, 7]. No entanto, é esperado que, com o decorrer da pesquisa, um maior número de refatorações e ferramentas sejam incorporadas ao estudo.

Agradecimentos

Este trabalho foi apoiado pela FAPEMIG, CAPES e CNPq.

Referências

[1] Martin Fowler. *Refactoring: improving the design of existing code*. Addison-Wesley, Boston, 1999.

[2] Radu Marinescu. Detection strategies: Metrics-based rules for detecting design flaws. In *20th International Conference on Software Maintenance (ICSM)*, pages 350–359, 2004.

[3] Oscar Nierstrasz, Stéphane Ducasse, and Tudor Gĭrba. The story of Moose: An agile reengineering environment. In *10th European Software Engineering Conference (ESEC)*, pages 1–10, 2005.

[4] William Opdyke. *Refactoring object-oriented frameworks*. PhD thesis, University of Illinois at Urbana-Champaign, 1992.

[5] H. Charles Romesburg. *Cluster Analysis for Researchers*. Lulu Press, North Carolina, 2005.

[6] Vitor Sales, Ricardo Terra, Luis Fernanda Miranda, and Marco Tulio Valente. Recommending Move Method refactorings using dependency sets. In *20th Working Conference on Reverse Engineering (WCRE)*, pages 232–241, 2013.

[7] Danilo Silva, Ricardo Terra, and Marco Tulio Valente. Recommending automated Extract Method refactorings. In *22nd International Conference on Program Comprehension (ICPC)*, pages 146–156, 2014.

[8] Ricardo Terra and Marco Tulio Valente. A dependency constraint language to manage object-oriented software architectures. *Software: Practice and Experience*, 32(12):1073–1094, 2009.

[9] Nikolaos Tsantalis and Alexander Chatzigeorgiou. Identification of Move Method refactoring opportunities. *IEEE Transactions on Software Engineering*, 99:347–367, 2009.

Formação de Equipes de Alto Desempenho para Desenvolvimento de Software

Alessandra C. S. Dutra, Rafael Prikladnicki

[1]Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Av. Ipiranga, 6681 – 90619-900 – Porto Alegre – RS – Brazil

{alessandra.dutra,rafaelp}@pucrs.br

Resumo. *Este artigo descreve o trabalho que tem sido desenvolvido para analisar a formação de equipes de alto desempenho para desenvolvimento de software frente às abordagens de capacitação existentes, avaliando a oportunidade de propor uma abordagem metodológica de capacitação visando formar equipes de alto desempenho para o desenvolvimento de software.*

1. Introdução

O mercado de desenvolvimento de software opera em um ambiente global, com mudanças rápidas, e precisam responder com agilidade a estas novas oportunidades e a estes novos mercados [Sommerville 2011]. Conseguir agilidade, competitividade e resultados sem uma equipe de desenvolvimento de software capacitada e de alto desempenho é uma tarefa difícil e pode trazer resultados pouco competitivos.

Este contexto indica que a formação qualificada e a capacitação de profissionais são cada vez mais necessárias na sociedade em que vivemos. Seja em cursos de curta duração, graduação ou pós-graduação, formar bons profissionais faz parte do compromisso das Instituições de Ensino Superior (IES) com a sociedade [Enricone 2006].

A qualidade da capacitação em ES pode contribuir significativamente à melhoria do estado da arte do desenvolvimento de software e auxiliar a solução de alguns problemas tradicionais e crises relacionadas com as práticas da indústria de software [Gibbs 1994]. Hoje, a capacitação e o treinamento para formar profissionais de software devem incluir não apenas conhecimentos básicos na área de computação, mas também o ensino de conceitos, processos e técnicas para definição, desenvolvimento e manutenção de software [Saiedian 1999] [ACM/IEEE 2013].

Neste sentido, o processo de ensino e aprendizagem de Engenharia de Software tem passado por questionamentos acerca dos métodos utilizados nas atividades de capacitação [Santos et al 2008]. Estudos recentes observam que estes métodos envolvem estratégias tradicionais de ensino, tais como apresentação de teoria, aulas expositivas e leituras complementares, fazendo com que os alunos encontrem na indústria um cenário diferente do que é ensinado na academia [Santos et al 2008] [Prikladnicki et al 2009].

Nesse contexto, faz-se necessário saber quais os resultados de investigações científicas em relação às características da formação de equipes de alto desempenho em desenvolvimento de software e as atuais abordagens de capacitação existentes e que de alguma forma exploram algumas destas características.

Este artigo tem como objetivo apresentar o trabalho que tem sido desenvolvido nesta pesquisa, onde pretende-se analisar a formação de equipes de alto desempenho para desenvolvimento de software frente às abordagens de capacitação existentes, avaliando a oportunidade de propor uma abordagem metodológica de capacitação visando formar equipes de alto desempenho para o desenvolvimento de software.

2. Referencial Teórico

A Engenharia de Software é uma disciplina preocupada com a aplicação de teoria, conhecimento e prática para o desenvolvimento efetivo e eficiente de sistemas de software que satisfaçam os requisitos dos usuários [ACM/IEEE 2008]. Os profissionais de ES [Conn 2002], estão insatisfeitos com a falta de preparo dos estudantes universitários que ingressam no mercado de trabalho, o que leva a indústria a ter que complementar a sua educação com treinamentos. Este preparo pode envolver profissionais ou equipes, incluindo equipes de alto desempenho.

Uma equipe de alto desempenho [Moscovici 2003], além de ter todos os requisitos de uma equipe, tem seus membros comprometidos com o crescimento e o sucesso pessoal de cada um dos membros da equipe. Esta equipe supera o desempenho de todas as outras equipes, além de conseguir resultados além das expectativas**.**

3. Metodologia de Pesquisa

3.1. Desenho e Etapas da Pesquisa

Para o desenvolvimento desta pesquisa, será utilizada a metodologia proposta por Mafra [Mafra et al 2006], cujo desenho de pesquisa está representado na Figura 1. Cada uma das etapas desta metodologia é detalhada a seguir [Mafra et al 2006], [Shul et al 2001]:

• Estudos Secundários: essa etapa consiste na condução de estudos secundários, tais como revisão sistemática da literatura, que tem como objetivo buscar evidências primárias na área em estudo.

• Proposta Inicial: essa etapa consiste na produção, com base no conhecimento adquirido e nas evidências identificadas através da condução da revisão sistemática, de uma proposta inicial da tecnologia.

• Estudo de Viabilidade: essa etapa verifica, através de estudos empíricos e experimentais, a viabilidade do processo ou tecnologia sendo analisada. Pesquisas utilizando estudantes são aplicáveis nesta etapa.

• Estudo de Observação: essa etapa avalia todos os passos, em detalhes, que constituem o processo de aplicação da nova tecnologia de forma a garantir que cada passo é efetivo e é executado na ordem correta.

• Estudo de Caso com Ciclo de Vida Real: essa etapa consiste de um caso de estudo avaliando o processo ou tecnologia em um ciclo de vida real de desenvolvimento de software.

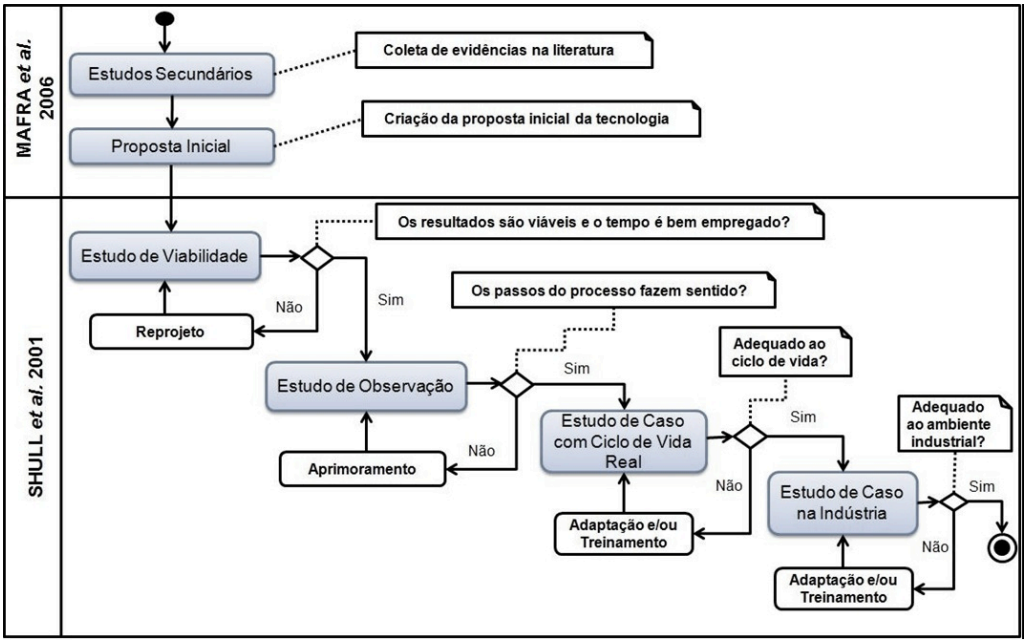


Figura 1 - Metodologia Experimental – [Mafra et al 2006]

• Estudo de Caso na Indústria: uma vez que nas etapas anteriores o processo já foi adaptado para ser executado em um ciclo de vida real e demonstrou ser eficaz, nesta fase o processo ou tecnologia é avaliado em um ambiente de indústria.

4. Resultados Preliminares

4.1. Revisão Sistemática

Como resultados preliminares foi executada uma Revisão Sistemática da Literatura onde identificamos algumas características que as equipes de alto desempenho devem ter no desenvolvimento de software. Identificamos características organizacionais, comportamentais e técnicas. As mais citadas foram comunicação eficiente, coordenação, trabalho em equipe, diversidade da equipe, liderança, coesão e motivação. Podemos sugerir que as equipes de alto desempenho (1) possuem uma comunicação eficaz, (2) apresentam uma diversidade que estimula a aprendizagem e a inovação, (3) possuem coesão, motivação, liderança e coordenação, a fim de alcançar seus objetivos.

Estes resultados geraram uma visão inicial das características esperadas para uma equipe de alto desempenho. Estas equipes precisam ser capacitadas e desenvolver seus pontos fortes, visando proporcionar um conjunto de competências que somente estas equipes apresentam. A capacitação neste caso é, portanto, um aspecto essencial para o desenvolvimento destas equipes.

5. Próximos Passos da Pesquisa

Esta pesquisa tem como passo seguinte o planejamento e a execução de um estudo de viabilidade. Este estudo terá como objetivo aprofundar a análise nas características das equipes de alto desempenho, identificando suas práticas, bem como técnicas de capacitação utilizadas pelas empresas para aprimorar estas práticas em seus projetos de desenvolvimento de software.

A pesquisa será exploratória, qualitativa, e executada através de uma survey. As pessoas entrevistadas serão: gerentes de projetos, líderes de projetos e coordenadores de projetos. A coleta de dados será feita através de entrevistas semi-estruturadas com questões abertas e fechadas. A aplicação do Questionário será com entrevistas pessoais.

A análise de dados será feita através de uma apresentação das contribuições do estudo e uma análise crítica com relação a estes resultados. Nesta análise será desenvolvida uma confrontação dos resultados obtidos com as teorias e estudos relacionados e será realizada uma análise qualitativa dos dados coletados, através de um mapeamento das respostas dos entrevistados.

Após o estudo de viabilidade, serão planejados e executados estudos de observação e de estudos de caso com ciclo de vida real, visando avaliar as práticas de alto desempenho e as técnicas de capacitação identificadas a partir das características destas equipes.

Referências Bibliográficas

Sommerville, I. (2011) "Engenharia de Software" 9a edição. São Paulo: Pearson Prentice Hall.

Enricone, D. (2006). "Ser Professor", 5a edição, EDIPUCRS.

Gibbs, W. (1994) "Software's chronic crisis". Scientific American 2713, pp. 86–95.

Saiedian, H. (1999) "Software engineering education and training for the next millennium, Journal of Systems and Software", v. 49, i. 2-3, p. 113-115

ACM/IEEE. (2013) Computer Science Curriculum, Guidelines for Undergraduate Degree Programs in Software Engineering.

Santos, R. P., Santos, P. S. M., Werner, C. M. L., Travassos, G. H. (2008) "Utilizando Experimentação para Apoiar a Pesquisa em Educação em Engenharia de Software no Brasil", Fórum de Educação em Engenharia de Software.

Prikladnicki, R., Albuquerque, A., Wangenheim Santos et al 2008, e Cabral R., (2009) "Ensino de Engenharia de Software: Desafios, Estratégias de Ensino e Lições Aprendidas," no FEES.

Conn, R. (2002) "Developing Software Engineers at the C-130J Software Factory". IEEE Software,Los Alamitos, v. 19, n. 5, p. 25-29.

Moscovici, F. (2003) "Equipes dão certo: A multiplicação do Talento Humano". Rio de Janeiro: José Olympio, 8a edição.

Mafra, S., Barcelos, R., Travassos, G. H. (2006) "Aplicando uma Metodologia Baseada em Evidência na Definição de Novas Tecnologias de Software". In: Proceedings of the 20th Brazilian Symposium on Software Engineering (SBES 2006), v. 1, pp. 239 – 254, Florianopolis.

Shull, F., Carver, J., Travassos, G. H. (2001) "An empirical methodology for introducing software processes". SIGSOFT Softw. Eng. Notes, 26(5):288–296.

Software Crowdsourcing: Barriers Faced by the Crowd

Leticia Santos Machado[1], Rafael Prikladnicki[1]

[1]Faculdade de Informática– Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Av. Ipiranga, – 90.619-900 – Porto Alegre – RS – Brazil

leticia.smachado@gmail.com, rafaelp@pucrs.br

Abstract. *Software Engineering has recently started to explore the Crowdsourcing's model in tasks of software development seeking collective solutions to problems, ways to accelerate the time-to-market and reduce costs. However, the crowd can face many challenges when contributing to a task in a crowdsourcing context. The purpose of this research is to collect empirical evidences in order to understand what are the barriers that hinder crowd workers in software crowdsourcing projects, and propose a set of strategies that can be used to support the crowd in software crowdsourcing projects.*

1. Introduction

The evolution of global accessibility through Web 2.0 media technologies have created opportunities that have transformed the collaborative format of the teams distributed software development [Begel et al, 2012; Peng et al. 2014]. Soon, classical organizations such as companies and open source communities will be replaced by decentralization and interrelation of software ecosystem it employs crowdsourcing, outsourcing, offshoring, spontaneous collaboration and social networking [Kaganer, 2013]. The introduction of collaboration ubiquitous tools already indicates changes in the design, development, test, software product delivery or software system solutions.

Crowdsourcing is a distributed problem-solving model. The term "crowdsourcing" was coined by Jeff Howe when discussing how businesses were effectively using the Internet to outsource work to many individuals (Howe, 2008). In this study we adopt the widely accepted definition, crowdsourcing is the act of an organization outsourcing their work to an undefined, networked labor using an open call for participation.

Crowdsourcing in software development derives from crowdsourcing more generally. It means to engage a global pool of online workers that can be tapped on-demand to provide software solutions or services [Lakhani, Garvin and Lonstein, 2010], [Stol and Fitzgerald, 2014]. Small, atomized, tasks that can be completed and paid for in small increments are unprecedented in the history of work. Software has been the pioneer in all the large mega-trends of the last generation: in computer technology, technological entrepreneurship, offshore outsourcing, and now—in crowdsourcing.

In order to effectively support software crowdsourcing, there are computational platforms that handle the technical aspects of the crowdsourcing tasks including the broadcasting of tasks to be performed, the selection of tasks, reception of results of the

tasks, and so on. Examples of these platforms include TopCoder [Lakhani, Garvin and Lonstein, 2010], Utest and Amazon Mechanical Turk.

Thus there are three key components in a crowdsourcing project (Figure 1). The platform is the middleman, i.e., it intermediates the communication between the two other parties. Second, is the crowd—the workers who will effectively perform the tasks. The crowd is a global dispersed and undefined crowd. Third, on the left, are the buyers or requesters. These are the firms that place the requests for work (the tasks) [Prikladnicki et al. 2014]. In this study we are interested to investigate the crowd component.

Figure 1. Basic model Crowdsourcing

An industry case study of crowdsourcing software development was presented in [Stol and Fitzgerald, 2014]. In this related work, it discussed a number of challenges that arise when software crowdsourcing process was adopted during a project of software development, such as: lifecycle model, software development tasks interdependencies, overhead in terms of company effort to prepare specifications and answer crowdsourcing community queries and quality issues. The authors also synthesized a set of six key concerns, which had relevance in a software development context: task decomposition, coordination and communication, planning and scheduling, quality assurance, knowledge and intellectual property and motivation and remuneration.

The goal of our research is to explore the barriers faced by the crowd when contributing to software crowdsourcing projects and analyzing which practices can be used to support the crowd in the software development life cycle. In other words we want to answer the following research question: "What are the barriers and the approaches that can be used to support the crowd in software crowdsourcing projects?" In order to answer these questions, a set of secondary research questions were defined:

- What are the barriers faced by the crowd when contributing to software crowdsourcing projects?
- What are the aspects that influence the quality and delivery time of the solutions provided by the crowd in software crowdsourcing projects?
- What are the approaches that can be used to help the crowd in developing successful software crowdsourcing projects?

2. Research Design

Software Crowdsourcing is a relatively new and emergent phenomenon with limited empirical research and theory. For this reason, this study uses an exploratory approach. An initial ad hoc literature review was conducted with the purpose of sharing the basic concepts and identifying the main challenges of software crowdsourcing. The research

design is composed of three phases, as presented in Figure 2 and, described as following.

Warm up. This phase consists of the planning and execution of interviews conducted with software developer professionals in order to explore their software crowdsourcing experiences and to better understand the problem addressed by this study.

Phase I – Find Barriers. This phase is composed of several empirical studies that have as a goal to identify the barriers that hinder the crowd of contributing to software crowdsourcing projects. We will first gather the barriers from the literature, by means of systematic and snowballing literature review. We will then conduct interviews with software crowdsourcing platforms users (the crowd). The results obtained in the literature review and in the interviews will be analyzed in focus groups that will be executed with a different set of crowd participants.

Phase II – Proposing practices to overcome barriers. In this phase, the goal is to consolidate the barriers that hinder the crowd of contributing to software crowdsourcing projects and propose a set of approaches that can be used to support the crowd and overcome the identified barriers. We plan to propose the approaches based on: (i) recommendations and empirical evidence that emerge during the Phase I; (ii) and the practices and experiences that crowd workers have used and adopt in their software crowdsourcing projects. In the Phase 2, we intend to design and conduct a controlled experiment to assess the effectiveness of using the proposed solutions to the selected barriers.

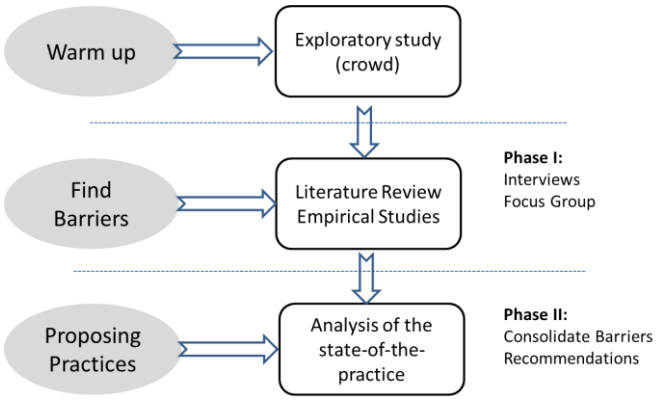


Figure 2. Research Design

3. Preliminary Results

During the warm up phase we have conducted several interviews with crowd workers from the Brazilian IT industry [Machado et al., 2014]. Preliminary findings showed that software crowdsourcing has potential and could benefit both the companies and the professionals. The main blockers about their experiences with software crowdsourcing were summarized as it follows:

- Unavailability of documentation;
- Poor feedback;
- Scarce context project information;

- Specific knowledge of the technologies and business rules.

Based on these impressions found, we plan to talk about several crowd workers in person, by voice, by video, and/or by email. We will keep on conducting semi-structured interviews and analyzing these data, and additional data will focus on the barriers (challenges) that they face when involved with software crowdsourcing projects. We will also collect feedback from Brazilian buyers to obtain information about Brazil crowdsourcing initiatives that companies had participated in and the challenges faced by the crowd. We will then be able to generate recommendations that will be useful for crowd workers, as well as to better understand the use of crowdsourcing in the context of software engineering.

Acknowledgment

The authors would like to thank the CNPq (process number 309000/2012-2) for the financial support.

References

Begel, A., Herbsleb, J. D. and Storey, M. (2012). "The future of collaborative software development", Proceedings of the ACM Conference on Computer Supported Cooperative Work Companion.

Peng, X., Babar, M. A., & Ebert, C. (2014). “Collaborative Software Development Platforms for Crowdsourcing”. IEEE software, 31(2), 30-36.

Kaganer, E., Carmel, E., Hirscheim, R., and Olsen, T. (2013), “Managing the Human Cloud”. MIT Sloan Management Review, 54(2), 23-32.

Howe, J. (2008). “Crowdsourcing: How the power of the crowd is driving the future of business”. Random House.

Lakhani, K., Garvin, D. A., & Lonstein, E. (2010), “Topcoder (a): Developing software through crowdsourcing”. Harvard Business School General Management Unit Case.

Stol, K. J., & Fitzgerald, B. (2014) “Two's company, three's a crowd: a case study of crowdsourcing software development”, In Proceedings of the 36th International Conference on Software Engineering (pp. 187-198). ACM.

Prikladnicki, R., Machado, L., Carmel, E., and de Souza, C. R. B. (2014) “Brazil Software Crowdsourcing: A First Step in a Multi-year Study”, 1st International Workshop on CrowdSourcing in Software Engineering (CSI-SE). Collocated with the 36th International Conference on Software Engineering (ICSE), Hyderabad, India

Machado, L., Pereira, G., Prikladnicki, R., Carmel, E. & de Souza, C. R. (2014). “Crowdsourcing in the Brazilian IT industry: what we know and what we don't know.” In *Proceedings of the 1st International Workshop on Crowd-based Software Development Methods and Technologies* (pp. 7-12). (CrowdSoft) ACM. Collacated with the 22nd Foundations of Software Engineering (FSE), Hong Kong.

Decisões sobre arquitetura de software em projetos que utilizam métodos ágeis

Andrey Baumhardt Ramos[1], **Raquel Aparecida Pegoraro**[1]

[1]Departamento de Ciência da Computação – Universidade Federal da Fronteira Sul (UFFS) – Chapecó, SC – Brasil

Andreybramos@hotmail.com, Raquel.pegoraro@uffs.edu.br

Abstract. *In the last few years the agile method using has been increasing. However, the lack of architectural focus in agile methods as created the necessity of developing evolutionary architecture studies. This research has the following objectives: (a) To identify the project evolution and software architecture problems and their reasons in the agile methods development; (b) To identify the practices that allow it to improve the capacity software architecture evolution; (c) To propose a decision analysis estructure that relates problems and practices, allowing the teams to make the right decisions on the agile projects architecture. With this study we hope to give knowledge basis about how to create a software architecture for agile projects.*

Resumo. *A adoção dos métodos ágeis vem crescendo significativamente nos últimos anos. Porém, a falta de foco arquitetônico tem gerado a necessidade de desenvolver estudos sobre a arquitetura evolutiva centrada em projetos que utilizam métodos ágeis. Esta pesquisa tem como objetivos: (a) Identificar os problemas relacionados à evolução do projeto e arquitetura de software e suas causas em projetos que utilizam métodos ágeis; (b) Identificar quais práticas permitem melhorar a capacidade de evoluir a arquitetura de software em projetos que utilizam métodos ágeis; (c) Propor uma estrutura de análise de decisão relacionando problemas e práticas que permita que as equipes, baseadas nas situações dos seus projetos, tomem decisões referentes à arquitetura de projetos ágeis. Espera-se com este estudo gerar uma base de conhecimento sobre arquitetura de software para projetos ágeis que permita que as empresas tomem melhores decisões sobre o assunto.*

1. Introdução

A importância de vincular metas de qualidade com a arquitetura de software é conhecida e praticada na construção de projetos de domínio complexo, sendo considerada um dos primeiros passos para construção de sistemas escaláveis [Ramakrishnan 2010],[Pressman 2011]. O projeto da arquitetura está preocupado com a compreensão de como um sistema deve ser organizado e com será a estrutura geral desse sistema, sendo o elo crítico entre o projeto e a engenharia de requisitos, pois identifica os principais componentes estruturais de um sistema e os relacionamentos entre eles [Sommerville 2011]. O planejamento maciço inicial de requisitos e arquitetura é muito utilizado na abordagem tradicional de desenvolvimento de software [Pressman 2011].

Para lidar com o fato que os requisitos evoluem e que a mudança é algo normal e aceitável em projetos de software, surgiram os métodos ágeis. Esses métodos se caracterizam por utilizarem abordagem iterativa e incremental de desenvolvimento, com equipes auto-organizadas, com constante colaboração do cliente que se ajusta dinamicamente às suas necessidades através da aceitação de mudanças de requisitos em qualquer momento do projeto, que busca a minimização de produtos de trabalho de engenharia de software, entre eles a documentação de software, e a simplicidade no desenvolvimento [Abbas et al. 2010], [Dybå and Dingsøyr 2008].

Apesar da crescente adoção dos métodos ágeis pelas empresas e de vários relatos de sucesso da sua adoção, os mesmos são criticados pela a sua falta de foco arquitetônico [Hochstein and Lindvall 2005], [Dybå and Dingsøyr 2008]. Há um consenso entre vários autores sobre a necessidade de pesquisar sobre arquitetura evolutiva centrada no desenvolvimento ágil, isso devido a importância das decisões dos envolvidos ao longo do projeto e da necessidade de adequação às mudanças [Babar and Abrahamsson 2008], [Ambler 2008], [Hadar and Silberman 2008], [Chen and Babar 2014]. Porém, a revisão sistemática feita por [Breivold et al. 2010] em estudos publicados sobre arquitetura em métodos ágeis aponta que há uma discrepância entre a literatura de arquitetura de software e os métodos ágeis, e que há falta de apoio científico e de estudos empíricos para conhecer plenamente as vantagens e desvantagens dos método ágeis de software, no que tange a arquitetura de software.

[Parsons et al. 2007] afirmam que para uma adoção bem sucedida de ágil é necessário não apenas a seleção de um método ágil, mas também adoção de técnicas apropriadas que possibilitem combinação de qualidade e melhoria do processo. Destacam que muitas equipes podem não ter o conhecimento e habilidades para escolher as melhores técnicas a serem utilizadas, e que a estratégia usual é de adotar um método ou prática ágil que julgar conveniente, avaliar e após adaptá-lo para melhorar os resultados. Utilizar esta estratégia de definição das práticas para decisões relacionadas a arquitetura do software pode ser um risco para a sua evolução.

Segundo [Babar 2009] problemas como a falta de tempo para considerar escolhas de arquitetura, ou a falta de foco em atributos de qualidade, podem ocasionar a dificuldade de manutenções e altos custos para os projetos que adotam abordagens ágeis. Algumas práticas podem ser adotadas para conseguir lidar com essas situações, tais como criar uma fase pré-desenvolvimento para planejar as diferentes opções relevantes de arquitetura do sistema, ou utilizar o atendimento aos atributos de qualidade como uma medida de sucesso.

Neste contexto, percebe-se que há uma lacuna na literatura sobre métodos ágeis e o impacto da arquitetura nos projetos, o que é entendido como um fator crítico na utilização desses métodos e, para se beneficiar plenamente dos métodos, é necessário entender esta correlação de dependência. A partir desta problemática, este projeto busca responder à seguinte questão de pesquisa: “Como evoluir de forma sistematizada e com qualidade a arquitetura de software em projetos que utilizam métodos ágeis?”

Motivados pela necessidade de compreender a evolução do projeto e arquitetura de software em empresas que utilizam métodos ágeis e de investigar maneiras de apoiar esta evolução, os objetivos desta pesquisa são:

- Identificar os problemas relacionados à evolução do projeto e arquitetura de software e suas causas em projetos que utilizam métodos ágeis;
- Identificar quais práticas permitem melhorar a capacidade de evoluir a arquitetura de software em projetos que utilizam métodos ágeis;
- Propor uma estrutura de análise de decisão para apoiar as escolhas e definições sobre a documentação de arquitetura de software em projetos de software que utilizam métodos ágeis. A estrutura analítica será composta pela relação entre os problemas e práticas identificados na fase anterior da pesquisa.

Espera-se através da estrutura proposta que as empresas, baseados no contexto específico dos seus projetos, possam refletir sobre as escolhas a cerca da arquitetura de software e tomar melhores decisões.

2. Procedimentos Metodológicos

A metodologia que será utilizada na pesquisa consistirá nos seguintes passos:

1. Mapeamento sistemático da literatura com o objetivo de identificar os fatores críticos que impactam na evolução satisfatória arquitetura de projetos de software que utilizam métodos ágeis, e seus efeitos. O resultado será apresentado através do diagrama causa e efeito, o qual permite estruturar hierarquicamente as causas de determinado problema ou oportunidade de melhoria. Para realização do mapeamento sistemático da literatura serão seguidas as recomendações de [Kitchenham et al. 2010];
2. Mapeamento sistemático da literatura identificando as práticas utilizadas para construir e evoluir a arquitetura de software em projetos que utilizam métodos ágeis, e as situações em que cada prática pode ser utilizada [Kitchenham et al. 2010];
3. Estudo de caso exploratório com o objetivo de identificar as práticas de arquitetura utilizadas pelas empresas e os desafios enfrentados [Yin 2015];
4. Embasado nos achados das etapas anteriores desta pesquisa será proposta uma estrutura de análise de decisão para apoiar as escolhas e definições sobre a documentação de arquitetura de software, relacionando problemas ou situações de projetos e práticas para solução.

3. Resultados esperados

Ao final deste projeto de pesquisa espera-se alcançar os seguintes resultados:

- Identificação dos fatores críticos que impactam na evolução satisfatória da arquitetura de projetos de software que utilizam métodos ágeis, e seus efeitos;
- Identificação das práticas utilizadas para construir e evoluir a arquitetura de software em projetos que utilizam métodos ágeis;
- Compreensão dos desafios enfrentados pelas empresas de software quanto a arquitetura de software na utilização de métodos ágeis;
- Proposição de uma estrutura de análise de decisão, relacionando problemas e práticas, que permita que as equipes, baseadas nas situações dos seus projetos, tomem melhores decisões referente à arquitetura do software.

Referências

Abbas, N., Gravell, A. M., and Wills, G. B. (2010). The impact of organization, project and governance variables on software quality and project success. In *Agile Conference (AGILE), 2010*, pages 77–86. IEEE.

Ambler, S. W. (2008). Agile software development at scale. In *Balancing agility and formalism in software engineering*, pages 1–12. Springer.

Babar, M. (2009). An exploratory study of architectural practices and challenges in using agile software development approaches. In *Software Architecture, 2009 European Conference on Software Architecture. WICSA/ECSA 2009. Joint Working IEEE/IFIP Conference on*, pages 81–90.

Babar, M. A. and Abrahamsson, P. (2008). Architecture-centric methods and agile approaches. In *Agile Processes in Software Engineering and Extreme Programming*, pages 242–243. Springer.

Breivold, H. P., Sundmark, D., Wallin, P., and Larsson, S. (2010). What does research say about agile and architecture? In *Software Engineering Advances (ICSEA), 2010 Fifth International Conference on*, pages 32–37. IEEE.

Chen, L. and Babar, M. A. (2014). Towards an evidence-based understanding of emergence of architecture through continuous refactoring in agile software development. In *Software Architecture (WICSA), 2014 IEEE/IFIP Conference on*, pages 195–204. IEEE.

Dybå, T. and Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and Software Technology*, 50(9):833–859.

Hadar, E. and Silberman, G. M. (2008). Agile architecture methodology: Long term strategy interleaved with short term tactics. In *Companion to the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications*, pages 641–652. ACM.

Hochstein, L. and Lindvall, M. (2005). Combating architectural degeneration: a survey. *Information and Software Technology*, 47(10):643–656.

Kitchenham, B., Pretorius, R., Budgen, D., Brereton, O. P., Turner, M., Niazi, M., and Linkman, S. (2010). Systematic literature reviews in software engineering–a tertiary study. *Information and Software Technology*, 52(8):792–805.

Parsons, D., Ryu, H., and Lal, R. (2007). The impact of methods and techniques on outcomes from agile software development projects. In *Organizational Dynamics of Technology-Based Innovation: Diversifying the Research Agenda*, pages 235–249. Springer.

Pressman, R. S. (2011). *Engenharia de software:Uma Abordagem Profissional*. McGraw Hill Brasil.

Ramakrishnan, S. (2010). On integrating architecture design into engineering agile software systems. *Issues in Informing Science and Information Technology*, 7.

Sommerville, I. (2011). *Engenharia de Software*. Pearson Education do Brasil, 9 edition.

Yin, R. K. (2015). *Estudo de Caso-: Planejamento e Métodos*. Bookman editora.

Sistema Multiplataforma para o Controle de Denúncias: modelagem para implantação em órgãos públicos de fiscalização

Lucas S. Rodrigues, Fernando M. Federson

Instituto de Informática – Universidade Federal de Goiás (UFG)
Alameda Palmeiras, Quadra D, Câmpus Samambaia
Caixa Postal 131 - CEP 74001-970 - Goiânia – GO - Brasil

{lucas.soarod, federson}@gmail.com

Abstract. *The capacity of the Society to inform the possible violations to the supervisory bodies responsible by efficient communication channels is always a challenge for any government. In order to assist the Government in this task, we propose a process and the resulting modeling of a tool that can adapt to many regulatory agencies using latest technologies, especially mobile technology. A model of business processes is outlined from the case study of administrative practices employed in providing the service within the Regional Council of Engineering and Agronomy of Goiás. The architecture of a multiplatform system is also presented to meet the modeled processes.*

Resumo. *A Sociedade poder informar as possíveis infrações às entidades fiscalizadoras responsáveis através de canais de comunicação eficientes é um desafio sempre presente para qualquer Governo. No sentido de auxiliar o Governo nesta tarefa, são propostos um processo e a modelagem resultante de uma ferramenta que pode se adaptar a muitos órgãos fiscalizadores utilizando tecnologias recentes, em especial, a tecnologia móvel. Um modelo de processos de negócio é delineado a partir do estudo de caso de práticas administrativas empregadas no oferecimento do serviço de denúncias no âmbito do Conselho Regional de Engenharia e Agronomia de Goiás. A arquitetura de um sistema multiplataforma também é apresentada para atender os processos modelados.*

1. Introdução

No Brasil, o chamado Governo Eletrônico determina uma série de diretrizes com foco na eficiência e efetividade das funções governamentais através da TIC (Tecnologia da Informação e Comunicação) para democratizar o acesso à informação, dar transparência de processos e informações e ainda dinamizar a prestação de serviços públicos [1]. O uso efetivo das tecnologias disponíveis, como passo seguinte à formulação de diretrizes, possibilitaria às agências governamentais, em especial as de caráter fiscalizador, darem uma resposta eficiente às situações de irregularidades informadas pela população, mas este recurso é ainda pouco explorado no Brasil. Este trabalho tem como propósito demonstrar um caso de aplicação dessas diretrizes.

Nas próximas seções apresentamos, resumidamente, a metodologia utilizada, a elucidação do processo de tratamento de denúncias empregado atualmente no Conselho Regional de Engenharia e Agronomia de Goiás (CREA-GO), o novo modelo de processos melhorado e a arquitetura de sistema, baseada nas plataformas móvel e *web*. Por fim, apresentamos os resultados obtidos e nossos objetivos futuros.

2. Metodologia

A metodologia empregada na execução do presente trabalho foi baseada na seguinte sequência de atividades: 1**)** Elucidação do processo de negócio atual; 2**)** Avaliação do modelo atual; 3**)** Proposta de um novo modelo de processos de negócios; 4**)** Especificação da Arquitetura de Sistema que suporte o modelo de processos proposto. O padrão de modelagem utilizado para os processos de negócio é o *Business Process Modelling Notation* (BPMN) [2]. A arquitetura de sistema foi dividida em Perspectiva de Implantação e Perspectiva Modular. Na Perspectiva de Implantação foi utilizado o padrão de notação *Unified Modeling Language* (UML) [3], através do Diagrama de Implantação. Para a Perspectiva Modular foi elaborado uma explanação textual para entendimento do modelo.

3. Modelo de Processos de Negócio

3.1. Análise e Avaliação

A seguir explicamos o funcionamento do processo de negócio para tratamento de denúncias atualmente praticado no CREA-GO. O processo inicia-se com a manifestação por parte do **cidadão** (Sociedade → Cidadão) sobre alguma irregularidade no campo profissional, por exemplo: execução de obra sem o acompanhamento de um Engenheiro Civil. A comunicação é feita através de e-mail ou telefone, à Central de Denúncias do CREA-GO, depois o **cadastramento** é realizado no sistema existente. É feito uma **análise** para identificar se realmente é uma denúncia procedente relativo ao escopo do órgão. Caso seja procedente, é pedido ao Departamento de Fiscalização uma **programação** da data e o **fiscal** que realizará a denúncia. Esta elucidação estará contida também na Figura 1, onde foi proposto um modelo melhorado para este processo.

O gargalo percebido é: o sistema computacional utilizado atualmente não permite que além da própria Central de Denúncias, os demais atores também encaminhem registros entre si e para o sistema. Isso faz com que todo e qualquer direcionamento deva enviar uma resposta (informação) à Central.

3.2. Modelo Proposto

Na Figura 1, é apresentado um modelo proposto após correção do gargalo identificado na subseção anterior. Tanto para a perspectiva de fases, que foram definidas como Cadastro, Análise, e Execução, quanto para as tarefas modeladas, o modelo de negócio especificado permite a “fluidez” da informação para que chegue com rapidez ao agente de fiscalização que averiguará os fatos.

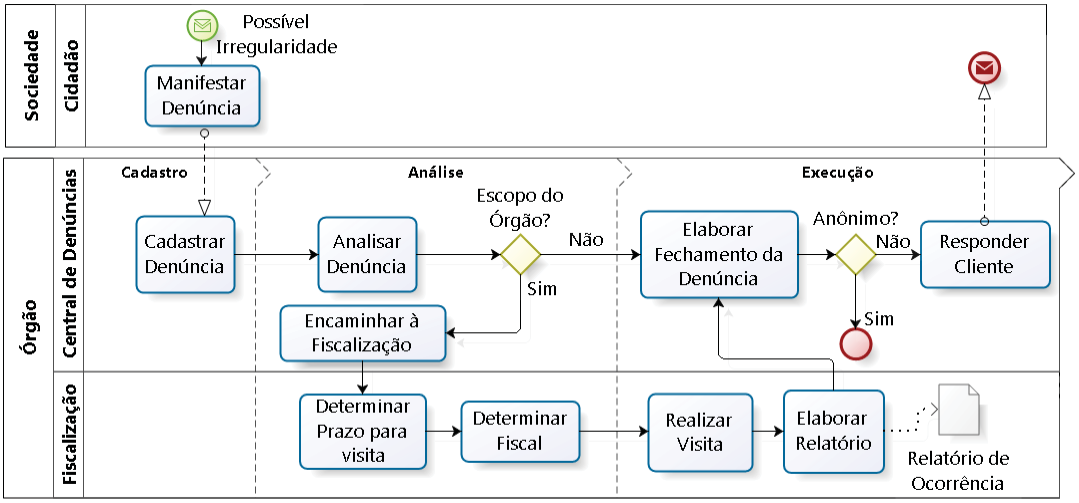


Figura 1. Modelo de Processo de Negócio melhorado.

A melhoria apresentada é a possibilidade da própria Fiscalização ao receber a demanda da denúncia encaminhá-la ao **fiscal,** sem que haja obrigatoriedade de repassar antes à Central de Denúncias para realizar o encaminhamento. Os próprios envolvidos no processo validam a proposta identificando como esta, a melhor alternativa para o modelo.

4. Arquitetura de Sistema

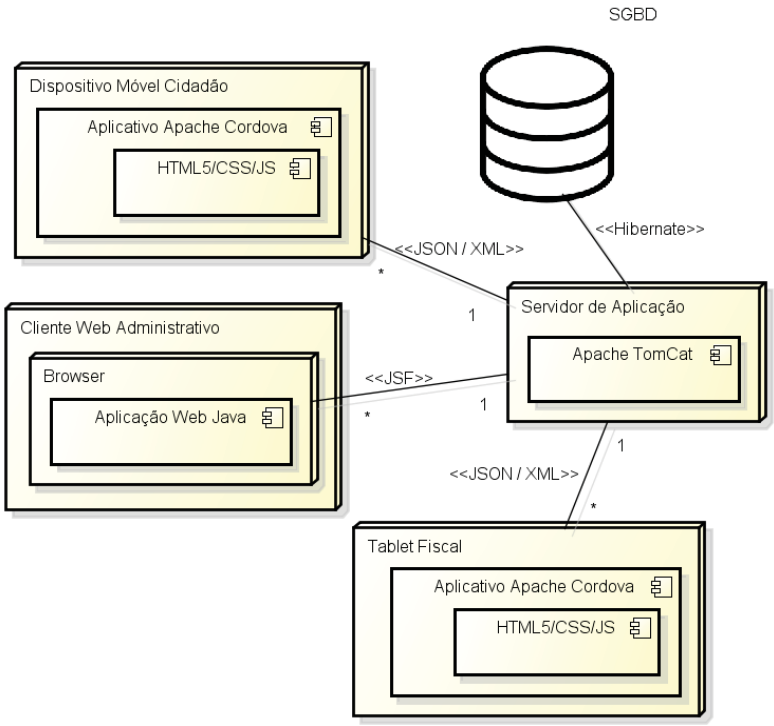


Figura 2. Diagrama de Implantação para o Sistema Multiplataforma para o Controle de Denúncias.

4.1. Perspectiva de Implantação

Quanto à Perspectiva de Implantação (Figura 2), além dos nós, definimos tecnologias subjacentes a cada um destes, assim como também as técnicas de comunicação entre eles. Pensando no compartilhamento do mesmo repositório de dados um dos padrões arquiteturais aplicados é o Cliente-Servidor. Baseado na pluralidade de Sistemas Operacionais dos *smartphones e tablets* indica-se que o aplicativo seja compilado para o executável nativo da maioria dos Sistemas Operacionais *Mobile* existentes através da codificação usando HTML(*HyperText Markup Language*), CSS(*Cascading Style Sheets*), *javascript* e o *framework* Apache Cordova [4].

4.2. Perspectiva Modular

A Perspectiva Modular, para este trabalho, foi dividida em quatro Visões Funcionais: Cidadão, Central de Denúncias, Área Administrativa da Fiscalização e Fiscal. A Visão do Cidadão foi dividida nos submódulos: Registro, Acompanhamento e Histórico. A Visão Fiscal terá o submódulo Relatórios de Visita, e também o submódulo *Backlog* (ou "em espera"), que mostrará as denúncias a serem elucidadas, que foram alocadas àquele fiscal, mas ainda não foram programadas para visita. A Visão Central de Denúncias foi dividida em: Análise, Acompanhamento, Estatísticas e Relatório Geolocalizado. O Relatório Geolocalizado é um componente que poderá orientar a fiscalização do órgão a concentrar esforços em uma determinada área, caso haja maior recorrência de casos em uma região geográfica. A Visão Área Administrativa da Fiscalização terá todos os mesmo submódulos da Central de Denúncias, com exceção do módulo de Análise, porém em acréscimo o submódulo Distribuição para alocação das denúncias aos fiscais.

5. Conclusão e Trabalhos Futuros

Neste trabalho concluímos a modelagem arquitetural, importante etapa no processo de construção de um sistema. Os resultados são animadores. Os modelos estão em processo de aprovação por especialistas do órgão e encontra-se em prototipagem com o objetivo de validação e detalhamento das especificações. Esta construção se prosseguirá nas etapas comuns para desenvolvimento do sistema: projeto detalhado, codificação, teste, implantação e treinamento.

6. Referências

[1] da Silva, C. R. C., Tavares, T. C., & Bicharra, A. C . (2009). Governo Eletrônico em Ambientes Colaborativos Virtuais. *IX Simpósio Brasileiro de Sistemas de Informação*, 122-132.

[2] Model, B. P. (2013). Notation (BPMN) Version 2.0. *Object Management Group specification*.

[3] Uml, O. M. G. (2004). 2.0 Superstructure Specification. *OMG, Needham*.

[4] Cordova, A. (2013). About Apache Cordova. Disponivel em:<http://cordova.apache.org>. *Acessado em: 15 dez. 2014.*

Abordagem de TBM para Automatizar Testes GUI no Contexto de Aplicações Móveis

Silvia Meireles[1], Arilo Dias-Neto[1]

Instituto de Computação (IComp) – Universidade Federal do Amazonas (UFAM)

Av. General Rodrigo Octávio, 6.200, Campus Universitário Senador Arthur Virgílio Filho – Setor Norte – Manaus – CEP 69.077-000 – Manaus – AM – Brasil

{silvia,arilo}@icomp.ufam.edu.br

Resumo. *A crescente demanda de Aplicações Móveis que devem ser desenvolvidas em um curto período de tempo traz consigo a necessidade de se buscar estratégias para reduzir o tempo e/ou custo de desenvolvimento e promover o aumento da qualidade do produto. Isso pode ser feito por meio do teste de software, especificamente por meio da automatização de atividades de teste, onde normalmente são escolhidas atividades repetitivas, como geração e execução de casos de teste. Este artigo descreve a proposta de uma abordagem de TBM para apoiar a automatização de testes GUI em Aplicações Móveis, que engloba a geração do modelo, geração e execução de casos de teste e suporte a atividades de pós-teste.*

1. Introdução

Desde o surgimento do primeiro *smartphone* em 2007, houve uma verdadeira revolução causada pela utilização de dispositivos móveis. Hoje, esses dispositivos são simplesmente fundamentais em nossa vida cotidiana e se torna difícil pensar em realizar algumas tarefas simples, como enviar/receber mensagens, emails e realizar operações bancárias, sem o uso de Aplicações Móveis, também conhecidas como *Apps*.

Apps possuem diversas características não encontradas em outros domínios, tais como: suporte a uma vasta gama de dispositivos, plataformas e versões. Para avaliação da sua qualidade, existem outras variáveis que devem ser consideradas no teste de dispositivos móveis, como, a qualidade da ligação de rede e o próprio movimento do dispositivo podem influenciar o comportamento de uma *App* (WILLIAMSON, 2013).

O teste de software é uma forma de se melhorar a qualidade em sistemas de software (DELAMARO *et al.*, 2007). *Apps*, como qualquer sistema de software, necessitam ser testadas, porém, testar é uma atividade cara, complexa e exige grande esforço (GAO *et al.*, 2014). Esse fato tem despertado o interesse de pesquisadores em buscar alternativas para reduzir seus elevados custos.

A automatização de teste é uma estratégia comumente utilizada para reduzir o custo e/ou tempo de teste e também permite aumentar sua eficiência e confiabilidade (MARIANI *et al.*, 2012). Apesar de inúmeros trabalhos demonstrarem os benefícios da automatização de teste, o teste manual é a abordagem mais comum em aplicações móveis na indústria (WILLIAMSON, 2013). Por outro lado, diversos trabalhos têm pesquisado aspectos individuais da automatização de teste em *Apps*, como: geração de modelos para teste (MOREIRA e PAIVA, 2014); geração de oráculos de teste (ZAEEM

et al., 2014), geração de dados de teste (LI *et al.*, 2014; ANAND *et al.*, 2012), teste baseado em reconhecimento de imagens (CHANG *et al*., 2010; WU e LIU, 2012), dentre outros. Embora diversos trabalhos abordem aspectos relacionados à automatização de teste, normalmente não há integração de técnicas/abordagens para englobar vários aspectos da geração/execução de teste.

Zaeem *et al.* (2014) alertam para a necessidade de se desenvolver ferramentas de teste automatizado para apoiar o desenvolvimento de aplicativos móveis. Wu e Liu (2012) ressaltam que o Teste Baseado em Modelos (TBM) é uma das estratégias mais populares para dar suporte à geração e execução de casos de teste.

Neste trabalho, pretende-se instanciar uma abordagem de TBM que suporte a geração e execução automática de casos de teste GUI para aplicações móveis. Isso será feito a partir de técnicas/abordagens existentes para geração automática de dados e oráculos de teste. Também faz parte do escopo deste trabalho integrar ferramentas em cada uma das atividades englobadas neste trabalho, com o objetivo de maximizar o uso dessas ferramentas.

2. Trabalhos relacionados

Moreira e Paiva (2014) descrevem uma abordagem de TBM para aplicações GUI, denominada *Pattern Based GUI Testing* (PBGT) que visa sistematizar e automatizar o processo de Testes GUI por meio de Padrões de Teste GUI. PBGT possui apoio ferramental, fornecendo um ambiente integrado de modelagem e teste, que permite gerar automaticamente casos de teste a partir de modelos e executá-los por meio de uma GUI.

Li *et al.* (2014) propõem o *framework ADAutomation* para teste automatizado GUI que utiliza o Diagrama de Atividades da UML. Esse framework modela o comportamento do usuário, gera casos de teste GUI, oferece análise de pós-teste e depuração.

Zaeem *et al.* (2014) propõem uma abordagem extensível para gerar oráculos de teste que permite a geração de sequências de teste que aproveitam esses oráculos. Nessa abordagem que possui suporte ferramental, *features* da aplicação são definidas e armazenadas em uma biblioteca. Ao testar uma *App*, *features* da biblioteca são instanciadas, e por meio das suas definições são gerados casos de teste com oráculos de teste para testar exaustivamente cada *feature*.

Amalfitano *et al.* (2014) apresentam uma técnica automática para teste GUI em aplicações Android, denominada *MobiGUITAR*. Nessa técnica, os estados dos *widgets* são extraídos em tempo de execução e são usados para gerar uma máquina de estado escalável em conjunto com critérios de cobertura de Teste Baseado em Evento, que permite gerar dados de teste automaticamente.

Anand *et al.* (2012) apresentam uma técnica, denominada *Contest*, para gerar eventos de entrada para *Apps*. Essa técnica realiza a execução *Concolic*, que é uma evolução da execução simbólica para gerar sequências de eventos sistematicamente.

3. Abordagem proposta

Neste trabalho, pretende-se instanciar uma abordagem automatizada para TBM no contexto de aplicações móveis que contemple a geração do modelo, geração e execução de casos de teste e apoie a comparação de resultados do teste. A primeira atividade desta

abordagem é a geração do modelo (estrutural ou comportamental) da *App* testada. Esse modelo pode ser construído a partir dos seguintes artefatos de software:

- Código da aplicação, como demonstrado em Amalfitano *et al.* (2014) que é uma extensão ao trabalho de Nguyen *et al.* (2013). Neste, a ferramenta *Guitar* explora automaticamente a aplicação testada capturando todas as possíveis interações de eventos GUI, e ao final deste processo gera o modelo estrutural da mesma, conhecida como Árvore GUI;
- Diagramas da UML, como o Diagrama de Atividades, que é utilizado para modelar o comportamento do usuário em (Li *et al.*, 2014);
- Modelos GUI descritos em outras linguagens, como o modelo em linguagem PARADIGM proposto em (Moreira e Paiva, 2014), que descrevem padrões de *widgets*.

Com base no modelo gerado, serão utilizadas técnicas para gerar um conjunto de casos de teste por meio da geração de dados e oráculos de teste. Para gerar as entradas do teste, podem ser utilizadas as seguintes abordagens:

- Execução simbólica, que permite gerar sequências de eventos, como proposta em (Anand *et al.*, 2012) que utiliza a execução *Concolic*;
- Mapeamento de Padrões GUI que gera caminhos entre elementos iniciais e finais dentro do modelo descrito em (MOREIRA e PAIVA, 2014).

Outra atividade importante é a geração de oráculos de teste para determinar se a execução dos testes está correta. Para essa etapa, pode-se partir do trabalho Zaeem *et al.* (2014), adaptando-se a abordagem que utiliza a definição de *features* de interação de usuários que são incrementalmente adicionadas.

Uma vez geradas as entradas e os oráculos de teste, é possível executar os casos de teste. Para isso, deve-se escolher uma plataforma sob a qual os casos de teste serão executados. Nessa etapa, pode-se optar pela plataforma Android, visto que a mesma é a plataforma mais popular de *Apps* e é utilizada em diversos trabalhos (ZAEEM *et al.*, 2014; NGUYEN *et al.*, 2013).

Ao final da execução de cada caso de teste, o resultado obtido é comparado com o resultado definido pelo respectivo oráculo. Nos casos em que o resultado obtido é igual ao previsto pelo oráculo, os testes são aprovados, caso contrário os incidentes de teste devem ser reportados.

A abordagem proposta será desenvolvida a partir de técnicas/abordagens existentes, pois o objetivo é reutilizá-las e adaptá-las quando necessário. Esta abordagem contemplará as principais atividades do processo automatição de teste em aplicações móveis com o propósito de integrar ferramentas utilizadas neste processo.

4. Resultados esperados

Como resultado espera-se automatizar testes GUI em aplicações móveis e com isso reduzir o ciclo de desenvolvimento da automatização de testes, por meio da redução de tempo/custo nas atividades de uma abordagem de TBM para *Apps*, como geração de modelo e geração/execução dos casos de teste GUI.

Dentro deste trabalho, está definida a avaliação experimental de ferramentas de automatização de testes GUI com o propósito de identificá-las e/ou integrá-las com o propósito de maximizem seu uso.

Agradecimentos

Os autores agradecem à FAPEAM e INDT pelo apoio financeiro para a realização desta pesquisa.

Referências

Amalfitano, D.; Fasolino, A.; Tramontana, P.; Ta, B.; Memon, A. (2014) "MobiGUITAR - A Tool for Automated Model-Based Testing of Mobile Apps," Software, IEEE , vol.PP, no.99, pp.1,1.

Anand, Saswat; Naik Mayur; Harrold, Mary Jean; Yang, Hongseok (2012) "Automated concolic testing of smartphone apps". In: Proceedings of the ACM SIGSOFT.

Chang, Tsung-Hsiang; Yeh, Tom; Miller, Robert C. (2010) "GUI testing using computer vision". In: Proceedings of the Conference on Human Factors in Computing Systems.

Delamaro, M. E.; Maldonado, J. C.; Jino, M. (2007) "Introdução ao Teste de Software". Rio de Janeiro: Elsevier.

Gao, J.; Wei-Tek Tsai; Paul, R.; Xiaoying Bai; Uehara, T., (2014) "Mobile Testing-as-a-Service (MTaaS) -- Infrastructures, Issues, Solutions and Needs," , In: 15th International Symposium on High-Assurance Systems Engineering (HASE), vol., no., pp.158,167, 9-11.

Li, Ang; Qin, Zishan; Chen, Mingsong; Liu, Jing (2014) "ADAutomation: An Activity Diagram Based Automated GUI Testing Framework for Smartphone Applications". In: 8th International Conference on Software Security and Reliability (SERE), pp.68-77.

Mariani, Leonardo; Pezzè, Mauro; Riganelli, Oliviero; Santoro, Mauro (2012) "Auto BlackTest: Automatic Black-Box Testing of Interactive Applications". In: 34th International Conference on Software Engenieering.

Moreira, Rodrigo M.L.; Paiva, Ana C.R. (2014) "PBGTTool: An Integrated Modeling and Testing Environment for Pattern-Based GUI Testing". In: 29th IEEE/ACM International Conference on Automated Software Engineering (ASE).

Nguyen, Bao; Robbins, Bryan; Banerjee, Ishan; Memon, Atif. (2013) "Guitar: an innovative tool for automated testing of gui-driven software".Automated Software Engineering, pp. 1-41, Springer US.

Zaeem, Razieh N.; Prasad, Mukul R.; Khurshid, Sarfraz (2014) "Automated generation of oracles for testing user-interaction features of mobile apps". In: 7th International Conference on Software Testing, Verication and Validation, pp.183-192.

Williamson, L. (2013) "A mobile application development primer: A guide for enterprise teams working on mobile application projects". IBM Software Thought Leadership White Paper.

Wu, Yumei; Liu, Zhifang (2012) "A Model Based Testing Approach for Mobile Device". In: International Conference on Industrial Control and Electronics Engineering (ICICEE), pp.1885-1888.

Discovery and Usage of Computing Devices in IoT Environments

Willian Lunardi[1], **Sabrina Marczak**[1], **Leonardo Amaral**[1], **Fabiano Hessel**[1]

[1]Faculdade de Informática – Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)

{willian.lunardi, leonardo.amaral}@acad.pucrs.br,

{sabrina.marczak, fabiano.hessel}@pucrs.br

Abstract. *During the past few years, with the fast development and proliferation of the Internet of Things (IoT) as well as with the growing number of active computing devices in IoT environments around the world, many application areas have started to exploit this new computing paradigm. Consequently, a mechanism to deal with different devices has become necessary. Middleware systems solutions for IoT have been developed in both research and industrial environments to supply this need. However, discover and usage of computing devices remain a critical challenge due to the large amount of devices available and the lack of intuitive mechanisms to deal with them. This paper presents a brief and preliminary discussion on the alternatives reported in literature to address this issue. Our long-term goal is to propose a framework to help programmers of IoT applications to select and to interact with middleware devices.*

1. Introduction

The term Internet of Things (IoT) was coined in 1998 [Kevin 2009] and defined as the computing paradigm that allows people and things to be connected Anytime, Anyplace, with Anything and Anyone, ideally using Any path/network and Any service [Guillemin et al. 2009]. In this sense, there are current market statistics and predictions that demonstrate a rapid growth in computing device deployments related to IoT environments. By 2020, it is estimated that there will be 50 to 100 billion IoT devices connected to the Internet [Sundmaeker et al. 2010]. These statistics and facts imply that we will be faced with a vast amount of IoT devices, which, when properly used, will add more value to the environment.

A inherited issue in this new setting is information overload, i.e. users face vast and distributed information sources, and have difficulty in selecting those that satisfy their needs and interests. In this sense, the biggest challenge and time-consuming task is to select and use the appropriate devices when there is a large amount of available devices to choose from and, often, with heterogeneous characteristics.

Middleware systems solutions for IoT have been developed to supply this need. However, discovery and usage of middleware devices remain a critical challenge [Perera et al. 2012]. End users, such as programmers that are in charge of selecting (aggregating) heterogeneous devices in order to contextualize virtual environments and apply operations among them, are not aware of domain modeling and middleware specification patterns (to define what and how they want to interact with things/devices). Besides, programmers may not have enough knowledge to perform such task without devoting time to

understand the middleware patterns or to engage others who are responsible for modeling the domain of devices.

By analyzing other IoT middleware systems (e.g., [GSN Team], [Digital Enterprise Research Institute], [Amaral et al. 2015]), we found that they do not provide intuitive search methods and also do not focus on ease of use of devices by programmers. Trying to fill this gap, our long-term goal is to propose a framework that will embed mechanisms to a middleware solution to solve these issues. Our first step in our research journey is to learn more about how to suggest devices based on users needs. This paper presents a brief overview about the IoT paradigm to contextualize the motivation to our work, and describes two device recommendation approaches we have found in our preliminary literature search that might be applicable to our solution.

2. Internet of Things and Context of Things

During the past decade, the IoT has gained significant attention by academia and by industry. It promises to create a world where all objects (also called 'smart objects') around us are connected to the Internet and communicate with each other with minimum human intervention [Le-Phuoc et al. 2009].

On the other hand, there is no any standard definition for IoT [Perera et al. 2014]. A commonly used definition is that IoT allows people and things to be connected Anytime, Anyplace, with Anything and Anyone, ideally using Any path/network and Any service [Guillemin et al. 2009].

In this sense, IoT ecosystems are based on a layered architecture style and use this view to abstract the integration of objects and to provide services solutions to applications [Jing et al. 2014]. In IoT, high-level system layers as the application layer are composed of IoT applications and middleware system, which is a software layer interposed between the infrastructure of devices and applications, and is responsible for providing services according to devices functionality [Atzori et al. 2010]. Many of the system architectures proposed for IoT middleware comply with Service-Oriented Architecture (SOA). This approach allows each device to offer its functionality as standard services. Therefore, IoT middleware systems have evolved from hiding network details to applications into more sophisticated systems to handle many important requirements, providing support for heterogeneity and interoperability of devices, data management, security, etc.

In most instances, the middleware device connection is followed by a contextualization process that aims to acquires and stored its characteristics. Context is considered any information that can be used to characterize the situation of an entity. Besides, context information about IoT devices needs to be acquired and stored with annotations that will make easy to retrieve it later. Abowd and Mynatt [Abowd and Mynatt 2000] identified the five Ws (Who, What, Where, When, and Why) as the minimum annotations that are necessary to understand context. Studies like this one imply that IoT middleware requires a mechanism for the acquisition of context characteristics of devices in order to provide features such devices discovery, management, and others.

3. Recommenders Systems

As mentioned earlier, device discovery remains a critical challenge. To move towards filling this gap, we learned that context information can be used to promote devices discovery

in IoT middleware solutions. Also, that a software system that uses such contextual information to recommend devices based on users needs can provide a more intuitive manner to the user discover and use devices.

A system that recommends things by producing customized recommendations as output or has the effect of guiding the user in a personalized way to interesting or useful objects in a large space of possible options is named a recommendation system [Burke 2002].

Adomavicius and Tuzhilin [Adomavicius and Tuzhilin 2011] elicited different approaches to use contextual information in the recommendation process. These can be broadly categorized into two groups: (1) recommendation via *querying and search*, and (2) recommendation via *preference elicitation and estimation*.

1. The *querying and search* approach uses information to query or search a certain repository of resources and present the best matching resources to the user.
2. The *preference elicitation and estimation* approach attempts to model and learn user preferences, i.e., by observing the interactions of this and other users with the systems or by obtaining preference feedback from the user on various previously recommended items, and uses these preferences to make its recommendations.

Search-based recommender systems usually create an index of objects and use this index to respond to queries from users. An index is a data structure that makes the system efficient to retrieve objects given the value of one or more elements of the objects. Queries are evaluated by processing the index in order to identify similarities, which are then returned to the user. These systems also have procedures to analyze variations in documents and queries. The verification of these variations can improve the searching process. These procedures can play functions such as 'synonym check' which aims to search for words with equivalent meanings, 'stopwords check' which aims to identify keywords that are not considered relevant, and so on.

4. Discussion and Research Plan

With the constant growth of the IoT and the provision of a huge number of IoT devices in the near future, it is unfeasible that application programmers manually look for desired devices without a standardized system support.

A IoT context-based recommender system can provide improvements to the IoT application development process that is composed of the process of finding and using middleware devices, especially for users that do not know the domain and/or the middleware configuration patterns.

Given our previously stated problem, facts, and definitions, with our preliminary review of literature, we found that we can recommend devices and guide the user through search-based recommender systems. In this case, the user will not be aware of the content and structure of the system data to ask a precise question. Therefore, through a search we can provide a ranked list of items that are strongly related to the search terms that the user entered, even if they do not exactly match.

The next step is to conduct a systematic literature review of search engines and search-based recommender systems to conclude which techniques are most appropriate to

compose our framework solution. Once we finish this more formal and organized review, we will develop a software prototype that implements our framework solution composed of the chosen techniques. The framework will be coupled on SOA-based middleware previously developed by our research group [Amaral et al. 2015]. Next, we will conduct a case study with real data from a health care system to evaluate the framework.

References

Abowd, G. D. and Mynatt, E. D. (2000). Charting past, present, and future research in ubiquitous computing. *ACM Trans. Comput.-Hum. Interact.*, 7(1):29–58.

Adomavicius, G. and Tuzhilin, A. (2011). Context-aware recommender systems. In *Recommender systems handbook*, pages 217–253. Springer.

Amaral, L., Tiburski, R., Matos, E., and Hessel, F. (2015). Cooperative middleware platform as a service for internet of things applications. In *Proc. of the ACM Symposium on Applied Computing (to be published)*, SAC '15, New York, NY, USA. ACM.

Atzori, L., Iera, A., and Morabito, G. (2010). The internet of things: A survey. *Computer Networks*, 54(15):2787 – 2805.

Burke, R. (2002). Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370.

Digital Enterprise Research Institute. Linked sensor middleware (lsm). `http://lsm.deri.ie/`. Accessed: 2015-03-26.

GSN Team. Global Sensor Network. `http://sourceforge.net/apps/trac/gsn/`. Accessed: 2015-03-11.

Guillemin, P., Friess, P., et al. (2009). Internet of things strategic research roadmap. *The Cluster of European Research Projects, Tech. Rep., September*.

Jing, Q., Vasilakos, A., Wan, J., Lu, J., and Qiu, D. (2014). Security of the internet of things: perspectives and challenges. *Wireless Networks*, 20(8):2481–2501.

Kevin, A. (2009). That internet of things thing, in the real world things matter more than ideas. *RFID Journal*, 22.

Le-Phuoc, D., Polleres, A., Hauswirth, M., Tummarello, G., and Morbidoni, C. (2009). Rapid prototyping of semantic mash-ups through semantic web pipes. In *Proceedings of the 18th international conference on World wide web*, pages 581–590. ACM.

Perera, C., Zaslavsky, A., Christen, P., and Georgakopoulos, D. (2012). Ca4iot: Context awareness for internet of things. In *Proc. Int'l Conf. on Green Computing and Communications*, pages 775–782. IEEE.

Perera, C., Zaslavsky, A., Christen, P., and Georgakopoulos, D. (2014). Context aware computing for the internet of things: A survey. *Communications Surveys Tutorials, IEEE*, 16(1):414–454.

Sundmaeker, H., Guillemin, P., Friess, P., and Woelfflé, S. (2010). Vision and challenges for realising the internet of things.

Análise da adoção de processo de medição no desenvolvimento ágil de software

Luis Paulo Correa[1], Raquel Aparecida Pegoraro [1]

[1]Departamento de Ciência da Computação – Universidade Federal da Fronteira Sul (UFFS) – Chapecó, SC – Brasil

correaluisp@gmail.com, raquel.pegoraro@uffs.edu.br

Resumo. *Motivados pelo crescimento da adoção dos métodos ágeis e pela necessidade de realizar a análise do desempenho desses projetos através da análise de métricas e da implantação de processos de medição, bem como pela carência de estudos que tratam sobre esse assunto, esta pesquisa tem como objetivos: (a) Avaliar o impacto da adoção de um processo de medição na análise do desempenho numa empresa que utiliza métodos ágeis na gestão de seus projetos; (b) Identificar os fatores comportamentais ou organizacionais que interfiram positivamente ou negativamente na implantação do processo de medição. Ao final deste projeto de pesquisa espera-se alcançar os seguintes resultados: (a) entender quais fatores interferem na implantação do processo de medição com métodos ágeis; (b) entender se a adoção de um processo de medição auxilia na análise de medição avaliando os fatores qualidade, produtividade e evolução do projeto, e qual a relação de impacto entre esses fatores.*

Abstract. *Motivated by developing of adoption of agile methods and the necessity to do the performance analysis projects through analyzing metrics and the implementation of measurement processes, as well as the lack of studies that deal with this issue, this research aims to: (a) Evaluate the impact of the measuring process adoption in the performance analysis on a company which uses the agile methods in the management of its project; b) Identify the behavioral or organizational factors that interfere positively or negatively in the implementation in the measuring process. By the end of this research we expected to achieve the following results: (a) Understand which factors influence the implementation of the measurement process with agile methods; (b) Understand if the adoption of measurement process helps in measuring analysis evaluating the quality factors, productivity and evolution of the project, and what the impact of relationship between these factors.*

Palavras-chaves: Métricas de software. Processo de medição. Métodos ágeis. Pesquisa exploratória. Projeto de experimentos.

1. Introdução

Os métodos ágeis tem chamado atenção das empresas de software devido aos inúmeros relatos de sucesso, porém, segundo Dingsoyr et al., (2012), apesar do crescimento ascendente de sua adoção, muito trabalho ainda tem de ser empreendido para tornar coerente o discurso atual sobre a agilidade. Mikulenas e Butleris (2010) afirmam que as pesquisas da área se concentram na apresentação de histórias de sucesso ou de lições aprendidas por organizações que adotaram métodos ágeis para projetos específicos, havendo uma falta de pesquisas de avaliação de adequação dos métodos ágeis, considerando diversas características ambientais em empresas de software, entre eles adoção de métricas de software.

As métricas são essenciais para as empresas desenvolvedoras de software, pois ajudam na medição da qualidade, na estimava dos recursos necessários e dos custos, no planejamento e controle do progresso de desenvolvimento de software (Mishra, Kumar e Kumar, 2009). Poonacha e Bhattacharya (2012) argumentam que muitas organizações adotam métodos ágeis sem entender quais os fatores devem ser medidos e controlados.

O *Standish Group* publicou em 2011 o relatório *Chaos Manisfesto* que compara o desempenho entre projetos ágeis e tradicionais de software e apresenta fatores motivadores para adoção de métodos ágeis. Nos projetos tradicionais (modelo cascata) 14% tiveram sucesso, 57% foram contestados e 29% falharam, sendo que nos projetos ágeis 42% tiveram sucesso, 49% foram contestados e 9% falharam (Standish Group, 2012). Os dados apresentados na pesquisa mostram que os projetos realizados a partir da abordagem ágil tiveram mais sucesso que os tradicionais, porém a taxa de projetos que falharam ou foram contestados ainda é alta, o que demonstra sérios problemas de gestão e reforça a importância de um monitoramento e controle eficazes nos projetos desta natureza.

A revisão sistemática publicada por Dyba e Dingsoyr (2008) a qual faz um vasto levantamento sobre estudos empíricos utilizando métodos ágeis, não cita nenhum estudo sobre métricas de software, processo de medição ou indicadores de desempenho, constatando que nos primeiros anos após o advento do Manifesto Ágil os estudos não trataram destes temas. Após este período alguns estudos foram publicados sobre o assunto, porém predominando estudos de caso com relatos de experiência sobre a adoção de métricas em métodos ágeis (Petersen e Wohlin, 2011) (Talby e Dubinsky, 2009) (Green, 2011) (Middleton e Joyce, 2012). Os autores Dingsoyr et al., (2012) avaliaram os primeiros 10 anos dos métodos ágeis e afirmam que entre os assuntos que ainda precisam ser evoluídos nos métodos ágeis está o gerenciamento desses projetos, especialmente sobre planejamento, controle, avaliação do desempenho e estimativas. Neste contexto, este projeto de pesquisa possui os seguintes objetivos:

- Avaliar o impacto da adoção de um processo de medição na análise do desempenho numa empresa que utiliza métodos ágeis na gestão de seus projetos;
- Identificar os fatores comportamentais ou organizacionais que interfiram positivamente ou negativamente na implantação do processo de medição.

Este artigo está estruturado da seguinte forma: na sessão 2 são apresentados os procedimentos metodológicos que serão utilizados e na sessão 3 são apresentados os resultados esperados para esta pesquisa.

2. Procedimentos Metodológicos

A metodologia que será utilizada na pesquisa consistirá em:

a) Realização de um estudo de caso numa empresa de grande porte que possui um setor de desenvolvimento de software para atender as demandas internas de sistemas de informação e utiliza métodos ágeis na gestão de seus projetos, sendo subdivido em 2 fases de pesquisa. Na primeira fase será utilizada a estratégia de pesquisa exploratória (Gil, 2010) e serão acompanhadas várias iterações de um projeto com o objetivo de conhecer a realidade vivenciada pela empresa quanto ao monitoramento dos seus projetos, neste momento sem interferência do pesquisador. Posteriormente será utilizada a estratégia de pesquisa-ação (Gil, 2010), nesta fase será implantado um processo de medição seguido às recomendações de Pegoraro (2014) que apresenta recomendações de como implantar um processo de medição para projetos ágeis de software, e de Softex (2011) que define resultados a serem esperados de um processo de medição maduro. Para definição das métricas será utilizado o método GQM (*Goals Questions Metrics)* proposto por Basili et al. (1996), abordagem muito utilizada para definição de métricas na engenharia de software. Nesta fase da pesquisa serão utilizadas as técnicas de coleta de dados de entrevistas e grupos focados.

b) Para a avaliação do impacto da adoção de um processo de medição na análise de desempenho, será realizado um estudo quantitativo através técnica de análise e projetos de experimentos. Os fatores a serem investigados serão: (a) qualidade; (b) produtividade; e (c) evolução do projeto. Foram definidos esses fatores por serem aspectos críticos da gestão ágil de projetos. A forma de controle desses fatores será através do monitoramento das métricas e os dados serão coletados durante as fases de pesquisa exploratória e pesquisa-ação; após, os resultados serão submetidos à análise de variância (ANOVA) para identificar quais fatores tiveram interação significativa após a adoção do processo de medição. Para a condução do projeto de experimentos será seguindo as recomendações de (Werkema e Aguiar, 1996) e (Montgomery, 1997).

3. Resultados esperados

Ao final deste projeto de pesquisa espera-se alcançar os seguintes resultados: (a) entender quais fatores interferem na implantação do processo de medição nos métodos ágeis; (b) entender se a adoção de um processo de medição auxilia na análise de desempenho avaliando os fatores qualidade, produtividade e evolução do projeto, e qual a relação de impacto entre esses fatores.

4. Referências

BIOLCHINI, J. C. de A. et al. Scientific research ontology to support systematic review in software engineering. Advanced Engineering Informatics, v. 21, n. 2, p. 133-151, 2007.

BASILI, V. et al. Goal question metric approach. Encyclopedia of software engineering. In: Encyclopedia of Software Engineering. John Wiley and Sons, pp. 528–532, 1996.

DINGSOYR, T. et al. A decade of agile methodologies: Towards explaining agile software development. Journal of Systems and Software, v. 85, n. 6, p. 1213–1221, jun. 2012.

DYBA, T.; DINGSOYR, T. Empirical studies of agile software development: A systematic review. Information and Software Technology, Amsterdam, v. 50, n. 9-10, p.833-859, ago. 2008.

GIL, A. C. Como elaborar projetos de pesquisa. 5. Ed. São Paulo: Atlas, 2010.

GREEN,P. Measuring the Impact of Scrum on Product Development at Adobe Systems. System Sciences (HICSS), 2011 44th Hawaii International Conference on. p. 1-10.

MIDDLETON, P.; JOYCE, D. Lean Software Management: BBC Worldwide Case Study. Engineering Management, IEEE Transactions on. v. 59, n.1, p.20-32, 2012.

MIKULENAS, G.; BUTLERIS, R. An approach for constructing evaluation model of suitability assessment of agile methods using analytic hierarchy process. Electronics and Electrical Engineering, v. 10, n. 106, p. 99-104, 2010.

MISHRA, D.; BALCIOGLU, E.; MISHRA, A. Measuring Project and Quality aspects in Agile Software Development. TTEM-Technics Technologies Education Management, v. 7, n. 1, 2012.

MONTGOMERY, D. C. Design and Analysis of Experiments. 5. ed. New York: John Wiley & Sons, 1997.

SOFTEX. Melhoria de Processo do Software Brasileiro: Guia Geral. Campinas: SOFTEX, 2011.

STANDISH Group. Chaos Manifesto. 2012. The Standish Group International. Relatório. Disponível em: <https://secure.standishgroup.com/reports/reports.php> Acesso em: 01 mar. 2015.

POONACHA, K. M.; BHATTACHARYA, S. Towards a Framework for Assessing Agility. System Science (HICSS), In: 45TH HAWAII INTERNATIONAL CONFERENCE, 2012. Proceedings… p. 5329-5338, 2012.

PEGORARO, R. A. Métricas de avaliação para abordagens ágeis em projetos de software. 2014. Tese. Universidade de Federal do Rio Grande do Sul, 2014.

PETERSEN, K. WOHLIN, C. Measuring the flow in lean software development. Software: Practice and Experience. v. 41. n. 9, p. 975-996, 2011.

TALBY, D.; DUBINSKY,Y. Governance of an agile software project. Software Development Governance, 2009. SDG '09. ICSE Workshop on. p.40-45.

WERKEMA, M. C. C.; AGUIAR, S. Planejamento e Análise de Experimentos: como identificar e avaliar as principais variáveis influentes em um processo. Belo Horizonte: Fundação Christiano Ottoni, 1996.

On the Transformation to Agile in a Large-Complex Globally Distributed Company: A Research Plan to Define Guidelines

Greice Roman, Sabrina Marczak, Alessandra Dutra

[1]Faculdade de Informática – Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Av. Ipiranga, 6681 – Partenon – 90.619-900 – Porto Alegre – RS – Brazil

greice.roman@acad.pucrs.br, {sabrina.marczak, alessandra.dutra}@pucrs.br

Abstract. *The transformation to agile is not a simple process and although there is vast literature on the topic, there is still no consolidated body of knowledge on how to proceed when this transformation happens in large-complex globally distributed companies. This paper presents the research plan to follow the transformation into agile of a large-complex distributed IT organization aiming to serve as an exploratory case study for our long-term goal of proposing a set of guidelines to guide the transformation in such type of company.*

1. Introduction

The Agile Manifesto [Beck and colleagues 2001] was written in February 2001. It offers new values to motivate software companies to deliver high-quality products faster and produce satisfied customers. Since then, companies are discussing whether to become agile and how to go about transforming themselves to achieve such 'agility'. The transformation process involves more than deciding on which agile method to adopt. It refers to making changes in such a way that the company and its projects will 'become' agile.

Academia has been supporting industry to go through the transformation process for as long as the agile philosophy has been defined. However, achieving success in large-scale companies as reported in [Fry and Greene 2007]), for example, is a complex process and brings numerous challenges to organizations [Korhonen 2013]. For instance, how much can requirements keep changing when they cross hundreds of applications at a time? [Dingsøyr and Moe 2014] presents a research agenda on the topic showing that there are still several open questions. Given the large number of large companies migrating to agile, there is a need for a consolidated and more extensive body of knowledge.

This paper presents the research plan to follow the transformation into agile of a large-complex distributed IT organization aiming to serve as an exploratory case study for our long-term goal of proposing a set of guidelines to guide the agile transformation process in large-complex globally distributed software companies. We will contribute to furthering the body of knowledge on the topic for this type of company.

2. Agile Transformation

The agile transformation process (ATP) has been encouraged to remedy inherent problems of traditional software development [Gandomani et al. 2014] and is defined as *the process of leaving the traditional way to development software and adopting the agile philosophy, tools, and principles* [Ranganath 2011]. A true ATP must focus on 'being' agile rather than 'doing' agile. This is the main reason that makes ATP more difficult than expected.

[Gandomani et al. 2014] identified a set of categories of an ATP, as follows: prerequisites to become agile, training on methods and what it is about, facilitators (people who will guide the process), transition framework (a stepwise view on how to do it), managing the transition, assessment of progress, reasons for aiming to agility, coaching the transition, technical issues, human aspects-related issues, customer-related issues, selection of pilot projects for the transition, and agile method selection. [Fontana et al. 2015], on the other hand, identified the following categories: practices to become agile, team composition and behavior, deliveries (evolution from traditional to continuous delivery), requirements (transition from traditional requirements elicitation to use stories), product (practices to improve the software product), and customer relationship.

In large-scale agile transformation, the key challenges seem to be managing a large number of agile teams, dividing work among those teams, achieving the system-wide properties of the software, and guaranteeing the simultaneous releases of cross-cutting features [Ganesh and Thangasamy 2012]. In their study, [Dingsøyr and Moe 2014] identified four principles to be observed in large-scale agile development, namely: *architecture* - figuring out how work is coordinated; *inter-team coordination* - creating effective knowledge networks is essential due to the knowledge-intensive nature of software development; *portfolio management* - providing continuous feedback from the portfolio to project levels enables the teams and project members to take decisions that are consistent with the goals of the large-scale agile portfolio, and *scaling* - describing the context for agility and scale is essential for understanding how to improve agility in large-scale agile.

The discussion becomes even more interesting when a large company is physically distributed and develops complex-interrelated applications.

3. Our Long-Term Research Plan and Case Study Design

To achieve our goal we will follow a qualitative approach organized in four major phases (see Figure 1). Phase 1, named *Foundation*, aims to build the foundation for our two-year long investigation and is organized in two major sub-phases. The *Literature Review* sub-phase aims to review definitions for related concepts such as agile transformation, large-scale development, and development of complex applications, and define the concepts we will adopt. It will also serve the purpose of identifying related work. We expect to identify an initial list of aspects that have to be considered when going through the referred transformation (Jan to May'15). The *Benchmarking* sub-phase aims to provide us with deeper knowledge about how large-scale complex globally distributed companies have gone through the ATP themselves. We will visit companies in Europe who have gone through this process and discuss the aspects identified during the Literature Review with them. We plan two months of work including visits and data analysis (Jun-Jul'15).

Phase 2, named *Exploration*, composed of a case study, aims to further the knowledge acquired in Phase 1 by observing the identified aspects in a more comprehensive manner (Aug-Dec'15). Section 3.1 presents the case study design in more details. Phase 3, named *Solution*, aims to have our guidelines for agile transformation in large-scale complex globally distributed companies defined (version alpha). We will systematically organize the insights from the prior phases in the format of policies or procedures that will indicate a course of actions to be taken for each of the aspects related to the transformation when a company wants to become agile (period: Jan-Apr'16).

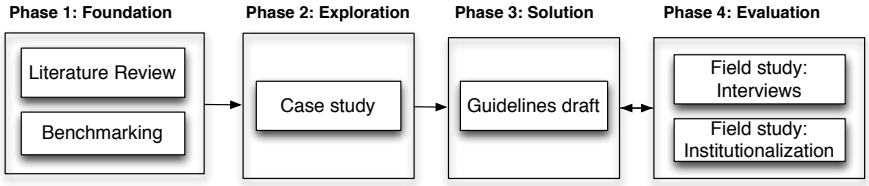


Figure 1. Proposed research design

In Phase 4, named *Evaluation*, we will evaluate whether our proposed guidelines is fit to help large-scale complex globally distributed companies to become agile. We will conduct a field study based on interviews with professionals who have been involved with agile transformation aiming to have them pointing out whether each of the proposed guidelines are proper to the aspect they mean to address and how they could be improved (May-Jul'16). Once we compile the provided feedback, we will generate a new version of the guidelines document (version beta)–as part of Phase 3 (Aug'16), and then consult a new set of professionals to evaluate this version–as part of Phase 4 again (Set-Dec'16). This time we will interview professionals who are currently involved in the transformation process and that would be willing to consider adopting our guidelines. We will invite them to select a sample to try in their companies and later ask them about their perception of the results. There is a risk that guidelines will not be institutionalized given the short time, however, we still think that having a preliminary try to observe changes and results is valuable to have a more refined feedback on how fit is our set of guidelines.

3.1. Case Study Background and Design

The case study will be investigate in a large IT multinational company chosen by convenience. The company, named ORG (fictitious name), has started its agile transformation in Jan'15. ORG's IT department develops software products to support the organization' business processes. Demands to develop or to update these products come from the several business departments distributed around the world. The IT department is organized by business area. Each IT team attends a business area only and is composed of the following functions: project management, business and requirements analysts, developers, and architects. The Test team is a separate organizational unit and has its members allocated per project by business area as requested by IT teams. IT personnel are distributed among the headquarters' office located in the US and also in Brazil, India, and Malaysia.

Projects are defined once a year as part of the IT roadmap plan. Once approved, projects are assigned to project managers who allocate their teams and start a discussion with business representatives acting as project owner proxies of what features should be developed first. A Sprint plan is then approved and development starts. A project backlog is kept and rediscussed each iteration until the scope is finished. Each project might tackle business requests that might cross software applications, presenting the organization with the challenge of having to manage the complexity of large systems interlocks.

Each business IT team is free to organize itself as it wants as long as the team respects the worldwide guidelines defined by the IT board, such as adopt the tools to support software development defined by the organization and follow agile practices proposed by Scrum and XP only. Managers anticipate that this 'freedom' will increase the complexity

of coordinating software development cross business areas and system interlocks.

We will observe four projects selected as pilots by the organization for the ATP. These projects are receiving training and guidance from a company specialized in Agile Transformation. Each project belongs to a distinct business area and are distributed in at least two sites each. We will be present at the Brazilian site but we also got permission to participate in virtual meetings and conference calls with remote team members.

We aim to gather data about how the teams are organizing themselves and why such organization, which practices they are adopting and why, which issues they are going through and how they are solving them, and which needs they have that require feedback from senior management. We will take into consideration the work environment they have (e.g., applications and team members background, imposed organizational guidelines, etc) and collect the perception they have on how such environment influences the transformation. Data will be collected and analyzed simultaneously to allow for follow-up clarifications and exploration of new insights coming from the observed data.

4. Final Remarks

This paper presents our long-term goal to define a set of guidelines to support the ATP in large-complex globally distributed companies. We discuss in more details one of the four research phases, Phase 2, composed of a case study, highlighting the importance to collect empirical evidence in this kind of study. Although we have just recently started our investigation, we have already identified a set of aspects that have to be observed during the ATP from literature. We expect that our findings will shed some light in understanding the phenomena and providing guidance to companies who want to go through this process.

Acknowledgment

This work is sponsored by the PDTI Program, financed by Dell Computers of Brazil Ltd. (Law 8.248/91).

References

Beck, K. and colleagues (2001). Manifesto for agile software development.

Dingsøyr, T. and Moe, N. (2014). Towards principles of large-scale agile development. In Dingsøyr, T., Moe, N., Tonelli, R., Counsell, S., Gencel, C., and Petersen, K., editors, *Agile Methods. Large-Scale Development, Refactoring, Testing, and Estimation*, volume 199 of *Lecture Notes in Business Information Processing*, pages 1–8. Springer.

Fontana, R., Meyer, V., Reinehr, S., and Malucelli, A. (2015). Progressive outcomes: A framework for maturing in agile sw development. *J. of Systems and Sw*, 102:88–108.

Gandomani, T., Zulzalil, H., and Nafchi, M. (2014). Agile transformation: What is it about? In *Proc. Malaysian SEng. Conf., 2014, Langkawi, Malaysia*, pages 240–245.

Ganesh, N. and Thangasamy, S. (2012). Lessons learned in transforming from traditional to agile development. *Journal of Computer Science*, 8:389–392.

Korhonen, K. (2013). Evaluating the impact of an agile transformation: a longitudinal case study in a distributed context. *Software Quality Journal*, 21:599–624.

Ranganath, P. (2011). Elevating teams from 'doing' agile to 'being' and 'living' agile. In *Proceedings of the Agile Conference, Aug 2011, Salt Lake City, USA*, pages 187–194.

Guidelines for Modularizing the Monitor Component when Refactoring Adaptive Systems

Marcel A. Serikawa[1], **Bento R. Siqueira**[1], **Fabiano C. Ferrari**[1], **Ricardo Menotti**[1], **Valter V. de Camargo**[1]

[1]Computing Department – Federal University of São Carlos (UFSCar)
Caixa Postal 676 – 13.565-905 – Sao Carlos – SP – Brazil

{marcel.serikawa, bento.siqueira, fabiano, menotti, valter}@dc.ufscar.br

Abstract. *Adaptive systems are system that can adapt themselves to environmental or internal changes in order to improve their quality of service. Most authors agree that control loops are an intrinsic part of these systems, but in most cases they are designed and implemented in a spread and tangled way, harnessing maintenance activities. An alternative for that is to perform refactorings aiming at re-modularizing control loops as first class entities. Therefore, in this paper we present our initial steps in creating a refactoring catalogue for adaptive systems based on MAPE-K model. Our intention is to provide some guidelines for refactoring the monitor component. Our approach is illustrated with code snippets from a context-aware mobile application.*

1. Introduction

Adaptive systems are able to modify their behavior and/or structure in response to a changing environment[Garlan and *et al* 2004, Cheng and *et al* 2009]. Most authors agree that this kind of systems is naturally and intrinsically composed by a set of control loops [Weyns and *et al* 2013], which are a sequence of processes to gather information from the managed system and its environment, process this information and make necessary changes to achieve a specific goal [Garlan and *et al* 2004]. Usually, a control loop is composed by the following components: Monitor, Analyzer, Planner and Executor [IBM 2006].

Previous research have shown that control loops are often tangled and spread with the system main logic [Cámara and *et al* 2013], harnessing maintenance and evolution activities. To solve these problems, several authors [Cámara and *et al* 2013, Garlan and *et al* 2004] state that designing control loops in a modular way can reduce the complexity and improve the maintainability. A possible alternative for that is by refactoring the source code, aiming to change its structure without changing its external observed behavior [Fowler and *et al* 1999]. However there is a lack of systematic knowledge about refactoring adaptive systems in order to modularize their control loops.

Although there is no an exact solution for designing adaptive software, the majority of solutions are based on the MAPE-K model proposed by IBM (2006) [Weyns and *et al* 2013]. Therefore in this paper we present two design alternatives for refactoring the Monitor component. The Monitor is the Control Loop element responsible for collecting information from the managed system and the environment in order to update the Control Loop Knowledge. Before updating the Knowledge, the Monitor

may preprocess the collected data, which may include conversions, standardization, data aggregation and data filtering [Gil de La Iglesia 2014].

In this paper we have focused on two major steps on refactoring the monitor: (1) the identification of code snippets that represent monitor abstractions and (2) modularization of the identified monitor code in two design alternatives. The first one is based on the observer pattern and it is called Event-Triggering, and the another is based on Polling and it is called Time-Triggering.

2. Monitor Identification

Although each system can be built in many different ways, we still have some good evidences to find the monitor abstraction in the code. An important characteristic is that they have to be frequently updated with sensor's data, it gives two important evidences about the monitor code snippet. The first is that it should be a sensor instantiated, and the second is that the monitor abstraction is inside a loop structure statement being constantly updated. Although, it is very important to emphasize that to accomplish an adaptive system monitor modularization, the software engineer needs a good knowledge about the system [Cámara and *et al* 2013].

To exemplify this identification and refactoring process it is used a context-aware mobile application called PhoneAdapter [Sama and *et al* 2010]. This application is responsible to adapt the mobile configuration according to the data gathered by sensors like GPS, Bluetooh, system calendar and users needs. In the Figure 1 it is shown the class ContextManager code, where it is possible to identify the Monitor abstraction evidences as listed before. In this class there are sensor devices imported as can be seen in lines 2-6 and a "while" statement in line 11 being processed every 2 minutes (line 31).

```
1  ...
2  import java.util.Calendar;
3  import android.bluetooth.BluetoothAdapter;
4  import android.bluetooth.BluetoothDevice;
5  import android.location.Location;
6  import android.location.LocationListener;
7  ...
8
9  public class ContextManager extends IntentService {
10 ...
11     while(!mStop){
12         mCal=Calendar.getInstance();
13         mTime=mTimeFormat.format(mCal.getTime());
14         switch(mCal.get(Calendar.DAY_OF_WEEK)){
15         case 1:
16             mWeekday="sunday";
17             break;
18         case 2:
19             mWeekday="monday";
20             break;
21         ...
22         }
23
24         if(mBtAdapter != null){
25             if(!mBtAdapter.isEnabled()){
26             ...
27             }
28         }
29         ...
30         try{
31             Thread.sleep(120000);
32         ...
33     }
34 }
```

Figure 1. Monitor evidences.

3. Monitor Modularization

In Figure 2 it is shown the class diagram of Event-triggering on the left and Time-triggering on the right. The Event-triggering Monitor is designed as the Observer Pattern, therefore the Sensor class, the Subject, adds the Monitor in a list of Observer, the Sensor implements the method `Sensor.notifyObservers()` responsible to call the `Monitor.notify()` method when there is a data change event. The

Time-triggering Monitor class has the method `Monitor.update()` that will call the `Sensor.getData()` method to gather the Sensor's data, this process occurs according to the attribute timer. The main difference between these monitors is which element will trigger the data update process and the choice between which monitor to be used is based on the data being monitored.

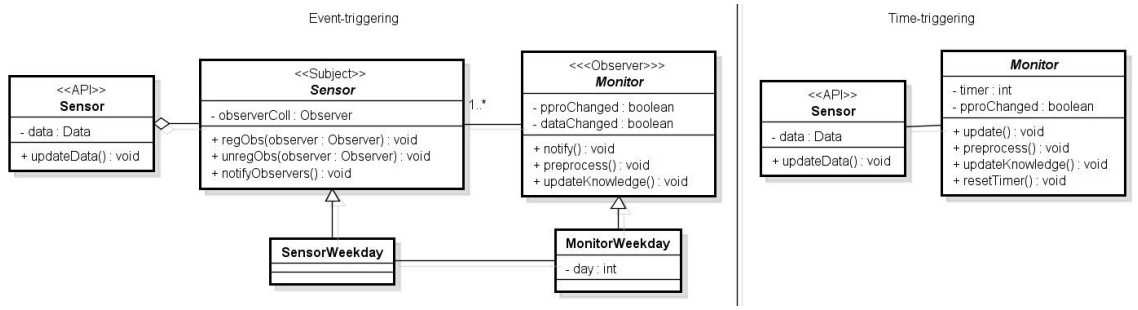


Figure 2. Monitor Class Diagram with Event- (left) and Time-triggering (right).

In the Figure 3 it is shown an Event-triggering Monitor example, there are two classes `SensorWeekday` on the right and `MonitorWeekday` on the left, with the stereotype Observable and Observer respectively. The SensorWeekday class implements Observable which is the Subject equivalent in JAVA, this class instantiates the system sensor Calendar (line 4) from where it collects the original data. The method `notifyObservers()` is responsible to update the Monitors everywhen the day attribute value has change, as can be seen in the method `setDay()`.

```
1  public class SensorWeekday extends Observable {
2
3      private SensorWeekda(){
4          this.calendar = Calendar.getInstance();
5      }
6
7      public void notifyObservers(){
8      ...
9      }
10
11     public void setDay(int day){
12         if(this.day!=day){
13             this.day = day;
14             notifyObservers();
15         }
16     ...
17
18
```

```
1  public class MonitorWeekday implements Observer {
2
3      public void update(Observable sensorWeekday, Object obj){
4          ...
5      }
6
7      public void preprocess(int day){
8          String mWeekday = "";
9          switch(day){
10             case 1:
11                 mWeekday="sunday";
12                 break;
13             case 2:
14                 mWeekday="monday";
15                 break;
16             ...
17         }
18     ...
```

Figure 3. Example of a event-triggering refactor.

The MonitorWeekday class implements the `update()` method (line 3) which is equivalent of the notify() method in Observer Patter, it is responsible to get the information provided by the Sensor (Subject). The `MonitorWeekday.preprocess()` method is responsible to pre-process the data gathered, therefore the code from lines 15 - 22 in Figure 1 is placed in this method. After this process, this data is updated in the Knowledge component and then processed by the others control loop components.

This monitor choice was based in the data type weekday, assuming that this value changes every 24 hours it does not need to be updated every two minutes as it is done in

the original class, Figure 1, line 31. And since in this Event-triggering monitor the Sensor component is responsible to trigger the monitoring process, it should be instantiate inside a class responsible to manage all Sensors or as a separated thread.

4. Conclusion

The guidelines provided in this paper are an initial set which still need to be extended in order to be fully applicable. Up to this moment, we have identified two kinds of monitors and explained their characteristics, what assists modernization engineers along the identification process. We have also provided two design alternatives for the re-modularization. Although we have not conducted an experiment yet, we have evidences that the modularized version of the system has its maintainability and understandability improved. As future work we intend to analyze the remaining components of control loop and develop the complete refactoring catalogue for Adaptive Systems.

References

Cámara, J. and *et al* (2013). Evolving an adaptive industrial software system to use architecture-based self-adaptation. In *Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2013 ICSE Workshop on*, pages 13–22. IEEE.

Cheng, B. H. C. and *et al* (2009). Software engineering for self-adaptive systems: A research roadmap. In Cheng, B. H. C., de Lemos, R., Giese, H., Inverardi, P., and Magee, J., editors, *Software Engineering for Self-Adaptive Systems [outcome of a Dagstuhl Seminar]*, volume 5525 of *Lecture Notes in Computer Science*, pages 1–26. Springer Berlin Heidelberg.

Fowler, M. and *et al* (1999). Refactoring: Improving the design of existing programs.

Garlan, D. and *et al* (2004). Rainbow: Architecture-based self-adaptation with reusable infrastructure. *IEEE Computer*, 37(10):46–54.

Gil de La Iglesia, D. (2014). *A Formal Approach for Designing Distributed Self-Adaptive Systems*. PhD thesis, Linnaeus University, Department of Media Technology.

IBM (2006). Autonomic computing white paper: An architectural blueprint for autonomic computing. *IBM White Paper*, page 34.

Sama, M. and *et al* (2010). Context-aware adaptive applications: Fault patterns and their automated identification. *IEEE Transactions on Software Engineering*, 36(5):644–661.

Weyns, D. and *et al* (2013). On patterns for decentralized control in self-adaptive systems. In *Software Engineering for Self-Adaptive Systems II*, pages 76–107. Springer.

Furthering Knowledge on How Behavior-Driven Development Can Support Requirements Elicitation

Lauriane Correa[1], **Sabrina Marczak**[1], **Cleidson R. B. de Souza**[2]

[1]Computer Science School – Pontifícia Universidade Católica do Rio Grande do Sul
90619-900 – Porto Alegre – RS – Brasil

[2]Instituto Tecnológico Vale e Universidade Federal do Pará
66055-080 – Belém – PA – Brasil

lauriane.moraes@acad.pucrs.br, sabrina.marczak@pucrs.br

cleidson.desouza@acm.org

Abstract. *Requirements elicitation major's challenge is establishing common ground with the customer. Behaviour-Driven Development (BDD), inspired on the concept of 'Specification by Example', proposes a structured, English-based format named scenario to state the desired behavior for the software to be built. We aim to understand BDD usage in the field by first exploring the topic through a multiple case study and later by confirming the preliminary findings in a large-scale survey. While we are still conducting the exploratory study, we have also started planning the survey. In this paper we introduce our research plan to conduct the survey aiming to promote discussion on how to better acquire comprehension about the phenomena.*

1. Introduction

Requirements elicitation tries to discover the application domain, business needs, requirements and system constraints by consulting stakeholders [Sommerville 2010]. Requirements analysts and stakeholders often do not share a common understanding of related concepts and terms. Such lack of common ground can cause misalignment of the elicited requirements [Zowghi and Coulin 2005]. Stakeholders also often have difficulties expressing their needs, making it harder to define the software expected behaviors.

A recurrent reported issue in literature is the difficulty of software teams to communicate clearly [International 2013], causing projects to go over budget or fail. That is why Behavior-driven Development (BDD) emerges as a promessing approach. BDD is the name given to a set of methods and techniques put together aiming to help teams to focus their efforts on identifying, understanding, and building valuable features that matter to businesses, and to ensure that these features are well designed and implemented [Smart 2014]. The way the pieces are tied together aims to ensure consistency and traceability of requirements throughout the development life cycle, to allow for timely communication with anyone involved in the project, including the customer. Communication aspects become less important since BDD uses a structured form.

Despite the promissed benefits and the anedoctal reports of how much BDD can do for a software team, there is little empirical evidence of the extent that it can specifically support requirements elicitation. To fill in this gap, we posed the following research

question: *How can BDD support requirements elicitation in practice?*, and designed an empirical study to answer it. We set to first explore how BDD is used in practice through a case study and then later confirm the preliminary findings in a large-scale survey. Our interest in a large-scale study is to better understand the contexts in which BDD can support requirements engineering (e.g., when the analyst writes the specification, when there is an internal customer), aiming for generalization of our findings. While we briefly introduce our entire research plan to provide context, the goal of this paper is to present the survey design aiming to collect feedback to help us ensure we are in the right path.

2. Behavior-Driven Development in a Nutshell

Behavior-driven Development (BDD) was designed to help teams build and deliver more valuable, higher-quality software faster [Smart 2014]. It was initially proposed by Dan North as a way to teach Test-driven Development (TDD) [North 2006]. It is composed of a set of practices from agile methodologies, such as TDD, automated acceptance testing, and continuous building [Smart 2014]. It also incorporates the definition of features or requirements based on examples as proposed by [Adzic 2011].

BDD provides a connection from code to the requirements, offering a better environment for managing project progress. This is done through scenarios that define a way to describe how the system should behave based in a language that is native to the stakeholder, promoting a common understanding of the business domain between stakeholders and development team [Evans 2003]. BDD starts by identifying relevant business goals and software features that will cover these goals. Collaborating with the customer, BDD practitioners use concrete examples to illustrate the features. These features can be broken down into smaller chunks, named user stories, when more than one aspect composes them. The defined examples are then automated in the form of executable specifications that follow a structured format named 'scenarios'.

Figure 1 illustrates the notation defined to write an example that represents a certain feature. This feature is composed of two scenarios, each composed of a set of steps marked by pre-defined clauses as explained next. A *Feature* is a descriptive text of what is desired by the customer. This description provides context to those reading and

Feature title: *Buying books with the bookstore card*
Narrative: **In order to** buy the books **As a** bookstore client **I want to** pay my chosen books with my bookstore card
Scenario 1: *Paying with a positive balance*
Given my bookstore card has a balance of 300.00 And my bookstore card give me 15% discount **When** I buy 100.00 from my cart in books **Then** I should pay 85.00 And I should have 215.00 left in my bookstore card
Scenario 2: *Paying without sufficient balance*
Given my bookstore card has a balance of 50.00 And my bookstore card give me 15% discount **When** I buy 100.00 from my cart in books **Then** I should receive an 'insufficient balance' error message And I should still have 50.00 in my bookstore card

Figure 1. Illustration of a Feature and Its Scenarios using BDD

using a feature definition and describes the business value of the feature to the software as a whole [North 2006]. A feature usually contains a list of *Scenarios* [Smart 2014]. Each scenario is composed by a set of pre-defined clauses, namely: 'Given', 'When', and 'Then'. *Given* describes the preconditions for the scenario and prepares the test environment. *When* describes the key action the user performs, or state transition. *Then* is used to describe outcomes. The observations should inspect the output of the system (a report, user interface, message, command output). *And* and *But* are additional clauses used to join the previous clauses and provide a more readable way to specify the feature.

3. Proposed Survey in the Context of Our Long-Term Research Plan

Given its novelty, there is little empirical evidence how BDD is used in practice and none, to the best of our knowledge, on how it addresses requirements elicitation issues. To better understand this phenomena and explain the benefits and challenges of BDD adoption, we are currently conducting an exploratory study organized in two smaller steps as indicated in Figure 2, Phase 2, and are preppering for applying a survey (Phase 3) as previously mentioned. We briefly introduce the exploratory studies to provide context.

Our goal conducting the Interview step was to develop a initial understanding about how BDD is defined and used in practice. We have conducted ten semi-structured interviews before the book 'BDD in Action' [Smart 2014] has been released. This book provides a consolidated description of BDD and its related techniques. While analyzing data from the interviews, we started observing team members of a project at a large agile IT company with development centers located in five continents. The case was selected based on convenience and access to the company's office in Brazil. In this project, analysts discussed the needs for a software solution with the customer and wrote the elicited features down using user stories. Later, these stories were transformed into scenarios with the development team's help and then automated by a tool that supports BDD. We have just recently joined the company on site. An iteration is about to be completed soon, allowing us to get familiar with all activities of an interation cycle.

Although we are still conducting the Case Study, we have already started designing the Survey study (Phase 3). Our goal with the survey is to aim for generalization of our findings from Phase 2. So, for now, we have decided that our population is IT professionals located in any place around the world who adopt BDD. We will follow a snowballing sampling to select our sample, that should be as large as possible given the two months we plan to keep the survey open. We will ask the ten participants of our interview step to indicate colleagues and request them to indicate other people. We have also already started to look for additional respondents by inspecting discussion groups in social media websites such as LinkedIn. Eight groups of interest located in Latin America, Europe, and Asia have been identified so far.

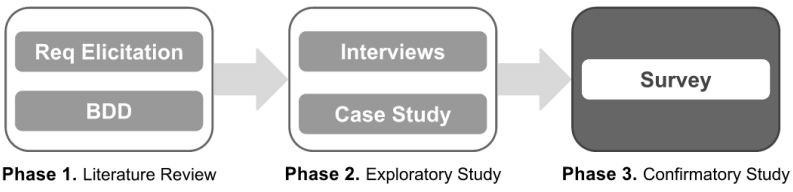


Figure 2. Proposed Research Method

The survey will be made available online and will be non-supervised. We have already tested the Qualtrics survey tool to make the questionnaire instrument available but we learned that the license our University owns is limitated to 200 respondents. We are looking for a reliable, free-of-cost alternative solution at this moment.

The survey instrument will be composed of closed questions only, in a Likert-scale format. Response choices will be designed based on the findings from the literature review, our interviews, and the insights from our case study. We have an initial draft but this work will be refined and completed as the case study is finished and data analyzed. The general constructs we are considering to design the questions are as follows: BDD concept, BDD activities, roles involved in the process, artifacts used, tools adopted, benefits of adoption and challenges faced when using BDD. Requirements engineering related specific contructs are as follows: requirements elicitation issues, requirements quality, product quality, communication issues, common ground establishment, and obsolete documentation. Our survey design is following Kitchenham's and Pfleeger's guidelines [Kitchenham and Pfleeger 2002].

4. Final Remarks

BDD aims to promote collaboration and to facilitate communication among stakeholders and the software team. We aim to further our knowledge about BDD usage in the wild by conducting a large-scale survey to confirm findings from our previous initial exploratory studies. This paper presented our high level plan to conduct the survey. We hope that our about-to-come-soon findings will motivate additional practitioners to adopt BDD and researchers to explore other aspects related to this topic.

Acknowledgment

This work is sponsored by the PDTI Program, financed by Dell Computers of Brazil Ltd. (Law 8.248/91).

References

Adzic, G. (2011). *Specification by Example: How Successful Teams Deliver the Right Software*. Manning Publications Co., Greenwich, CT, USA.

Evans, E. (2003). *Domain-Driven Design: Tacking Complexity In the Heart of Software*. Addison-Wesley, Boston, MA, USA.

International, T. S. G. (2013). The chaos manifesto.

Kitchenham, B. A. and Pfleeger, S. L. (2002). Principles of survey research part 2: Designing a survey. *SIGSOFT Softw. Eng. Notes*, 27(1):18–20.

North, D. (2006). Introducing BDD. `http://dannorth.net/introducing-bdd/`.

Smart, J. (2014). *BDD in Action: Behavior-Driven Development for the Whole Software Lifecycle*. Manning Publications, Shelter Island, NY.

Sommerville, I. (2010). *Software Engineering*. Addison-Wesley, England, 9 edition.

Zowghi, D. and Coulin, C. (2005). *Requirements Elicitation: A Survey of Techniques, Approaches*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

What challenges project managers face in software crowdsourcing?

Graziela Basilio Pereira, Alexandre Lazaretti Zanatta, Rafael Prikladnicki

[1]Computer Science School - PUCRS, Porto Alegre, Brazil 90619-900

{graziela.basilio, alexandre.zanatta}@acad.pucrs.br,

rafael.prikladnicki@pucrs.br

Abstract. *Crowdsourcing means outsourcing to the crowd and can be used to support software development activities. In this paper we present software crowdsourcing, a global trend and show how Brazilian project managers are inside this context. We also introduce a preliminary study about the challenges faced by project managers in software crowdsourcing projects.*

1. Introduction

Crowdsourcing (CS) means outsourcing to the crowd. It is a compound contraction of Crowd and Outsourcing and represents "the act of a company or institution taking a function once performed by employees and outsource it to an undefined (and generally large) network of people in the form of an open call" [Howe 2006].

CS rises as an option to software development projects, providing access to a scalable workforce of online experts. It also promises cost savings, time to market, productivity and flexibility, tapping the intellect of the crowd [Carmel 1999]. At first sight CS emerges as a solution for software projects with limited budgets and constrained resources.

In a software development process CS can be adopted at certain stages or throughout the whole project. Any software development task can be crowdsourced, including requirements, design, coding, testing, evolution and documentation [Huhns et al. 2013]. There are platforms that cover every software development process like TopCoder and others which were designed to attend specific tasks, like uTest. Furthermore, CS can be incorporated independently of the software methodology or lifecycle, such as waterfall or agile [Huhns et al. 2013]. Enterprises have been outsourcing software development for a long time through the use of other companies to build their software solutions. During this period project managers were learning, developing methods and techniques to work on projects with this feature.

Software development projects require communication, coordination, and management. Organizations increasingly sought the project management area to solve problems and guarantee successful projects, that meet clients' needs with quality [Schwalbe 2013].

Recently, a new software engineering approach is emerging, enabled by cloud solutions that provide large scale and highly available computational resources. This new approach, known as crowdsourcing software development or software crowdsourcing, uses the cloud to outsource parts or the entire software project to a crowd of developers [Huhns et al. 2013].

In software CS processes the project manager can play the part of any of the three main CS actors. Buyer (sponsor/client) - company that places the work requests (tasks). In this case the project manager is responsible for monitoring and controlling the contracted project. Crowd - a community of developers globally dispersed. Individual suppliers that will effectively perform the tasks. As development (coding, testing, analysis, and documentation) can be outsourced, nothing avoids the project manager to join the crowd. CS Platforms - the platform is the middleman, i.e., it intermediates the communication between buyers and the crowd itself. The CS systems show the tasks requirements that should be solved by solvers (crowd). Some platforms have the role of the project manager to ensure that projects contracted through the platform meet the goals.

Unfortunately, there is a lack of information available about project management on software CS. As a result, there are several questions surrounding this area, such as: What are the challenges of managing software CS projects? What strategies can be adopted to minimize the challenges while managing a software CS project? Is there any difference between manage a software CS and a distributed software development project?

In order to identify and answer these questions, we planned an empirical research study based on a survey and collected data from project managers in Brazil. In the next sections we introduce the preliminary results of a pilot survey and four interviews. We conclude the paper presenting our analysis based in the data collected.

2. Software crowdsourcing: Is it increasing?

The crowd work industry is now quickly growing in scope and ambition. Crowd work today spans a wide range of skills and pay levels, with commercial vendors providing access to a range of workers and focused support for various tasks [Kittur et al. 2013].

The benefits of CS to the organizations, like cost reduction, time to market, productivity and flexibility, tapping the intellect of the crowd are evidenced by many authors [Huhns et al. 2013] [Kittur et al. 2013] [Carmel 1999].

Along with the benefits to the companies, CS equally presents advantages to the workers like flexible schedule, extra remuneration and learning. Besides this, it also provides new opportunities for income and social mobility in regions where local economies are stagnant or in those where local government structures discourage investment [Kittur et al. 2013].

This context is emphasized by most of the industrial software giants like Apple, Oracle and Microsoft. These firms are engaging and adopting CS to create new products, start new projects, secure funding and identifying talents [Huhns et al. 2013]. The workforce in a crowd also evidences this growing. TopCoder, a CS plataform created in 2001 has reached more than 500,000 members in 2014 [Huhns et al. 2013].

The presented information supports that the crowd work continues to expand, unlocking an incredible number of opportunities for careers and skilled work in online marketplaces.

To identify if Brazil is following this global trend, from the perspective of the project managers, we performed a pilot survey, which received 363 project managers answers. The survey presented questions about the usage, experience, challenges and

recommendations for managing software CS projects. Through a quantitative analysis it became clear that Brazil is not following the CS rapid growth. The collected data showed us how many of the respondents were aware of the term Crowdsourcing. Only 35 % had already heard about CS before the survey, while 65 % were unaware about it. Just 7% of the respondents already had some experience with CS, indicating that the Brazilian market is immature in this area.

This moderated adhesion occurs due to several factors according to [Machado et al. 2014]: buyers prefer to develop in-house or outsourcing, fear of exposing strategic business information, concern to ensure deadlines, uncertainty about laws and taxes that may be involved in CS activities.

3. Software CS: What are the challenges faced by project managers?

According to [Malone et al. 2010] making CS manageable and controllable is currently a concern in CS projects. Software development processes require coordination, communication, and management. A large number of organizations increasingly sought the project management area to solve problems and guarantee successful projects that meet clients' needs with quality [Schwalbe 2013]. To unlock the potential of the crowd's work, managers need a deeper understanding of how these projects are developed and if additional or different practices are needed to manage software CS projects.

We have gathered the challenges related to the pilot survey and four project manager interviews. The most quoted challenges for managing CS software projects were related to communication, data confidentiality, people management and time management. The key challenges mentioned were:

- Coordinate activities in the crowd
- Track the work progress and guarantee the tasks deadline and quality
- Ensure the project confidentiality
- Changes management
- To specify product requirements without limiting innovation and creativity
- Recognize how to define which activities can be done in the crowd
- Resource management - allocation, engagement and ensure the commitment

Respondents expressed concern about the exposure of project's strategic information and recommended that the more strategic it is, the less CS should be used. The ability to communicate with people at all management levels was the most important concern identified.

Finally, the appropriate project selection and the team training methods were also cited as concerns, once not every project is suitable for CS application and some people have to be trained to work in this environment.

4. Conclusion

CS is relatively new and many of its grand promises and bad predictions have yet to spread. Despite of that, firms and employees are already engaged into the CS market, seeking the model, its unique benefits and accepting consciously or not the associated risks [Felstiner 2011]. The project manager has an important role to identify the inherent risks, set its boundaries, mitigate them and benefit from the advantages that CS can provide.

Throughout our analysis we believe that all the challenges were adherent to any of the knowledge areas of PMBOK [PMI 2014]. There were no issues that did not fit in these areas. This lead us to think that CS projects can be managed based on the guide of knowledge proposed by PMI. This hypothesis was reinforced along two interviews in which project managers affirmed that software CS projects can be treated like any other project. According to them, these projects may require deeper management in some fields of expertise or specific techniques and tools to communicate or monitor the project.

The next steps in this research includes (1) to conclude the analysis of all answers received in the pilot survey and interviews, (2) to execute a comprehensive literature review on managing CS projects, (3) to interview project managers in order to deeply explore the challenges and recommendations identified, and (4) to propose project management practices for managing CS projects.

References

Carmel, E. (1999). *Global software teams: collaborating across borders and time zones*. Prentice Hall PTR.

Felstiner, A. (2011). Working the crowd: employment and labor law in the crowdsourcing industry. *Berkeley J. Emp. & Lab. L.*, 32:143.

Howe, J. (2006). The rise of crowdsourcing. *Wired magazine*, 14(6):1–4.

Huhns, M. N., Li, W., and Tsai, W.-T. (2013). Cloud-based software crowdsourcing (dagstuhl seminar 13362). *Dagstuhl Reports*, 3(9).

Kittur, A., Nickerson, J. V., Bernstein, M., Gerber, E., Shaw, A., Zimmerman, J., Lease, M., and Horton, J. (2013). The future of crowd work. In *Proceedings of the 2013 conference on Computer supported cooperative work*, pages 1301–1318. ACM.

Machado, L., Pereira, G., Prikladnicki, R., Carmel, E., and de Souza, C. R. (2014). Crowdsourcing in the brazilian it industry: what we know and what we don't know. In *Proceedings of the 1st International Workshop on Crowd-based Software Development Methods and Technologies*, pages 7–12. ACM.

Malone, T. W., Laubacher, R., and Dellarocas, C. (2010). The collective intelligence genome. *IEEE Engineering Management Review*, 38(3):38.

PMI, P. M. I. (2014). A guide to the project management body of knowledge(pmbok guide). fifth edition.

Schwalbe, K. (2013). *Information technology project management*. Cengage Learning.

Colaboração e Cooperação em Equipes Ágeis: uma investigação baseada na simulação de agentes

Adriana Neves dos Reis [1, 2]

[1]Instituto de Ciências Exatas e Tecnológicas – Universidade Feevale
Novo Hamburgo – RS – Brasil

[2]Programa de Pós-Graduação em Engenharia de Produção e Sistemas – UNISINOS
São Leopoldo – RS – Brasil

adriananr@feevale.br

Abstract. *Agile software development teams have their productivity linked to interpersonal collaboration mechanisms. In practice, however, this feature is not always observed, compromising the team's results, and even continued to maintain agility as base value of your process. Whereas collaboration it is a social behavior, dependent people, the proposed research presented in this paper aims to apply a model based on agents to understand the relationships between profiles and people in agile teams. To this end, we intend to investigate scenarios for analysis of actions that contribute to the increase in the degree of collaboration and cooperation between the actors of the process, and to evaluate the influence of context on these behaviors.*

Resumo. *As equipes ágeis de desenvolvimento de software têm sua produtividade atrelada aos mecanismos de colaboração interpessoal. Na prática, entretanto, nem sempre esta característica é observada, comprometendo os resultados da equipe, e até mesmo a continuidade de manter a agilidade como valor base de seu processo. Considerando que colaboração trata-se de um comportamento social, dependente das pessoas, a proposta de pesquisa apresentada neste artigo tem como objetivo a aplicação de um modelo baseado em agentes para compreender as relações entre perfis e pessoas em equipes ágeis. Para tanto, pretende-se investigar cenários para a análise de ações que contribuam para o aumento do grau de colaboração e cooperação entre os atores do processo, bem como avaliar a influência do contexto nestes comportamentos.*

1. Introdução

Os métodos ágeis de desenvolvimento de software têm sua produtividade atrelada ao trabalho em equipe, visto que um de seus pilares é a interação entre os indivíduos em substituição à intensa documentação para fins de comunicação entre os mesmos nos modelos de processo tradicionais. Contudo, apesar de mais de dez anos de adoção no mercado, sabe-se empiricamente que nem toda equipe consegue apresentar os níveis de produtividade esperados na abordagem ágil.

Entre as razões mencionadas no dia-a-dia das empresas de software para o insucesso das práticas ágeis estão: a não adequação da cultura organizacional à filosofia ágil, a não adoção das práticas ágeis no contexto previsto (como tamanho de equipe, papéis, etc), a falta de maturidade técnica da equipe, a falta de compreensão do que é agilidade, entre outros. Porém, além desses, observa-se uma dificuldade dos desenvolvedores em assumir uma postura de colaboração, de a comunicação pessoal ser uma constante, e de cooperação, ou seja, de conseguir construir software e resolver problemas em conjunto.

Tal contexto de colaboração versus cooperação gera evidências de que, além do contexto organizacional, é necessário entender como os indivíduos da equipe se aderem à prática ágil, o que parece ser influenciado pelo seu perfil em diferentes aspectos: profissional, técnico, social, entre outros. Dessa forma, é preciso considerar que cada membro da equipe possui suas crenças, competências, e objetivos, os quais são fatores de interferência no direcionamento das ações a favor de traçar e alcançar o objetivo do grupo. Além disso, a interação entre eles e o ambiente da empresa podem contribuir ou atrapalhar a execução das práticas ágeis.

Uma estratégia para abstrair esse fenômeno é a Modelagem e Simulação baseada em Agentes, por ser uma técnica que opera no nível micro individual [1]. Um agente é a representação de um indivíduo com características específicas, o qual interage com outros agentes em um contexto compartilhado. Logo, o modelo descreve um sistema reativo que exibe certa autonomia para decidir quão bem deve executar uma tarefa a ele delegada [2].

Assim, o objetivo desta pesquisa é investigar quais os efeitos das características dos desenvolvedores para o comportamento da equipe ágil em relação aos aspectos de colaboração e cooperação. Para tanto, a proposta é, a partir de dados coletados em equipes reais, investigar a dinâmica da equipe ágil, a partir da investigação *in silico*, utilizando modelagem e simulação baseada em agentes.

2. Agenda de Pesquisa

Em um sistema adaptativo, “agentes representam as unidades básicas do processo da tomada de decisão” [3] (p. 55). Um modelo baseado em agentes utiliza uma abstração *bottom up,* baseada em agentes com funções simples.

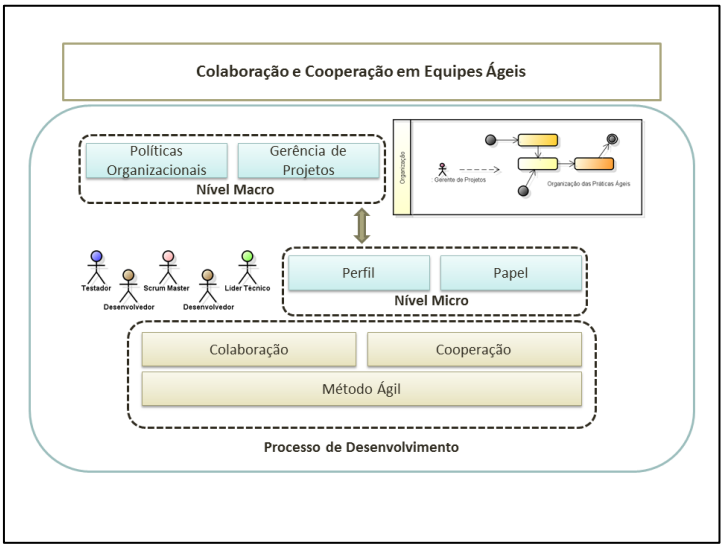


Figura 1. ***Framework*** **da pesquisa proposta. Fonte: o autor.**

Além disso, quando estes indivíduos interagem, considerando suas funções, alguns comportamentos específicos emergem [2]. Dessa forma, neste contexto, é relevante associar simulação para investigar propriedades da dinâmica do processo estudado.

A abordagem baseada em agentes tem sido adotada em estudos para permitir a definição de modelos, os quais analisados em cenários reais, consequentemente, servem de base para a proposição de recomendações de ação nestes contextos [1].

Assim, na Figura 1 é apresentado o *framework* da pesquisa proposta, organizado em níveis de abstração, de forma esquemática.

3. Potencialidades do Estudo

De forma empírica, mesmo com a disseminação das práticas ágeis e o maior grau de sua adoção, ainda são comuns relatos sobre a dificuldade de obtenção de melhorias significativas em relação à produtividade das equipes que praticam agilidade. Assim, compreender o comportamento dos atores em uma equipe ágil e seus critérios para comportamento em relação à colaboração e cooperação é um desafio na perspectiva de estudos que abordam os aspectos sociais em Engenharia de Software.

O modelo proposto evidencia que Modelagem e Simulação baseada em Agentes possuem características potenciais como ferramenta de investigação do ambiente de TI (Tecnologia da Informação) e, consequentemente, oferece suporte para tomada de decisão que estimule este tipo de prática com resultados de produtividade satisfatórios.

Referências

[1] E. Kiesling, M. Günther, C. Stummer, and L. M. Wakolbinger, "Agent-based simulation of innovation diffusion: A review," *Cent. Eur. J. Oper. Res.*, vol. 20, no. 2, pp. 183–230, 2012.

[2] R. H. Bordini, J. F. Hübner, and M. Wooldridge, *Programming multi-agent systems in AgentSpeak using Jason*, vol. 8. John Wiley & Sons, 2007.

[3] N. Nan, R. Zmud, and E. Yetgin, "A complex adaptive systems perspective of innovation diffusion: an integrated theory and validated virtual laboratory," *Comput. Math. Organ. Theory*, vol. 20, no. 1, pp. 52–88, 2013.

Meta-Aprendizado Aplicado à Estimativa de Esforço em Projetos de Desenvolvimento de Software

Silvia Nunes das Dôres[1], **Duncan Ruiz**[1]

[1]Faculdade de Informática – Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Caixa Postal 1429 – 90.619-900 – Porto Alegre – RS – Brasil

silvia.dores@acad.pucrs.br, duncan.ruiz@pucrs.br

Abstract. *Effort estimation is one of the cores of a software development project due to its influence in many management tasks, such as cost and schedule estimate. Given the importance of this area, there is a great deal of effort to propose new techniques and improve the accuracy of estimates. Among these approaches there are techniques based on Machine Learning (ML). Although there is a large number of ML approaches aimed at predicting effort, there is no rule of thumb to choose the most appropriate algorithm for a given organizational context. Thus, this paper aims to present a proposal for meta-learning use to recommend ML algorithms on estimating effort in software development projects.*

Resumo. *Estimativa de esforço é um dos cernes de um projeto de desenvolvimento de software uma vez que ela influencia diversas tarefas gerenciais, tais como, estimativa de custo, cronograma e prazo. Dada a relevância da área, existem diversas pesquisas voltadas para a proposta de novas técnicas para melhorar a precisão das estimativas. Dentre essas abordagens destacam-se as técnicas de baseadas em Aprendizado de Máquina (AM). Embora exista um grande número de abordagens de AM voltadas para predição de esforço, faltam métodos que auxiliem na escolha do algoritmo mais apropriado para um dado contexto organizacional. Neste sentido, este trabalho tem o objetivo apresentar uma proposta para utilização de meta-aprendizado na recomendação de algoritmos de AM para estimar esforço em projetos de desenvolvimento de software.*

1. Introdução

Uma boa estimativa de esforço é essencial para ajudar os gerentes de projeto a alocar recursos e controlar os custos e o cronograma de seus projetos, o que por sua vez permite que os projetos sejam finalizados no tempo e dentro do orçamento [Mendes 2014]. Porém, estimar esforço não é uma tarefa fácil, uma vez que diversos fatores como complexidade do projeto, tamanho e grau de incerteza podem afetar a confiabilidade destas estimativas levando a superestimação ou subestimação do esforço dos mesmos [Pressman 2011].

Dentre as soluções existentes para apoio a esta atividade destacam-se as soluções baseadas em Aprendizado de Máquina (AM), que visam predizer o esforço para um projeto novo baseado em dados históricos de projetos anteriores. Dentre as técnicas de AM aplicadas a estimativa de esforço podemos destacar: Árvores de Regressão e Classificação [Basgalupp et al. 2013]; Lógica Fuzzy [Ziauddin et al. 2013]; Redes Neurais Artificiais

[Attarzadeh and Ow 2010]; Algoritmos Genéticos [Benala et al. 2012] e Estimativa por Analogia [Mendes and Counsell 2000].

Apesar de ser crescente o número de soluções desenvolvidas utilizando AM para apoio a estimativa de esforço, tais soluções ainda não são utilizadas na prática, ou seja, pelas empresas de desenvolvimento de software [Jørgensen 2004]. Uma possível justificativa para isto é o fato de que os modelos de estimativa baseados em AM geralmente são gerados com base em um conjunto de amostra de projetos e tem sua eficácia demonstrada para esta amostra. Porém, ao serem empregados em contextos diferentes daqueles nos quais foram baseados, estes modelos nem sempre se mostram satisfatórios.

Estudos como [Mendes et al. 2014] investigam as vantagens e desvantagens de se projetar modelos "personalizados" para um determinado contexto, ou seja, construir modelos de estimativa para uma organização utilizando os dados daquela organização. A vantagem óbvia neste caso será o alinhamento deste modelo à realidade organizacional para o qual foi projetado, o que leva a uma maior precisão das estimativas. Por outro lado, construir modelos com base em dados de uma única organização tem como desvantagens, ou impedimentos, fatores como: (i) o tempo que levaria para uma organização construir uma base de dados de projetos (*dataset*) suficientemente grande para ser útil na construção de um modelo; (2) durante esse tempo, a organização poderia mudar algum aspecto (como as tecnologias empregadas), o que tornaria os projetos antigos pouco relevantes para as novas práticas.

Com base no contexto apresentado, verifica-se que um desafio científico atual no desenvolvimento de soluções de estimativa de esforço baseadas em AM não seria a construção de um algoritmo que se adapte a todos os contextos de desenvolvimento possíveis, mas sim "como atribuir um algoritmo mais adequado para um determinado problema", dada a existência de diversos algoritmos que se mostram satisfatórios para variadas situações.

Assim, visando apoiar a resolução deste desafio, este trabalho propõe a utilização de meta-aprendizado para a seleção de algoritmos de estimativa de esforço baseados em AM, de acordo com a base de dados organizacional. De maneira genérica, o meta-aprendizado estuda como sistemas de aprendizado podem aumentar sua eficiência através da experiência. O objetivo é entender como a própria aprendizagem pode se tornar flexível de acordo com o domínio ou tarefa em estudo [Vilalta and Drissi 2002]. Com isso, pretende-se determinar sob quais condições cada algoritmo é mais apropriado, possivelmente ampliando o entendimento do mesmo e levando a sugestões de uso mais adequadas [de Souza 2010].

O restante desde documento está dividido da seguinte maneira: na Seção 2 é apresentada uma visão geral da proposta e na Seção 3 são apresentadas as considerações finais do trabalho.

2. Visão Geral da Proposta

Na Figura 1, adaptada de [de Souza 2010], é apresentado o processo genérico de meta-aprendizado que será aplicado nesta pesquisa:

Inicialmente são adquiridas as Bases de Dados (B) de projetos de desenvolvimento de software, cujos dados contenham métricas significativas para a estimativa de

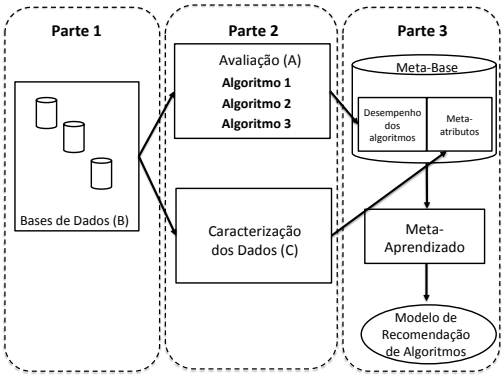


Figura 1. Visão Geral da Proposta

esforço. Essas bases podem ser tanto oriundas de repositórios públicos de dados (tais como PROMISE[T. Menzies and Turhan 2012] e ISBSG[ISBSG 2015]), quanto de empresas de desenvolvimento de software que desejem colaborar com a pesquisa (Parte 1).

Em um segundo momento (Parte 2), duas etapas são aplicadas para cada elemento de (B): a avaliação dos algoritmos, em (A), e a extração de características, em (C). Os algoritmos de AM a serem utilizados devem ser selecionados a partir da realização de uma revisão sistemática da literatura, onde serão identificadas e analisadas as soluções de AM já aplicadas para estimar esforço em desenvolvimento de software. Esta revisão também tem como objetivo identificar Medidas de Avaliação, tais medidas são utilizadas para avaliar o desempenho dos algoritmos e, a partir deste desempenho, determinar qual algoritmo deve-se utilizar para um determinado conjunto de dados. A caracterização do conjunto de dados procura extrair características presentes nos dados que possam influenciar o desempenho dos algoritmos de AM. Posteriormente, tais características serão utilizadas para recomendar os algoritmos de AM mais promissores para um conjunto de dados com tais características.

A terceira etapa (Parte 3) prevê a associação da Avaliação (A) com a Caracterização (C) para cada Base de Dados (B). São obtidos meta-exemplos, formados por meta-atributos de entrada e meta-atributos alvo. O conjunto de meta-exemplos é denominado meta-base. Para induzir então o mapeamento entre meta-atributos de entrada e meta-atributos alvo, aplica-se um algoritmo de AM, referido como meta-aprendiz. Por meio dele, pode-se utilizar o meta-conhecimento obtido do processo de aprendizagem e realizar, por fim, a recomendação de algoritmos no contexto de meta-aprendizado.

Em relação à metodologia aplicada na pesquisa, além da revisão sistemática já mencionada, serão realizados exaustivos experimentos *in-silico* para validação da solução de meta-aprendizado proposta, onde a solução será testada em relação ao seu desempenho individualmente e comparativamente a outras soluções de apoio à Estimativa de Esforço. Por fim, espera-se também realizar a avaliação da solução proposta, a partir da realização de um estudo de caso em uma organização de desenvolvimento de software.

3. Conclusão

Este artigo apresentou a proposta de uma solução de meta-aprendizado para realização de estimativa de esforço em desenvolvimento de software. Esta solução visa recomendar o

melhor algoritmo de AM para uma determinada base de dados, possibilitando, com isso, a adequação da solução a um determinado contexto organizacional.

Como resultado, pretende-se que a pesquisa desenvolvida contribua não apenas para a ampliação do conhecimento científico referente à meta-aprendizado no contexto de Engenharia de Software, mas também estimule seu uso no contexto real de desenvolvimento de software para estimar esforço.

Referências

Attarzadeh, I. and Ow, S. H. (2010). Proposing a novel artificial neural network prediction model to improve the precision of software effort estimation. In *Bio-Inspired Models of Network, Information, and Computing Systems*, pages 334–342.

Basgalupp, M. P., Barros, R. C., da Silva, T. S., and de Carvalho, A. C. (2013). Software effort prediction: A hyper-heuristic decision-tree based approach. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 1109–1116. ACM.

Benala, T., Dehuri, S., Satapathy, S., and Madhurakshara, S. (2012). Genetic algorithm for optimizing functional link artificial neural network based software cost estimation. In *Proceedings of the International Conference on Information Systems Design and Intelligent Applications*, volume 132 of *Advances in Intelligent and Soft Computing*, pages 75–82. Springer Berlin Heidelberg.

de Souza, B. F. (2010). *Meta-aprendizagem aplicada à classificação de dados de expressão gênica*. PhD thesis, Instituto de Ciências Matemáticas e de Computação – USP, São Carlos, SP, Brasil.

ISBSG (2015). International software benchmark and standards group.

Jørgensen, M. (2004). A review of studies on expert estimation of software development effort. *Journal of Systems and Software*, 70(1):37–60.

Mendes, E. (2014). *Practitioner's Knowledge Representation: A Pathway to Improve Software Effort Estimation*. Springer Publishing Company, Incorporated.

Mendes, E. and Counsell, S. (2000). Web development effort estimation using analogy. In *Software Engineering Conference, 2000. Proceedings. 2000 Australian*, pages 203–212. IEEE.

Mendes, E., Kalinowski, M., Martins, D., Ferrucci, F., and Sarro, F. (2014). Cross- vs. within-company cost estimation studies revisited: An extended systematic review. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, EASE '14, pages 12:1–12:10, New York, NY, USA. ACM.

Pressman, R. S. (2011). *Engenharia de Software*. McGraw-Hill, São Paulo, 7 edition.

T. Menzies, B. Caglayan, E. K. J. K. F. P. and Turhan, B. (2012). The promise repository of empirical software engineering data.

Vilalta, R. and Drissi, Y. (2002). A perspective view and survey of meta-learning. *Artificial Intelligence Review*, 18(2):77–95.

Ziauddin, S. K., Khan, S., and Nasir, J. A. (2013). A fuzzy logic based software cost estimation model. *International Journal of Software Engineering and Its Applications*, 7(2):7–18.

Sistema de Recomendação de APIs na Engenharia de Software

Luisa Hernández, Heitor Costa

Departamento de Ciência da Computação - Universidade Federal de Lavras - MG - Brasil

luisahernandezr@dcc.ufla.br,heitor@dcc.ufla.br

Resumo. *A maioria dos projetos de software depende de bibliotecas externas para alcançar seus objetivos. Essas bibliotecas são conhecidas como Application Programming Interfaces (APIs). Neste estudo, é apresentada uma metodologia que permitirá realizar uma análise empírica de Sistemas de Software quanto à utilização de APIs e apoiar o desenvolvimento de sistemas de software com qualidade. Essa metodologia permitirá a recomendação de APIs em sistemas que utilizem APIs. Recomendações para desenvolvedores na fase inicial do projeto também serão consideradas.*

Abstract. *Most software projects depend on external libraries to achieve their goals. These libraries are known as Application Programming Interfaces (APIs). In this study, we present a methodology to reach empirical analysis software systems on the use of APIs and thus support the development of software systems with quality. This methodology will let to recommend APIs in systems using APIs. Recommendations to developers in the initial phase of the project will be also considered.*

1. Introdução

Desenvolvimento de software não é uma tarefa fácil. Prova disso é a existência de várias propostas metodológicas que afetam componentes e aspectos do processo de desenvolvimento [Canós *et al*., 2003]. Entre os diferentes componentes, encontram-se as bibliotecas de software conhecidas como Interfaces de Programação de Aplicativos (*Application Programming Interfaces* - APIs). Uma API é a interface para uma entidade de software reutilizável utilizada por vários clientes e que pode ser distribuída separadamente do código ambiente [Robillard *et al*., 2013].

Quase todos os sistemas de software dependem de funções reutilizáveis fornecidas pelas APIs e o desenvolvimento de software moderno é inseparável desse reúso [Duala-Ekoko; Robillard, 2012]. Entre as principais vantagens de utilizar APIs está o fato de prevenir que os desenvolvedores reconstruam recursos existentes [Teyton *et al*., 2013], proporcionando uma forma rentável para construir sistemas de software com melhora na (1) produtividade dos programadores, fornecendo variedade de funções desejadas e (2) qualidade de software, porque são geralmente "bem-testadas" e utilizadas por grande quantidade de usuários (desenvolvedores) [Sun *et al*., 2011].

No entanto, para Engenheiros de Software, é difícil selecionar de forma eficaz as APIs durante o desenvolvimento e verificar a utilização correta depois da construção do sistema. Por causa do aumento do tamanho e da quantidade de APIs, os desenvolvedores devem frequentemente aprender como usar as APIs desconhecidas [Acharya *et al*., 2007]. Isso significa que, antes de aproveitar os benefícios de uma API para determinadas tarefas, um desenvolvedor deve descobrir e entender o

comportamento e as relações entre os elementos de uma API [Duala-Ekoko; Robillard 2012]. Para ajudar os desenvolvedores, foi proposto LIBTIC, um motor automático de busca de desenvolvedores especialistas em APIs Java utilizando mineração no repositório de software GitHub [Teyton *et al*., 2013a]. No entanto, contratar especialistas em APIs requer investimento, disponibilidade e recursos humanos que podem não estar dentro do escopo do projeto de desenvolvimento. Por outro lado, para analisar as APIs utilizadas por um conjunto de sistemas de software e sugerir APIs para desenvolvedores (sistema alvo da recomendação deve utilizar APIs), foi apresentada uma metodologia que combina técnicas de Mineração de Regras de Associação e de Filtragem Colaborativa [Thung *et al*., 2013].

Neste estudo, o objetivo é apresentar uma metodologia de recomendação de APIs para desenvolvedores com sistemas em estado inicial (podem ou não ter código, mas se tiver, não usam APIs) e com sistemas em desenvolvimento que usam APIs. Este artigo está organizado da seguinte forma. Breve apresentação de técnicas para sistemas de recomendação está na Seção 2. A metodologia proposta e adotada para o desenvolvimento de um sistema de recomendação é discutida na Seção 3. Considerações finais são apresentadas na Seção 4.

2. Sistemas de Recomendação

Sistemas de recomendação para Engenharia de Software *(Recommendation Systems for Software Engineering - RSSEs)* fornecem itens de informação valiosos para tarefas de engenharia de software em determinados contextos. Maior parte desses sistemas surge para ajudar desenvolvedores em diversas atividades, desde reúso de código a escrever relatórios eficazes de *bugs* [Robillard *et al*., 2010]. Hoje em dia, mais esforços são necessários para realizar essas atividades por causa de novas tecnologias, componentes e ideias que desenvolvedores são continuamente introduzidos [Robillard *et al*., 2014].

Entre as vantagens do uso de RSSEs, podem ser citadas diminuição do esforço, aumento da produtividade nas tarefas da engenharia de software e auxílio nas tomadas de decisões [Robillard *et al*., 2014]. Para desenvolver esses sistemas, são comumente utilizadas diferentes técnicas que dependem do contexto do problema e dos itens a serem recomendados: i) **Filtragem Baseada no Conteúdo**: fundamentada nas características ou palavras-chave dos itens e nas preferências do usuário; e ii) **Filtragem Colaborativa**: utilizada para recomendar itens baseando-se em outros usuários que interagem com o sistema. A Filtragem Colaborativa será utilizada neste estudo e há dois principais métodos [Breese *et al*., 1998]: i) **Baseado na Memória**, utiliza uma base de dados inteira para fazer recomendações; e ii) **Baseado no Modelo**, utiliza uma parte da base de dados para aprender um modelo e predizer as preferências dos usuários.

3. Metodologia

Como parte dos requisitos para elaboração de uma metodologia para auxiliar desenvolvedores e engenheiros de software com a recomendação de APIs necessárias, serão utilizados, pelo menos, 1.000 sistemas de software (repositório *P*). A relação entre esses sistemas e as categorias de aplicação será identificada, portanto precisam ser consideradas categorias estabelecidas pelo próprio repositório com a finalidade de evitar critérios de classificação subjetivos. Desse modo, *SourceForge*[1] foi selecionado por

[1] http://sourceforge.net

sugerir 10 categorias principais. Os sistemas selecionados seguem os critérios: i) ser desenvolvido em Java; ii) pertencer a uma categoria de aplicação do S*ourceForge*; iii) ter, pelo menos, 10.000 linhas de código; e iv) ter *status*[2] "Produção/Estável" ou "Maduro". Na Figura 1, é apresentada uma metodologia para recomendar APIs, com duas fases: **Fase A)** Recomendação de APIs para os desenvolvedores iniciando o desenvolvimento de um sistema de software (podem ou não ter código, mas se tiver, não usam APIs) e consideram uma categoria de aplicação; e **Fase B)** Recomendação de APIs para os sistemas de software em desenvolvimento que utilizam APIs e que podem ser classificados em uma categoria de aplicação.

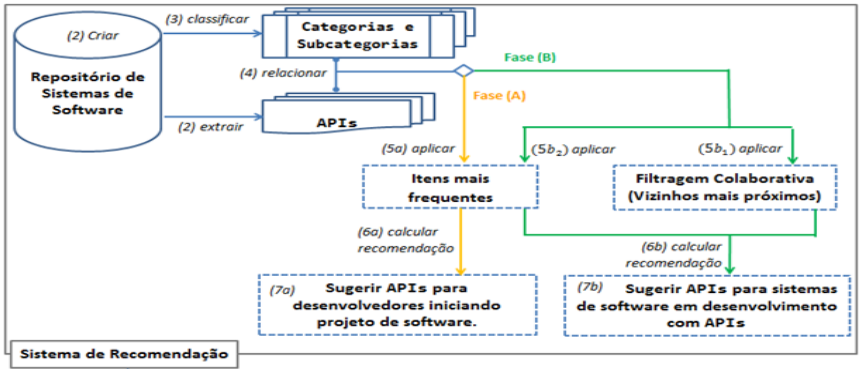


Figura 1. Metodologia Proposta para Sistema de Recomendação de APIs

Por causa do contexto do problema deste estudo e das necessidades da **Fase A**, técnicas de recomendação não são uma boa escolha, pois não têm APIs para relacionar. No entanto, há a possibilidade de recomendar APIs aos desenvolvedores determinando a popularidade e identificando APIs mais frequentes para cada categoria do repositório *P* e relacionando esses resultados às categorias consideradas pelo desenvolvedor alvo de recomendação. Quanto à **Fase B**, será utilizada a filtragem colaborativa baseada em memória. Isso implica que a similaridade será estabelecida entre os projetos do repositório *P* e o sistema alvo de recomendação (considerando categorias), obtendo os sistemas mais próximos. Em seguida, será estabelecida a similaridade entre as APIs do sistema alvo e as APIs dos sistemas mais próximos, gerando uma lista de APIs a recomendar. Nessa fase, poderá ser feita a união entre a lista gerada por itens mais frequentes e a lista gerada pela técnica de filtragem colaborativa.

Para avaliar os resultados, serão criados dois repositórios de teste (Teste A - T_A e Teste B - T_B) com os sistemas alvo de recomendação segundo as necessidades de cada fase. Para simular o cenário da **Fase A**, serão selecionados n sistemas de software do repositório principal. Para esses sistemas, serão retiradas as APIs guardando essas informações para aplicar as medidas de avaliação (*Precision*, *Recall* e *Fallout*) entre essa lista e as APIs geradas após a execução do sistema de recomendação. Quanto à simulação do cenário da **Fase B**, serão coletados os sistemas de teste dentro do repositório T_B. Para isso, será necessário selecionar n sistemas de software do repositório principal e retirar algumas APIs, obtendo mesma quantidade de APIs para cada sistema de software. Essa quantidade estará limitada segundo o sistema de software com menor uso de APIs. As informações sobre as APIs removidas serão guardadas para aplicar as medidas de avaliação (*Precisão*, *Recall* e *Fallout*) entre essa lista e as APIs geradas após a execução do sistema de recomendação.

[2] *Status* - Estado do sistema fornecido pelo repositório *SourceForge* que indica se ele está inativo, estável, em versão Beta, etc.

4. Considerações Finais

Os Sistemas de Recomendação têm sido estudados e desenvolvidos com mais frequência na área Engenharia de Software para apoiar os engenheiros na tomada de decisões de tarefas, componentes, pessoal, código, etc. Esses sistemas permitem o incremento da produtividade nas atividades de desenvolvimento e fornecem suporte aos desenvolvedores. Por outro lado, um componente indispensável no desenvolvimento de software é a API. As APIs são funções reutilizáveis em sistemas de software que permitem melhorar a produtividade e a qualidade do software desde que sejam escolhidas de forma correta para suprir as necessidades, mas a escolha de APIs é uma tarefa difícil para os desenvolvedores.

Como resultado deste trabalho, é apresentada uma proposta da metodologia de recomendação automatizada de APIs para sistemas de software em desenvolvimento que usam APIs e para desenvolvedores que apenas consideram as categorias de aplicação correspondentes. Entre os benefícios, espera-se apoiar os Engenheiros de Software na tomada de decisões sobre quais APIs utilizar em seus projetos. Assim, espera-se ter: i) aumento na produtividade dos desenvolvedores, pois serão reutilizadas funções, evitando a reconstrução e, consequentemente, minimização do tempo de desenvolvimento; e ii) aumento na qualidade do software, pois o uso de APIs populares é um indicador da qualidade das APIs e da qualidade dos sistemas de software. Finalmente, espera-se divulgar os avanços da pesquisa em futuras publicações.

Referências

Acharya, M.; Xie, T.; Pei, J.; Xu, J. (2007). Mining API Patterns as Partial Orders from Source Code : From Usage Scenarios to Specifications. In: Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering. pp. 25-34.

Breese, J. S., Heckerman, D., & Kadie, C. (1998). Empirical analysis of predictive algorithms for collaborative filtering. In Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence. Vol. 40, pp. 43-52.

Canós, J. H.; Letelier, P.; Penadés, C. (2003). Métodologías Ágiles en el Desarrollo de Software. In: DSIC Universidad Politécnica De Valencia. pp. 1-8.

Duala-Ekoko, E.; Robillard, M. P. (2012). Asking and Answering Questions about Unfamiliar APIs: An Exploratory Study. In: International Conference on Software Engineering. pp. 266-276.

Robillard, M., & Walker, R. (2014). An Introduction to Recommendation Systems in Software Engineering. In Robillard, M., Maalej, W., Walker, R., & Zimmermann, T. (Eds.), *Recommendation Systems in Software Engineering*, (pp. 1-11). Berlin, Heidelberg: Springer.

Robillard, M. P., Walker, R. J., & Zimmernann, T. (2010). Recommendation Systems for Software Engineering. IEEE Software, pp.80-86.

Robillard, M. P.; Bodden, E.; Kawrykow, D.; Mezini, M.; Ratchford, T. (2013). Automated API Property Inference Techniques. In: IEEE Transactions on Software Engineering, 39(5), pp. 613-637.

Sun, C.; Khoo, S.; Zhang, S. J. (2011). Graph-Based Detection of Library API Imitations. In: International Conference on Software Maintenance. pp. 183-192.

Teyton, C.; Falleri, J.-R.; Palyart, M.; Blanc, X. (2013). A Study of Library Migration in Java Software. The Computing Research Repository (CoRR). pp. 1-20.

Teyton, C.; Falleri, J.-R.; Morandat, F.; Blanc, X. (2013a). Find your Library Experts. In: Working Conference on Reverse Engineering. pp. 202-211.

Thung, F.; Lo, D.; Lawall, J. (2013). Automated Library Recommendation. In: Working Conference on Reverse Engineering. pp. 182-191.

An Automatic Approach to Detect Unusual Events in Software Repositories

Larissa Leite, Christoph Treude, Fernando Figueira Filho

[1] Departamento de Informática e Matemática Aplicada
Universidade Federal do Rio Grande do Norte (UFRN) – Natal, RN – Brazil

{larissaleite,ctreude,fmarquesfilho}@gmail.com

Abstract. *This work presents an automatic approach to detect unusual events in software repositories. The approach collects data from source code repositories and analyzes new commits based on historical data in order to detect unusual events that are displayed to developers and managers in an awareness tool.*

1. Introduction

Software development teams use source control repositories and issue trackers to support their development processes and activities. Managers can use information extracted from such tools to become aware of the team productivity, and plan the cost and time of future releases [Weiss et al. 2007], while developers are fed with insight into workspaces of other developers [Treude and Storey 2010]. Such insight is usually provided by awareness tools. Awareness is defined as *"an understanding of the activities of others, which provide context for your own activity"* [Dourish and Bellotti 1992]. Since the success of software projects largely depends on the effectiveness of communication and coordination, software development teams need to maintain awareness of different aspects ranging from overall project status and process bottlenecks to current tasks and incoming artifacts [Treude and Storey 2010].

Despite the large amount of previous work on awareness, no tool or approach has specifically focused on awareness of unusual events in a software repository. Being aware of unusual events can be useful in preventing errors, but also in alerting developers and managers of events that may require justification or that can affect the work of other developers, especially when they relate to significant changes to the project. The motivation of this work comes not only from input from fellow developers, but also from a recent survey with 156 GitHub users [Treude et al. 2015], in which developers reported the need to be aware of unusual events: *"Commits that take particularly long might be interesting. If a developer hasn't committed anything in a while, his first commit after a long silence could be particularly interesting, for example, because it took him a long time to fix a bug. Also, important commits might have unusual commit messages, [...] indicating that the developer was emotional about that particular commit"*. Another developer added: *"Changes to files that haven't been changed in a long time or changes to a large number of files, a large number of deletions, etc."*. This work proposes a mechanism to automatically detect such unusual events, and make managers and developers aware of them.

2. Method to Detect Unusual Events

To validate our proposal, we are using data from the software repository of Superintendência de Informática (SINFO), a company that belongs to Universidade Federal do

Rio Grande do Norte (UFRN). SINFO is responsible for the development and maintenance of all information systems used by employees, students, and faculties at the university. There are more than 75 developers, testers, and requirement analysts working at SINFO, using Apache Subversion (SVN) as their source control repository.

Unusual – or unexpected – events are, by definition, events that are not in conformance with normality. The first step to detect such events is to determine what is normal, which, of course, depends on the context being analyzed. In our work, this is done by gathering historical data from software repositories, which is usually associated with commits, tasks, and issues or bugs. At the current step, our work relies mainly on commit-related data to investigate unusual events.

What is unusual depends on the development context, team size, work dynamics, software process, development cycle, domain, product size and criticality, as well as the development model (community-based open source or industry). Although this work is being conducted in the context of a specific software development team, we believe that the unusual events discussed in this work can be generalized to other contexts since we have chosen events that are likely to occur in any software project that uses source control. Future work needs to investigate this further. In the following, we describe different rules we use to detect unusual events in source control repositories.

Long time between commits. Time between commits is considered an indicator of project activity [Kolassa et al. 2013]. One or two working days without any commits from the whole team might be caused by infrastructure problems. From the developer point of view, time between commits can be a measurement of how difficult a task is or how challenging it was to fix a bug. In some cases, long time between commits may also be a potential cause of conflict when trying to incorporate changes from a local working copy to the current version of the project. We determine whether the time between commits is "long" in a given development context by calculating the mean time between commits in a project and by considering all those times that are longer than the mean plus two standard deviations.

Large number of files touched (added, modified, deleted) in the same commit. The number of files touched in a commit is especially important when inspecting added or deleted files, since modifying existing files is much more common than their creation or removal. Adding or deleting a considerable amount of files might be an indicator of changes to the software architecture and it is probably a sign of a refactoring or work on a disruptive task, i.e., a task that changes a lot and "disrupts" the current code. Again, we use the historical mean and standard deviation to define "large".

Large number of code modifications (LOC, methods, code complexity). Commits with a notably large number of changes to lines of code (LOC), methods, or code complexity may represent significant modifications to the code base, similar to what happens when many files are added, modified, or deleted in a single commit. Thus, it is important to notify the development team of such changes.

Modification in files without their related files. It is very common to have strongly coupled files that are often changed together in software development. ROSE [Zimmermann et al. 2005] is a tool for Eclipse aiming to: (i) suggest and predict likely changes; (ii) prevent errors due to incomplete changes; and (iii) de-

tect coupling undetectable by program analysis. ROSE uses the Apriori Algorithm [Agrawal and Srikant 1994] to compute association rules. Unlikely ROSE, the approach used in this work does not try to predict changes or prevent errors, but rather to notify developers of a possibly incomplete change after the commit.

Modification in files changed by many different developers. Files created and/or changed by many developers are more bug-prone than files only maintained by one or two developers, which was evidenced by a tool called Seesoft [Balsiger 2010]. In the security domain, source code files changed by many developers are also more likely to have at least one post-release security vulnerability [Meneely 2011]. Therefore, modifications to these specific files are worth mentioning to the development team. Similar to the rules presented above, we use the historical mean and standard deviation to determine what is "many" in a given development context.

Modification in files that had many modifications. The number of modifications made to a file during the lifetime of the project is a commonly analyzed factor in the area of software maintenance. [Graves et al. 2000] state that the number of times that the code has been changed is a good indication of how many faults it will contain. Although this work does not aim to predict defects, a modification to a file that has already been changed many times can be an indicator for instability in the code, and, thus, it is considered an important notification to the development team.

Modification in files not modified in a long time. Files that have not been modified in a long period of time can indicate two things: either the code is stable or it has been "forgotten" and it is not up to date with the current version/status of the project (architecture, requirements, etc).

Our approach stores data about unusual events in a database and presents it to managers and developers using a web application. The Data Extraction process is supported by a Java project called UEMiner, which consists of three main components: (i) Miner, (ii) DataCollector, and (iii) DataAnalyzer. Miner is responsible for accessing a source code repository and for communicating with the database in order to save the retrieved data. The DataCollector component consists of an infrastructure to collect and prepare data to allow the identification of unusual events. Since the data related to each event is different, there is one collector for each type of unusual event. To store the relevant data, the DataCollector component creates spreadsheets, along with a few related statistics – especially mean and standard deviation. The DataAnalyzer component analyzes the data collected and prepared by DataCollector aiming to identify unusual events by: (i) getting statistical information about the historical data from the spreadsheets, and (ii) comparing such information with data of the current commit being analyzed. If an outlier is identified, the event is saved to the database, allowing it to be accessed by the web application. The DataAnalyzer component constantly monitors the repository for new commits, but the analysis process can be triggered by other factors, such as specific days of the week. The whole process is illustrated in Figure 1.

3. Future Work

The next steps for this work involve the evaluation of the proposed approach. Events are displayed to developers and managers in an awareness tool called UEDashboard, which shows the events in a notification feed. Managers and developers can provide feedback

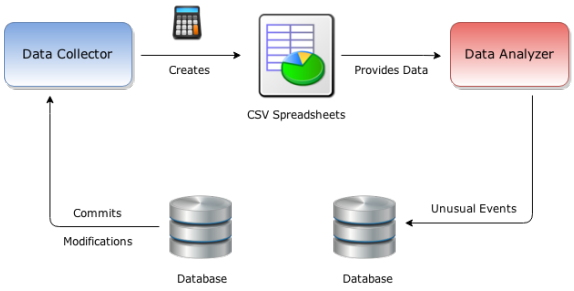


Figure 1. Process for Data Extraction and Analysis.

by classifying the event as useful or not useful, and they can write comments about each notification. In future work it might be possible for the tool to learn from these inputs to understand what is relevant for the development team and provide better notifications. We also intend to interview development teams to deeply understand what is behind unusual events. Additionally, we aim to apply our method to different development teams, since teams with different characteristics – team size, project age, development process – may require different analysis and could bring up various other types of unusual events. We also plan to extend our work beyond version control systems by additionally analyzing data from issue trackers, communication channels, and release management systems.

References

Agrawal, R. and Srikant, R. (1994). Fast algorithms for mining association rules in large databases. In *Proc. of the 20th Intl. Conf. on Very Large Data Bases*, pages 487–499.

Balsiger, M. (2010). Representing software features in the Eclipse IDE. Univ. of Bern.

Dourish, P. and Bellotti, V. (1992). Awareness and coordination in shared workspaces. In *Proc. of the Conf. on Computer-supported Cooperative Work*, pages 107–114.

Graves, T. L., Karr, A. F., Marron, J. S., and Siy, H. (2000). Predicting fault incidence using software change history. *IEEE Trans. on Software Engineering*, 26(7):653–661.

Kolassa, C., Riehle, D., and Salim, M. A. (2013). The empirical commit frequency distribution of open source projects. In *Proc. of the 9th Intl. Symp. on Open Collaboration*, pages 18:1–18:8.

Meneely, A. (2011). *Investigating the Relationship Between Developer Collaboration and Software Security*. PhD thesis. North Carolina State Univ.

Treude, C., Figueira Filho, F., and Kulesza, U. (2015). Summarizing and measuring development activity. *Submitted to Symp. on the Foundations of Software Engineering*.

Treude, C. and Storey, M.-A. (2010). Awareness 2.0: staying aware of projects, developers and tasks using dashboards and feeds. In *Proc. of the 32nd Intl. Conf. on Software Engineering*, pages 365–374.

Weiss, C., Premraj, R., Zimmermann, T., and Zeller, A. (2007). How long will it take to fix this bug? In *Proc. of the 4th Intl. Workshop on Mining Software Repositories*, page 1.

Zimmermann, T., Zeller, A., Weissgerber, P., and Diehl, S. (2005). Mining version histories to guide software changes. *IEEE Trans. on Software Engineering*, 31(6):429–445.

An Evaluation Model for Software Ecosystem Practice Improvement

Simone da Silva Amorim[1], John D. McGregor[2], Eduardo Santana de Almeida[3], Christina von Flach G. Chavez[3]

[1]1Departamento de Informtica - Instituto Federal da Bahia (IFBA)
CEP 40301-015 - Salvador - BA - Brazil

[2]Department of Computer Science - Clemson University
Clemson – USA

[3]Departamento de Computação
Universidade Federal da Bahia (UFBA) - Salvador, BA - Brazil

simone.amorim@ifba.edu.br, johnmc@clemson.edu, {esa,flach}@dcc.ufba.br

Abstract. *Many software ecosystems have achieved success in recent years. However, there is not an accepted quality process model which evaluates the essential practices that are commonly used in a software ecosystem. This paper presents the outline of a Ph.D. research aimed at developing an improvement model for software ecosystems. It summarizes a representation of the steps of the model and the identification of key factors that are relevant to the practices. The model is based on the practices and activities that can improve various facets of the ecosystem, modeled by three views of the ecosystem: community, business and technical. We also propose a research plan for developing such a model.*

1. Introduction

Software ecosystems are composed of one or more platforms and a set of applications built on top of these platforms, developed and used by a community. The community encompasses users, domain experts, sponsors and a set of internal and external developers all of whom interact in a shared market for software and services [Jansen et al. 2009]. In spite of the growing body of software ecosystem research, existing research does not address how and why they evolve over time. Empirical research is necessary to understand how different software ecosystems face the challenges of software evolution. Through this research, models can be defined and calibrated to use measurements to successfully guide the evolution processes in a software ecosystem. Such models would be useful to strategic decision makers within the context of software ecosystems, as a way of providing feedback to ecosystem managers.

Ecosystems evolve as the result of numerous forces acting on the organizations within the ecosystem. Evolution brings changes in the health of the ecosystem, in the quality of the products and practices in the ecosystem, and in the attractiveness of the ecosystems products and product development environment. We are investigating how these changes affect the quality of the ecosystem. By quality we mean the degree to which the practices meet the needs of process designers and the expectations of the organizations for the ecosystem to achieve its full potential.

This work presents an ongoing Ph.D. investigation on how to build a model of practice improvement for Software Ecosystems. We introduce the Evaluation Model for Software Ecosystems Practice Improvement (EMSEPI) and its three views on ecosystems: community, business and technical. These views represent different but overlapping aspects of the elements of a software ecosystem and their relationships.

2. Research Questions

The goal of this research can be stated as follows: to develop a model to evaluate the practice improvement in software ecosystems and to guide the evolution of these practices, giving support to all phases of the lifecycle.

Based on this goal, we state key questions to drive the research design. These are next detailed, together with a sketch on how we intend to address them.

RQ1. **How do ecosystems evolve as they grow more successful?** We intend to understand which issues have important influence on the evolution of software ecosystems. We investigate this question by means of the identification, description, and classification of evolution characteristics of software ecosystems, based on a detailed analysis on existing literature, and also based on the analysis of data from real-world projects.

RQ2. **How can the practices responsible for evolution be compared and evaluated in software ecosystems?** By analyzing the issues that impact on software ecosystem evolution, we develop a model to evaluate the improvement practices and to guide these practices during its evolution. This approach will focus to evaluate key factors considering the development process for the three ecosystems views: community, business and technical. To make such a model useful for practitioners, we intend to develop a tool to aid stakeholders to classify their ecosystem practices.

RQ3. **How can goals and recommended useful actions be established for software ecosystems?** Based on the classification of the practices improvement, we intend to build guidelines for the ecosystem to evolve their practices and achieve new levels of the practices improvement.

3. Evaluation Model for Software Ecosystem Practice Improvement

In this section, we present the Evaluation Model for Software Ecosystem Practice Improvement (EMSEPI). It is an attempt to evaluate a set of factors that influence the quality of practices for software ecosystems of all sizes and granularities. This model uses a key factors approach, based on progressive phases, which ecosystems can follow to achieve success. The key factors are classified considering the three ecosystems perspectives: community, business and technical. It is important for software ecosystem models to consider interactions among these views, since one of the more important ecosystem characteristics is the influence that each element exerts on other elements inside of the environment. Quantitative measures to define each progressive level of a key factor do not currently exist.

EMSEPI is organized in two steps that should be performed for each software ecosystem view and each key factor. In the First step, all activities related to key factor are identified. In the second step, the ecosystem is classified in accordance with PCT. The steps that analyze the key factors were defined through a literature review. Key factors

are organized in accordance with the ecosystem view. For example, in the community view, we present the following key factors: coordination, communication, interactions, experience of developers, and so on. For business view, we introduce: cost management, taking decisions, market share, return on investment, and so on. Lastly, for technical view, we introduce: change management, quality attributes, documentation, and so on. They cover several practices that are part of the dynamics ecosystem evolution.

In addition, EMSEPI uses a Phase Classification Template (PCT). The PCT shows a natural improvement progression for each key factor. They are sequentially ordered. The stages in the progression are named: minimal, low, medium and high. An ecosystem is evaluated against each key factor and placed along the progression for that factor. The classifications indicate how the organization can anticipate future changes. Due to space limitations, Figure 1 shows only an example of three key factors on the PCT for each view of this model: interactions, business decisions and design decisions.

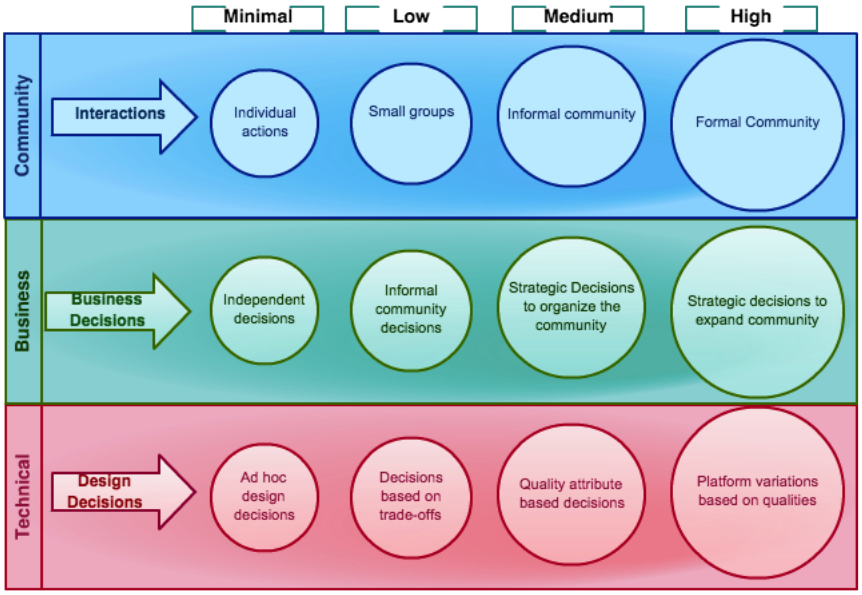


Figure 1. PCT - Phase Classification Template

4. Research Approach

We are developing EMSEPI by starting with the framework proposed by De Bruin *et al.* [De Bruin et al. 2005]. They introduced a methodology and outline the main phases of generic model development for a maturity model. However, this methodology will be adjusted to develop an improvement model. The model of De Bruin is composed of six phases of development: scope, design, populate, test, deploy and maintain.

During scoping, the stakeholders for the ecosystem determine the coverage of the desired model. The design phase defines the model architecture and aims to answer questions such as: why and how to apply the model, who should be engaged in the application of the model and what can be gained from the success of the model. The populate phase identifies key factors and a progressive series of improving levels for each. During this phase measures for each key factor are also determined. During the test phase, the model

is validated. The deploy phase applies an initial organizational application to verify that the model is sufficiently generalizable. Finally, the maintain phase manages the growth of the model and use.

During our initial scoping phase, we have identified twenty-one practices from our previous work that are important to a successful ecosystem [Amorim et al. 2013, Amorim et al. 2014a, Amorim et al. 2014b]. The model should be applied to software ecosystems that engage in platform-based software development, by providing a software platform to internal and external developers who are in service to a community of users and domain experts. Currently, we are iterating within the population phase of model development.

5. Conclusion and Future Work

We presented a high-level view on our PhD work. Our research objectives were motivated by the lack of models that evaluate the essential practices that are commonly used in a software ecosystem. A large number of quality models have been developed in the last years. However, none of them addressed the quality of the practices needed in a software ecosystem. In our research, we propose a model for software ecosystems practice improvement that can help consolidate knowledge and contribute to an ecosystem development. This allows stakeholders to model various aspects of software evolution within the context of software ecosystems and community, business, and technical views of that ecosystem. In this context, we formulated research questions and designed a research methodology based on studies about ecosystem evolution.

We hope this work will widen the understanding of the ecosystem practices improvement. We believe that our findings have practical implications to ecosystem management and development, as well as of a tool support. Moreover, we are going to develop this model in further detail by identifying more key practices. We plan to apply this model to real-world ecosystems for validation and, at the same time, extract relevant knowledge about the changes software ecosystems experience over time.

References

Amorim, S. d. S., de Almeida, E. S., and McGregor, J. D. (2013). Extensibility in ecosystem architectures: An initial study. In *Proceedings of the 2013 International Workshop on Ecosystem Architectures*, WEA 2013, pages 11–15.

Amorim, S. d. S., de Almeida, E. S., and McGregor, J. D. (2014a). Scalability of ecosystem architectures. In *Proceedings of the 11th Working IEEE/IFIP Conference on Software Architecture*, WICSA '14, pages 49–52.

Amorim, S. d. S., de Almeida, E. S., McGregor, J. D., and Chavez, C. v. F. G. (2014b). Flexibility in ecosystem architectures. In *Proceedings of the 2014 European Conference on Software Architecture Workshops*, ECSAW '14, pages 14:1–14:6.

De Bruin, T., Freeze, R., Kaulkarni, U., and Rosemann, M. (2005). Understanding the main phases of developing a maturity assessment model. In *Proceedings of the 16th Australasian Conference on Information Systems*, ACIS '05.

Jansen, S., Finkelstein, A., and Brinkkemper, S. (2009). A sense of community: A research agenda for software ecosystems. In *Proceedings of the 31st International Conference on Software Engineering: Companion Volume*, ICSE '09, pages 187–190.

Melhoria da Qualidade Interna de Software Orientado a Objetos Usando Medidas de Acoplamento e de Coesão

Danilo Santos, Antônio Resende, Heitor Costa

Departamento de Ciência da Computação - Universidade Federal de Lavras - MG - Brasil

danilobatista@posgrad.ufla.br, {tonio,heitor}@dcc.ufla.br

Resumo. *A manutenção é uma fase onerosa do ciclo de vida do software por serem realizadas sucessivas manutenções não planejadas, deteriorando a qualidade do software mais rapidamente. Dentre as causas da deterioração, as mudanças arquiteturais forçam a perda de coesão e o aumento do acoplamento afetando a qualidade de sistemas de software. Neste artigo, o objetivo é desenvolver uma metodologia para rearranjar a arquitetura de software pela movimentação de classes entre pacotes e determinar a "melhor" organização arquitetural para aumentar a qualidade interna e externa do software. Para suportar e avaliar a metodologia proposta, um apoio computacional (plug-in para Eclipse) será desenvolvido.*

Abstract*. Maintenance is a costly phase of the software life cycle due to perform successive unplanned maintenance, deteriorating software quality. Among the causes of deterioration, architectural changes force the loss of cohesion and increased coupling affecting the software quality. In this paper, aim is to develop a methodology to rearrange the software architecture for handling classes among packages and determining "best" architectural organization to increase internal and external software quality. To support and evaluate the proposed methodology, computational support will be developed (plug-in for Eclipse).*

1. Introdução

Engenheiros de Software esforçam-se para agregar qualidade a sistemas de software, propondo soluções de projeto adequadas ou identificando pontos no projeto que podem ter a qualidade aprimorada [Al Dallal, 2013]. Embora sistemas de software sejam desenvolvidos empregando as melhores práticas de desenvolvimento e provendo qualidade desde o início do seu ciclo de vida, eles podem deteriorar-se em decorrência de sucessivas manutenções [Stephen *et al*., 2001].

Portanto mesmo que um software, no início de seu ciclo de vida, atenda aos requisitos de qualidade perceptíveis aos usuários, com o passar do tempo ele precisa evoluir, para atender as novas necessidades do usuário e do ambiente, respeitando a lei da mudança contínua [Pressman; Maxim, 2014]. Caso isso não ocorra, ele será superado em qualidade e substituído por um concorrente existente no mercado [Parnas, 1994]. Para evitar essa deterioração na qualidade executa-se o processo de manutenção, entretanto não basta simplesmente executa-lo, existe a necessidade de realizá-lo seguindo padrões de projeto, para manter a qualidade inicial ou melhorá-la.

Não aplicar esses padrões pode ocasionar deterioração na qualidade do software, denominada *erosão de software* [Parnas, 1994]. Por isso a manutenção deve ser realizada de forma planejada e não estruturada, evitando-se corrigir erros ou inserir funções sem considerar fatores de qualidade ou padrões de projeto inicialmente utilizados. Manutenções não planejadas e não estruturadas tornam o código desorganizado, ilegível e propenso a falhas [Erdil *et al*., 2003]. Isso afeta atributos internos primordiais que conferem qualidade ao software (e.g., acoplamento e coesão) [Martin; McClure, 1983]. Acoplamento e coesão são atributos internos de software [Morasca, 2009] que podem ser afetados por uma manutenção mal feita.

Diversos autores concordam que pacotes de um sistema de software devem possuir baixo acoplamento e alta coesão, para obter um software com qualidade [Chen *et al*., 2002; Bavota *et al*., 2013; Al Dallal, 2013]. Como a manutenção afeta diretamente o acoplamento e coesão [Lanza e Marinescu, 2006], caso ela não seja executada seguindo padrões de projeto, tem-se redução na qualidade do software. Assim, uma reestruturação da arquitetura do software torna-se necessária, por exemplo, divisão/combinação de pacotes.

Neste trabalho, a reestruturação consistirá na movimentação de classes entre pacotes. Optou-se por essa abordagem em nível de pacotes, pois aperfeiçoando a estrutura dos pacotes pode-se reduzir os prejuízos ocasionados pela degradação de qualidade, reestabelecendo a "melhor" estrutura possível para o projeto. Essa degradação também pode ser notada nos valores de acoplamento e coesão, onde em decorrência da manutenção tem-se aumento da coesão diminuição do acoplamento [Lanza e Marinescu, 2006]. Uma estratégia a ser utilizada na reestruturação de movimentação de classes será buscar uma arquitetura, cujas medidas favoreçam baixo acoplamento e alta coesão entre pacotes. Assim, pode-se ter aumento/diminuição da quantidade de classes de um pacote e a possibilidade de acrescentar pacotes ao sistema.

Para dar apoio a essa reestruturação algoritmos/técnicas de Inteligência Artificial (IA) serão utilizados para buscar uma solução boa, não necessariamente ótima, automatizando a solução. Espera-se que esta automatização reduza a subjetividade da existente na tomada de decisão se uma reestruturação deve ser feita. Além disso, a avaliação por um humano é mais propensa a falhas e tornaria o processo menos eficiente, visto que demandaria mais tempo para ser executada e geraria menor quantidade de possíveis configurações de arquiteturas em comparação com as técnicas/heurísticas de IA.

2. Metodologia

A metodologia de desenvolvimento é apresentada na Figura 1. Inicialmente, será realizada uma Revisão Sistemática de Literatura (RSL), com o objetivo de saber quais são as medidas de software existentes na literatura. O resultado dessa RSL fornecerá bases sólidas para determinar as medidas a serem utilizadas na metodologia proposta, as quais avaliarão os atributos de acoplamento e coesão de sistemas de software.

Em seguida, serão estudadas técnicas/heurísticas de IA (*Simulated Annealing*, Redes Neurais Artificiais, Lógica *Fuzzy*, etc) para identificar qual é a mais apropriada para ser utilizada na metodologia proposta. Essas técnicas/heurísticas de IA possuem papel fundamental nessa metodologia, pois permitirão que as "melhores decisões" sejam

tomadas a cada passo, maximizando as chances de sucesso e diminuindo o tempo de execução.

Com as técnicas/heurísticas de IA e as medidas de acoplamento e de coesão definidas, um plug-in para Eclipse será desenvolvido. Esse plug-in irá utilizar uma técnica/heurística de IA e, com base nos valores de acoplamento e coesão dos pacotes de software, alterará a arquitetura do software, tendo como saída uma sugestão de arquitetura de software melhorada.

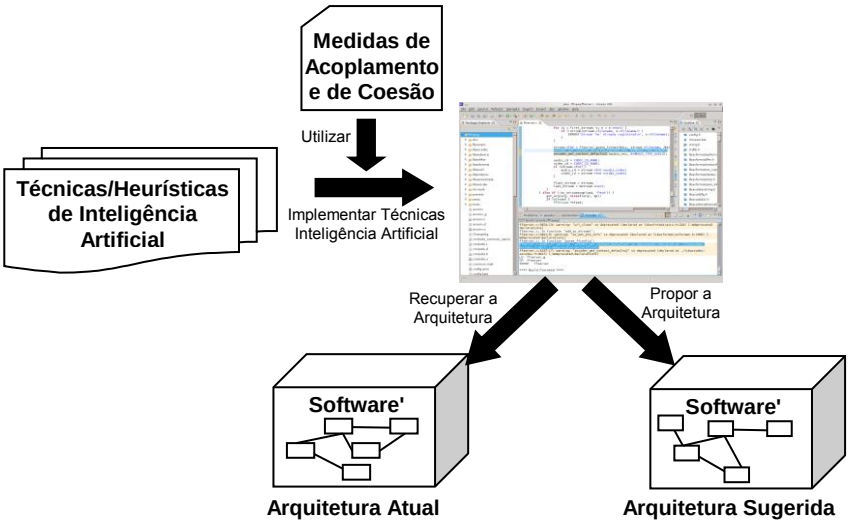


Figura 1. Metodologia de Desenvolvimento

3. Trabalhos Relacionados

Grande esforço tem sido dedicado por diversos autores para realizar trabalhos que contribuam na melhoria da qualidade de software, deteriorada durante a manutenção, por meio de alterações estruturais no projeto do software. Uma técnica para remodelagem dos pacotes utilizando medidas estruturais e semânticas para decompor o pacote em unidades menores e mais coesas foi proposta [Bavota, 2013]. Com a utilização dessa técnica, melhorias podem ocorrer na coesão dos pacotes do software sem deterioração do acoplamento, tendo sido validada por meio de um estudo empírico.

Em outro trabalho [Abdeen, 2009], foi implementada uma técnica e um algoritmo baseados em uma heurística de busca para otimizar a conectividade interpacotes de sistemas de software orientados a objetos. Nessa heuristica, pacotes são decompostos, a partir do conhecimento do acoplamento e da coesão, minimizando a conectividade ciclomática entre os pacotes. Uma análise foi realizada para verificar como o refatoramento afeta o acoplamento e a coesão [Du Bois *et al*., 2004]. A partir dessa análise, foram sugeridas orientações de como identificar oportunidades para aplicar o refatoramento de software. Concluiu-se que a utilização de técnicas de refatoramento, por exemplo, substituição/movimentação de métodos em conjunto de classes sob análise de acoplamento e de coesão, possibilita atingir melhorias na qualidade.

Diferentemente dos trabalhos citados anteriormente, nesta investigação pretende-se utilizar o acoplamento e a coesão de software como medidas norteadoras das melhorias da arquitetura, apoiada em técnicas/heuristicas de IA. As classes serão

movidas automaticamente entre os pacotes, enquanto a coesão e o acoplamento serão medidos e acompanhados para se identificar a melhor configuração de arquitetura.

4. Resultados Esperados

Espera-se que a metodologia proposta seja útil para aumentar a qualidade interna de sistemas de software (diminuir acoplamento e aumentar coesão). Para realizar a avaliação da metodologia proposta, será utilizado um conjunto de medidas de acoplamento e coesão diferente do conjunto de medidas utilizado para a condução do processo de reestruturação.

Essa reestruturação será aplicada em sistemas de software presentes em repositórios de sistemas *open-source*, pois há necessidade de ter acesso ao código fonte. Os sistemas a serem analisados deverão atender alguns critérios, por exemplo, quantidade de classes, quantidade de linhas de código, sistemas atuais (2014 em diante), aceito pela indústria e constantemente atualizado.

5. Referências

Abdeen, H.; Ducasse, S.; Sahraoui, H.; Sahraoui, H. Automatic Package Coupling and Cycle Minimization. In: WCRE. pp. 103-112, 2009.

Al Dallal, J. Object-Oriented Class Maintainability Prediction Using Internal Quality Attributes. I: Information and Software Technology. pp. 2028-2048, 2013.

Bavota, G.; Lucia, A.; Marcus, A.; Oliveto, R. Using Structural and Semantic Measures to Improve Software Modularization. In: Emp. Softw. Engineering. pp.901-932.2013.

Chen, Z.; Zhou, Y.; Xu, B.; Zhao, J.; Yang, H. A Novel Approach to Measuring Class Cohesion Based on Dependence Analysis. In: International Conference on Software Maintenance. pp. 377-384, 2002.

Du Bois, B.; Demeyer, S.; J. Verelst. Refactoring-Improving Coupling and Cohesion of Existing Code. In: Working Conference on Reverse Engineering. 2004.

Erdil, K.; Finn, E.; Keating, K.; Meattle, J.; Park, S. Software Maintenance as Part of the Software Life Cycle. Department of Computer Science, Tufits University: Comp180: Software Engineering Project, 2003.

Lanza M., e Marinescu R. Object-oriented metrics in practice: using software metrics to characterize, evaluate, and improve the design of object-oriented systems. Springer Science & Business Media, 2006.

Martin, J.; McClure, C. L. Software Maintenance: The Problem and Its Solutions. Englewood Cliffs, NJ.: Prentice Hall Professional Technical Reference, 1983.

Morasca, S. A Probability-based Approach for Measuring External Attributes of Software Artifacts. In: International Symposium on Empirical Software Engineering and Measurement, pp. 44-55, 2009.

Parnas, D. L. Software Aging. In: ICSE. pp. 279-287, 1994.

Pressman, R.; Maxim, B. Software Engineering: A Practitioner's Approach. 976p. 2014.

Stephen, E. G.; Graves, A. F.; Marron, J. S.; Mockus, A. Does Code Decay? Assessing the Evidence from Change Management Data. In: Transactions on Software Engineering. pp. 1-12, 2001.

Realization

Promoted by

Diamond Sponsor

Gold Sponsors:

Support:

2nd Latin-American School on Software Engineering
Campus do Vale - Instituto de Informática - UFRGS
Av. Bento Gonçalves 9500 Bl. IV - Porto Alegre - Brazil - CEP: 91501-970